Windows Applications

A windows application is an application that runs on windows desktop. Windows applications will have graphical user interface (GUI). Because of GUI, designing the application will be easy and fast. A windows application in vb.net contains a windows form (Win Form in short) by default and you can add any number of win forms to a windows application. A win form is saved in two files, a file with extension **.cs** and a file with extension **.designer.cs.** The **.cs** file contains code written by the programmer and **.designer.cs** file contains the automatically generated code for the design of the form by vs.net. As design related code and code written by the programmer are separated in to two different files, form is created as a partial class.

To design the win form, controls will be used. All the controls that can be used to design the win form are available in toolbox and are called windows controls. To design the windows form, you have to drag and drop controls from toolbox on to the form. Shortcut to open toolbox is **Control+Alt+X.** Every control is a class and all windows controls classes are inherited from a class called **control.** All classes related to windows controls are available in the namespace **system.windows.forms.** When you place a control on the form, an instance is created for the class of that control. Every control has its own **properties, methods** and **events.**

A **property** is a characteristic feature of the control that determine the behavior and appearance of the control. For example, **backcolor** property of a control determines the background color for the control and **enabled** property determines whether the control responds to events. Properties of a control can be accessed using **properties window.** This window can be opened by using the shortcut **F4** after selecting a control. This window provides two columns, where first column displays the property name and second column is used to set a value for the property. Properties are classified into **design time properties** and **runtime properties.** Design time properties are the properties that are available in properties window and can be set a value at design time. Runtime properties are the properties that are not available in properties window and can be accessed only at runtime.

Methods are the functions that perform an action against the control. Methods of a control are accessible only at runtime and there is no special window like properties window for methods.

Events are the special methods that are automatically invoked when user make an action against the control. Events are used to write the code for windows application. Code of the windows application was completely dependent on events. Hence programming in windows applications is called as **event driven programming.** To create an event for a control, first open its properties and within the properties window select events button to list the events of that control and double click on the event to create an event method in which you can write the code. Every control has one default event and when you have to write the code in default in event, just double click on that control.

When you have multiple forms in the windows application, then to execute a particular form, you have to set that form as startup form and for this open **program.cs** available in the project from solution explorer and within **Main** change **Application.Run()** statement by specifying the name of the form you want to execute as argument.

Windows Form

A windows form is used to design and write the code for windows application. Win form has **design** and **code** interfaces. To switch between **design** and **code** interfaces, use the shortcut **F7.** A windows form is also a class and it is created as partial class inherited from the class **form** available in **system.windows.forms** namespace. To refer to the current form in code, use the keyword **this.** A win form has the following important properties.

Sno	Property Name	Description
1	**Name**	Used to uniquely identify the control in code
2	**AcceptButton**	Used to set a button available on the form as accept button so that the code written for that button is automatically executed when user press enter key wherever the focus is on the form.
3	**AutoScroll**	Indicates whether scroll bars are displayed when a control on the form is out side the boundary of the form.
4	**BackColor**	Used to set a background color for the control.
5	**BackgroundImage**	Used to display an image in the background of the form.
6	**BackgroundImageLayout**	Used to set layout for the background image on the form.

7	**CancelButton**	Used to set a button available on the form as cancel button so that the code written for that button is automatically executed when user press escape key wherever the focus is on the form.
8	**ControlBox**	Indicates whether minimize, maximize and close buttons and system icon are displayed in the title bar of the form.
9	**Enabled**	Indicates whether the events of the control are raised.
10	**Font**	Used to set font for the text displayed in the control
11	**ForeColor**	Used to set Font Color for the control.
12	**Icon**	Used to set the icon to display in the title bar of the form.
13	**Location**	Used to set position of the control from left and top.
14	**Size**	Used to set width and height of the control.
15	**Text**	Used to set a caption for the form.
16	**WindowState**	Indicates the initial state of the form when it was displayed for the first time. It has three possible values., **minimized, maximized** and **normal.**

A win form has following important methods.

Sno	Method Name	Description
1	**Close()**	Used to close the form.
2	**Hide()**	Used to hide the form.
3	**Show()**	Used to display the form as non-modal window
4	**ShowDialog()**	Used to display the form as modal window.

Text Box

Text box is used to accept almost any data from the user except binary data i.e. audio and video and images. A text box has the following important properties.

Sno	Property Name	Description
1	**AcceptsReturn**	Indicates whether return character is accepted as input when **multiline** property is set to true.
2	**AcceptsTab**	Indicates whether tab character is accepted as input when **multiline** property is set to true.
3	**CausesValidation**	Indicates whether validation events are raised for the Previous control in tab order.

4	**CharacterCasing**	Indicates whether all characters in text box are left alone or converted to uppercase and lowercase. It has three possible values, **Normal, Upper** and **Lower.**
5	**MultilIne**	Indicates whether multiple lines are allowed in a text box.
6	**Lines**	An array of lines in text box when **multiline** property is set to true.
7	**MaxLength**	Indicates maximum number of characters allowed in a text box.
8	**PasswordChar**	Used to set a character as password character to hide the password when text box is used to accept password.
9	**ReadOnly**	Used to make a text box as read only.
10	**ScrollBars**	Indicates whether only horizontal or only vertical or both or no scrollbars are displayed when **multiline** property is set to true.
11	**TabIndex**	Used to set tab order for the controls on the form.
12	**Text**	Used to access the text in text box
13	**TextAlign**	Used to set alignment for the text in text box. It has three possible values. **Left, right** and **center**
14	**WordWrap**	Indicates whether words are wrapped to new line when **mutiline** property is set to true.
Runtime Properties		
1	**SelectedText**	Returns the selected text in Text box
2	**SelectionLength**	Returns number of characters selected.
3	**SelectionStart**	Returns the position of the character from which selection was started.

Text box has the following important methods.

Sno	**Method Name**	**Description**
1	**Clear()**	Used to clear the text box.
2	**Cut()**	Moves the selected text in text box to windows clipboard.
3	**Copy()**	Copies the selected text in text box to windows clipboard.

4	**Paste()**	Pastes the content of windows clipboard at cursor position in text box.
5	**Focus()**	Takes the focus on to the text box.
6	**Select(start,n)**	Selects **n** number of characters starting from **start.**
7	**SelectAll()**	Selects entire text in the text box.
8	**DeselectAll()**	Deselects entire text in the text box.
9	**Undo()**	Undoes the last change made to the text box.
10	**Redo()**	Redoes the last change made to the text box.

Button

Button control is used to initiate an action like save data or delete data from database. Button has the following important properties.

Sno	Property Name	Description
1	**Image**	Used to set an image to be displayed on button.
2	**ImageAlign**	Used to set alignment for the image on button.
3	**Text**	Used to specify a caption for the button. To provide an access key for the button, precede the character in text with **&.**

Label

Label is used to display static text on the form and it has the following important properties.

Sno	Property Name	Description
1	**Image**	Used to set an image to be displayed on label.
2	**ImageAlign**	Used to set alignment for the image on label.
3	**Text**	Used to specify a caption for the label. To provide an access key for the label, precede the character in text with **&.** Access key of a label will take the focus on to the next control in tab order for the label.

Events of the Windows Controls

Every control has its own events and events are used to write the code for windows application. Every event has two arguments; **sender** and **e. sender** contains the information regarding the control that raises the event and **e** contains data related to the event. For all the events type of **sender** is **object** and type of **e** varies depending on the event. The following set of events is common for almost every windows control.

Sno	Event Name	Description
1	**Click**	Will be raised when user clicks on the control with any button of the mouse. In this event it is not possible to determine which button of the mouse was clicked by the user. Type of second argument **e** is **EventArgs.**
2	**MouseClick**	Will be raised when user clicks on the control with any button of the mouse. In this event you can determine which button of the mouse was clicked by the user. Type of second argument **e** is **MouseEventArgs.**
3	**DoubleClick**	Will be raised when user double clicks on a control with any button of the mouse. In this event it is not possible to determine which button of the mouse was double clicked by the user. Type of second argument **e** is **EventArgs.**
4	**MouseDoubleClick**	Will be raised when user double clicks on a control with any button of the mouse. In this event it is possible to determine which button of the mouse was double clicked by the user. Type of second argument **e** is **MouseEventArgs.**
5	**Enter**	Will be raised when the focus enters in to the control and type of second argument **e** is **EventArgs.**
6	**Leave**	Will be raised when the focus leaves the control and type of second argument **e** is **EventArgs.**
7	**Validating**	Will be raised immediately after the **leave** event and is used to write validation code. Type of second argument **e** is **CancelEventArgs.** In this event you can cancel navigation of cursor to next control when validation was failed.
8	**Validated**	Will be raised immediately after the **validating** event

		and is also used to write validation code. Type of second argument **e** is **EventArgs.** In this event you can't cancel navigation of cursor to next control when validation was failed.
9	**KeyDown**	Will be raised when any key of the keyboard was hold down. This event is used to handle the keys that doesn't have ASCII value and have keycode. Type of second argument **e** is **KeyEventArgs.**
10	**KeyUp**	Will be raised when any key of the keyboard was released. This event is used to handle the keys that doesn't have ASCII value and have keycode. Type of second argument **e** is **KeyEventArgs.**
11	**KeyPress**	Will be raised when any key of the keyboard was completely down. This event is used to handle the keys that have an ASCII value. Type of second argument **e** is **KeyPressEventArgs.**
12	**MouseDown**	Will be raised when a button of the mouse was down on the control and type of second argument **e** is **MouseEventArgs.**
13	**MouseUp**	Will be raised when a button of the mouse was up on the control and type of second argument **e** is **MouseEventArgs.**
		MouseDown and **MouseUp** events are used to draw graphics and to start and stop drag and drop operations.
14	**MouseEnter**	Will be raised when mouse pointer enters in to the control and type of second argument **e** is **EventArgs.**
15	**MouseLeave**	Will be raised when mouse pointer leaves the control and type of second argument **e** is **EventArgs.**
		MouseEnter and **MouseLeave** events are used to change the icon while performing drag and drop operations based on whether drop is possible.
16	**MouseMove**	Will be raised when mouse pointer moves over the control and type of second argument **e** is **MouseEventArgs.**
17	**MouseHover**	Will be raised when mouse pointer was placed over

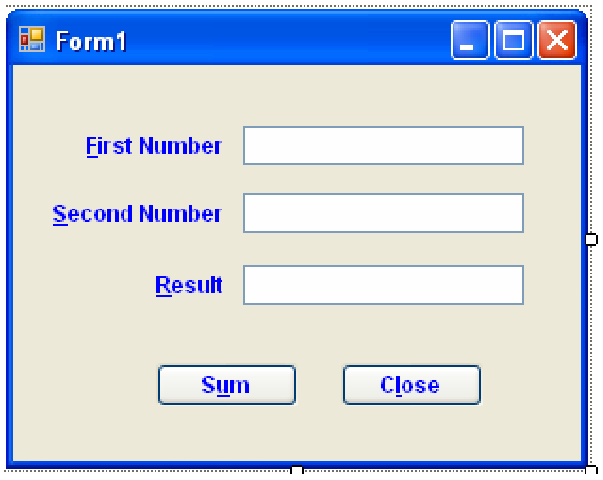
		the control constantly for at least one second and type of second argument **e** is **EventArgs.**

The following set of events is available only for a form.

Sno	Event Name	Description
1	**Load**	This is the default event for form and will be raised whenever the form is loaded. This event can be used to initialize variables and type of second argument **e** is **EventArgs.**
2	**FormClosing**	Will be raised when the form is about to be closed. This event can be used to write the code that needs to be executed while the form is closed and type of second argument **e** is **FormClosingEventArgs.**
3	**FormClosed**	Will be raised immediately after **FormClosing** event. This event can be used to write the code that needs to be executed while the form is closed and type of second argument **e** is **FormClosedEventArgs.**
4	**Resize**	Will be raised when form is resized and type of second argument **e** is **EventArgs.**

Example : The following example creates a win. Form with three text boxes and two button controls. When user enters two numbers in to first two text boxes and clicks on first button then sum has to be calculated for those two numbers and display the result in third text box.

1. Create a windows application and design the form as follows.



2. Set the following properties for the controls on the form.

TextBox1	Name	: TxtNum1
TextBox2	Name	: TxtNum2
TextBox3	Name	: TxtResult
Button1	Name	: BtnSum
	Text	: S&um
Button2	Name	: BtnClose
	Text	: C&lose

3. Double click on the sum button and write the following code in the click event of that button to calculate sum and display the result in result textbox.

```
private void BtnSum_Click(object sender, EventArgs e)
{
      TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) +
                            Convert.ToInt32(TxtNum2.Text)).ToString();
}
```

4. Double click on close button and write the following code in the click event of that button to close the form.

```
private void BtnClose_Click(object sender, EventArgs e)
{
     this.Close();
}
```

5. Run the Application with shortcut F5.

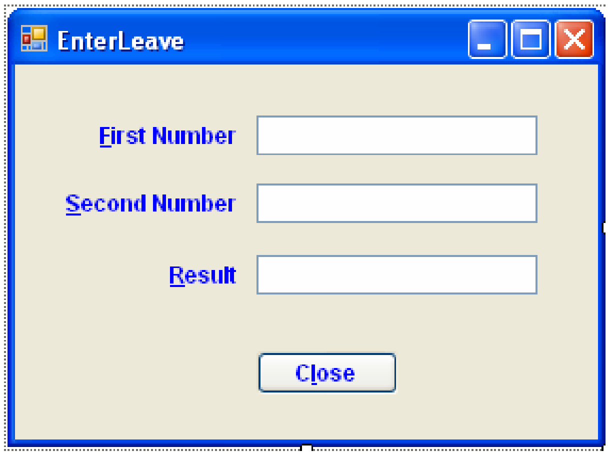
6. in the above example if you want to calculate sum when user press enter key and close the form when user press escape key, set sum button as **acceptbutton** for the form and close button as **cancelbutton** for the form.

Working With Enter And Leave Events

When focus enters in to a control then enter event will be raised and when a control lost focus then leave event will be raised. These events can be used to write the code that needs to be executed when a control got and lost the focus.

Example : in the above example when you want to calculate sum when the second text box lost focus and without clicking on sum button then do the example as follows.

1. Add another win. Form to the project and design it as follows.



2. Set following properties to the controls on the form.

TextBox1	Name	: TxtNum1
TextBox2	Name	: TxtNum2
TextBox3	Name	: TxtResult
Button	Name	: BtnClose
	Text	: C&lose

3. Choose leave event of second textbox txtnum2 by opening properties window, clicking on events button and then double clicking on leave event and write the following code in it.

```
private void TxtNum2_Leave(object sender, EventArgs e)
{
      TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) +
                              Convert.ToInt32(TxtNum2.Text)).ToString();
}
```

4. Double Click on close button and write the following code in its click event to close the form.

```
private void BtnClose_Click(object sender, EventArgs e)
{
      this.Close();
}
```

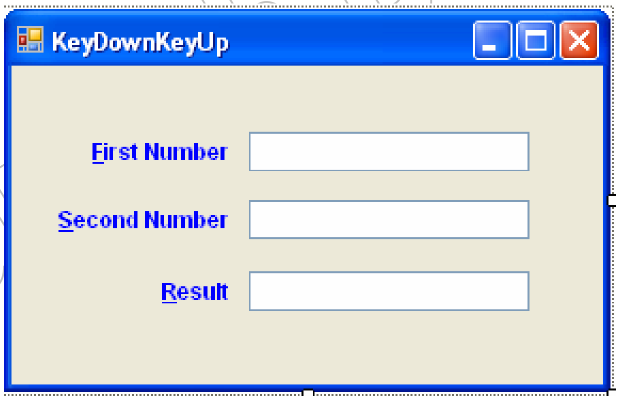
5. Run the application with shortcut F5.

Working With KeyDown And KeyUp Events

Keydown and keyup events are used when you want to write the code based on the keys having keycode but not ASCII value. The second argument **e** for keydown and keyup events is of type **keyeventargs** and it has a property **keycode** that contains the keycode of the key pressed by the user on the keyboard and this property is used to write the code in either keydown or keyup event. When you are writing the code in keyboard events of the form, then the **keypreview** property of the form must be set to **true,** because by default only keyboard events of the control that contains the focus will be raised and not the keyboard events of the form. When you set **keypreview** property true, then the keyboard events of both the control that contains the focus and form will be raised.

Example : The following example accepts two numbers from user and performs addition or subtraction or multiplication or division based on the function keys F1,F2,F3 and F4 respectively. F5 will clear the text boxes and Escape will close the form.

1. Add a win form to the project and design it as follows.



2. Set the following properties to the controls.

TextBox1	Name	: TxtNum1
TextBox2	Name	: TxtNum2
TextBox3	Name	: TxtResult

3. write the following code within the keydown event of the form. For creating the keydown event for the form, open properties of the form and within the properties window click on events button and then double click on keydown event.

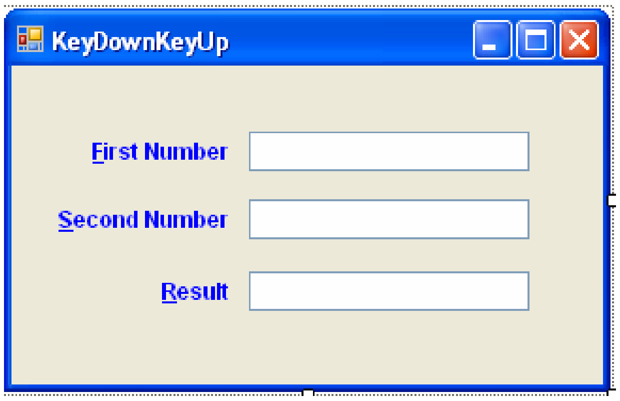
```
private void KeyDownKeyUp_KeyDown(object sender, KeyEventArgs e)
{
switch (e.KeyCode)
{
case Keys.F1:
      TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) +
                        Convert.ToInt32(TxtNum2.Text)).ToString();
      break;
case Keys.F2:
      TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) -
                        Convert.ToInt32(TxtNum2.Text)).ToString();
      break;
case Keys.F3:
      TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) *
                        Convert.ToInt32(TxtNum2.Text)).ToString();
      break;
case Keys.F4:
      TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) /
                        Convert.ToInt32(TxtNum2.Text)).ToString();
      break;
case Keys.F5:
      TxtNum1.Clear();
      TxtNum2.Clear();
      TxtResult.Clear();
      TxtNum1.Focus();
      break;
case Keys.Escape:
      this.Close();
      break;
}//switch
}//keydown event
```

4. Run the application with shortcut F5.

Control, shift and alt keys are called modifier keys. If you want to execute the code in keydown or keyup event only when user press a key along with modifier keys then use **modifiers** property of second argument **e** that contains the modifier keys pressed along with the key pressed on the keyboard.

Example : The following example accepts two numbers from user and performs addition or subtraction or multiplication or division based on the key combination Ctrl+F1, Ctrl+F2, Ctrl+F3 and Ctrl+F4 respectively. Ctrl+F5 will clear the text boxes and Escape will close the form.

1. Add a win form to the project and design it as follows.



2. Set the following properties to the controls.

TextBox1	Name	: TxtNum1
TextBox2	Name	: TxtNum2
TextBox3	Name	: TxtResult

3. write the following code within the keydown event of the form. For creating the keydown event for the form, open properties of the form and within the properties window click on events button and then double click on keydown event.

```
private void KeyDownModifiers_KeyDown(object sender, KeyEventArgs e)
{
if (e.Modifiers == Keys.Control)
{
switch (e.KeyCode)
{
      case Keys.F1:
            TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) +
                              Convert.ToInt32(TxtNum2.Text)).ToString();
            break;
      case Keys.F2:
            TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) -
                              Convert.ToInt32(TxtNum2.Text)).ToString();
            break;
      case Keys.F3:
            TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) *
                              Convert.ToInt32(TxtNum2.Text)).ToString();
            break;
      case Keys.F4:
            TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) /
                              Convert.ToInt32(TxtNum2.Text)).ToString();
            break;
      case Keys.F5:
            TxtNum1.Clear();
            TxtNum2.Clear();
            TxtResult.Clear();
```

```
                TxtNum1.Focus();
                break;
}//switch
}//if
if (e.KeyCode == Keys.Escape)
      this.Close();
}//keydown event
```

4. Run the application with shortcut F5.

5. if you want to check for the condition with the combination of more than one modifier keys then write the condition by specifying the modifier keys in parentheses separated by single pipe(|) symbol .

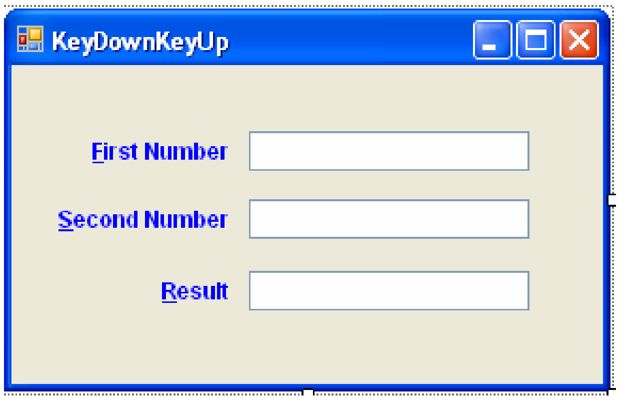
```
if (e.Modifiers == (Keys.Control | Keys.Alt))
{

}
```

Working with KeyPress Event

Keypress event is used when you want to write the code based on the keys having ASCII value. The second argument **e** for keypress event is of type **keypresseventargs** and it has a property **keychar** that contains the character pressed by the user on the keyboard and this property is used to write the code in keypress event. Within the keypress event it is not possible to access modifier keys. The second argument **e** in keypress event has a property **Handled** that can be used to exit from keypress event in middle. For this set **Handled** property to **true**.

Example : The following example accepts two numbers from user and performs addition or subtraction or multiplication or division based on the keys A, S, M and D respectively. C will clear the text boxes and X will close the form.

1. Add a win form to the project and design it as follows.



2. Set the following properties to the controls.

TextBox1	Name	: TxtNum1
TextBox2	Name	: TxtNum2
TextBox3	Name	: TxtResult

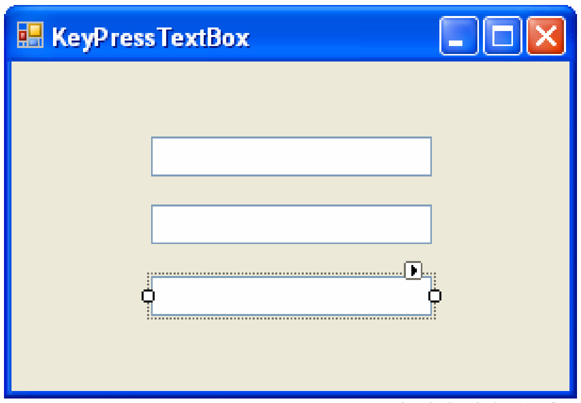
3. Write the following code within the keypress event of the form. For creating the keypress event for the form, open properties of the form and within the properties window click on events button and then double click on keypress event.

```
private void KeypressEvent_KeyPress(object sender, KeyPressEventArgs e)
{
switch (e.KeyChar)
{
      case 'A':
      case 'a':
            TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) +
                              Convert.ToInt32(TxtNum2.Text)).ToString();
            e.Handled = true;
            break;
      case 'S':
      case 's':
            TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) -
                              Convert.ToInt32(TxtNum2.Text)).ToString();
            e.Handled = true;
            break;
      case 'M':
      case 'm':
            TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) *
                              Convert.ToInt32(TxtNum2.Text)).ToString();
            e.Handled = true;
            break;
      case 'D':
      case 'd':
            TxtResult.Text = (Convert.ToInt32(TxtNum1.Text) /
                              Convert.ToInt32(TxtNum2.Text)).ToString();
            e.Handled = true;
            break;
      case 'C':
      case 'c':
            TxtNum1.Clear();
            TxtNum2.Clear();
            TxtResult.Clear();
            TxtNum1.Focus();
            e.Handled = true;
            break;
      case 'X':
      case 'x':
            this.Close();
            break;
}//switch
}//keypress event
```

Keypress event of the text box can be used to restrict the text box to accept only alphabets or only digits or only specified characters.

Example : The following example creates a form with three text boxes where first text box is restricted to alphabets, second textbox restricted to digits and third textbox to both alphabets and digits.

1. Add a win form to the project and design it as follows.



2. Set following properties to the control.

TextBox1	Name	: T1
TextBox2	Name	: T2
TextBox3	Name	: T3

3. Create keypress event for the textbox T1 and write the following code in it. To create keypress event for the textbox T1, open its properties and within the properties window click on events and then double click on keypress event.

```
private void T1_KeyPress(object sender, KeyPressEventArgs e)
{
      if (char.IsLetter(e.KeyChar) == false && e.KeyChar !=(char)Keys.Back)
      {
            e.Handled = true;
      }
}
```

4. Create keypress event for the textbox T2 and write the following code in it. To create keypress event for the textbox T2, open its properties and within the properties window click on events and then double click on keypress event.

```
private void T2_KeyPress(object sender, KeyPressEventArgs e)
{
      if(char.IsDigit(e.KeyChar)==false && e.KeyChar!=(char)Keys.Back)
      {
            e.Handled = true;
      }
}
```

5. Create keypress event for the textbox T3 and write the following code in it. To create keypress event for the textbox T3, open its properties and within the properties window click on events and then double click on keypress event.

```
private void T3_KeyPress(object sender, KeyPressEventArgs e)
{
      if (char.IsLetterOrDigit(e.KeyChar) == false
                                      && e.KeyChar != (char)Keys.Back)
      {
            e.Handled = true;
      }
}
```

6. Run the application using the shortcut F5

Error Provider

Error provider control is used to associate an error to a control when validation was failed. It is invisible at run time and it is available in **components** tab of the tool box. It has the following important properties.

Sno	Property Name	Description
1	**BlinkRate**	Rate in milliseconds at which error icon blinks
2	**BlinkStyle**	Indicates whether error icon blinks when an error is set. It has three possible values; **BlinkIfDifferentError, AlwaysBlink** and **NeverBlink.**
3	**Icon**	Used to specify the icon to display for error provider.

Error provider has the following important methods.

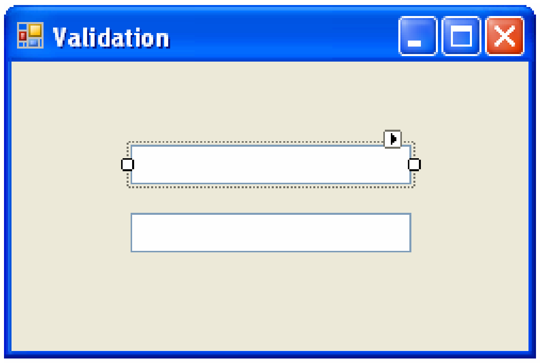
Sno	Method Name	Description
1	**SetError(Control,Message)**	Sets error description for specified control.
2	**Clear()**	Clears all settings associated with error provider.

Working with Validation Events

Almost every windows control in .net has two validation events, **validating** and **validated**. These events will be raised immediately after leave event and are used to write validation code for the controls. Within the **validating** event the data type of second argument **e** is **CancelEventArgs** and it has a property **Cancel** by setting which to true will cancel the navigation of cursor to next control. But this is not available in **validated** event.

Example : The following example takes two text boxes and an error provider control on the form and performs validation on first textbox for a minimum of 7 characters in the textbox.

1. Add a win form to the project and design it as follows.



2. Set following properties to the controls.

TextBox1	Name : T1
TextBox2	Name : T2
ErrorProvider1	Name : Ep

3. Write the following code in validating event of the textbox T1. To create validating event for T1, open its properties and within the properties window click on events button and then double click on validating event.

```
private void T1_Validating(object sender, CancelEventArgs e)
{
      if (T1.Text.Trim().Length < 7)
      {
            Ep.SetError(T1, "A Minimum Of 7 Characters Are Required");
            e.Cancel = true;
      }
      else
      {
            Ep.Clear();
      }
 }
```

4. Run the application by using the shortcut F5.

Working with FormClosing Event

When form is closed, two events **FormClosing** and **FormClosed** will be raised. These events can be used to write the code that needs to be executed while the form is closed. The type of second argument **e** in **FormClosing** event is **FormClosingEventArgs** and it has a property **cancel** by using which you can cancel the closing of the form. But this is not possible in **FormClosed** event and in **FormClosed** event type second parameter e is **FormClosedEventArgs**.

Example : The following example displays a message to the user "Do You Want To Save The Changes" while the form is closed.

1. Add a win form to the project and write the following code in FormClosing event of the form. To create FormClosing event for the form, open its properties and within the properties window click on events and then double click on the FormClosing event.

```
private void FormClosingEvent_FormClosing(object sender,
                                          FormClosingEventArgs e)
{
DialogResult Res;
Res = MessageBox.Show("Do You Want To Save The Changes", "Confirmation To
            Save", MessageBoxButtons.YesNoCancel, MessageBoxIcon.Question);
if (Res == DialogResult.Yes)
      MessageBox.Show("Changes Saved");
else if (Res == DialogResult.No)
      MessageBox.Show("Changes Not Saved");
else
      e.Cancel = true;
}
```

2. Run the application using the shortcut F5.

Working with MouseClick And MouseDoubleClick Events

When you have different code to be executed for different buttons of the mouse, then you have to use **MouseClick** and **MouseDoubleClick** events. Type of second argument **e** for these events is **MouseEventArgs** and it has a property **Button** that can be used to determine which button of the mouse was clicked or double clicked.

Example : The following shows how to write the code in MouseClick and MouseDoubleClick events by creating MouseClick event for the form.

1. Add a win form to the project and write the following code in MouseClick event of that form. To create MouseClick event for the form, open its properties and within the properties window click on events and then double click on the MouseClick event.

```
private void MouseClickEvent_MouseClick(object sender, MouseEventArgs e)
{
      switch (e.Button)
      {
            case MouseButtons.Left:
                  MessageBox.Show("Left Button Clicked");
                  break;
            case MouseButtons.Right:
                  MessageBox.Show("Right Button Clicked");
                  break;
            case MouseButtons.Middle:
                  MessageBox.Show("Middle Button Clicked");
                  break;
        }
 }
```

2. Run the application using the shortcut F5.

To work with modifier keys **Ctrl, Alt** And **Shift** within the mouseclick and mousedoubleclick events, use the property **ModifierKeys** of **Control** class that is the base class for every windows control in .net.

```
private void MouseClickEvent_MouseClick(object sender, MouseEventArgs e)
{
      if (Control.ModifierKeys == Keys.Control)
      {
            switch (e.Button)
            {
                  case MouseButtons.Left:
                        MessageBox.Show("Control + Left Button Clicked");
                        break;
                  case MouseButtons.Right:
                        MessageBox.Show("Control + Right Button Clicked");
                        break;
                  case MouseButtons.Middle:
                        MessageBox.Show("Control + Middle Button Clicked");
                        break;
            }
      }
 }
```

Check Box

Check boxes are used to accept Boolean value from user or to provide multiple options to the user from which he can select one or more options. It has the following important properties.

Sno	Property Name	Description
1	**Appearance**	Indicates whether check box is displayed like button or normal check box.
2	**Checked**	Indicates whether check box is checked or unchecked.
3	**ThreeState**	Indicates whether check box has three states, **checked, unchecked** and **indeterminate**
4	**CheckState**	Indicates the state of the check box when **threestate** property is set to true.
5	**Text**	Used to specify a caption for the check box.

Check box has the following important events.

Sno	Event Name	Description
1	**CheckedChanged**	This is the default event for check box and will be raised when check box is checked or unchecked.
2	**CheckStateChanged**	Will be raised when check box **threestate** property is set to true and state of the check box is changed.

Radio Button

Radio buttons are used to provide multiple options to the user from which user can select only one option. Radio button has the following important properties.

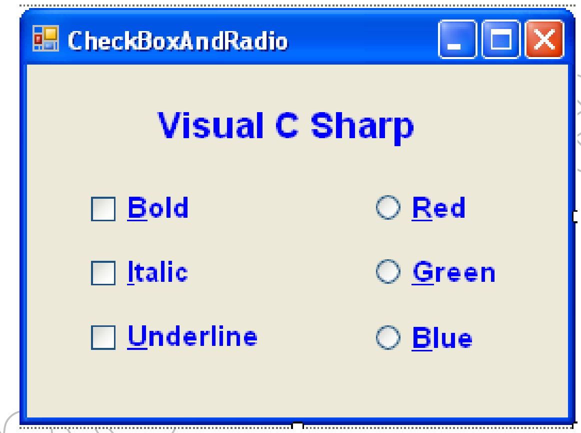
Sno	Property Name	Description
1	**Appearance**	Indicates whether radio button is displayed like button or normal radio button.
2	**Checked**	Indicates whether radio button is selected.
3	**Text**	Used to specify a caption for the radio button.

Radio Button has the following Event.

Sno	Event Name	Description
1	**CheckedChanged**	This is the default event for radio button and will be raised when radio button is checked or unchecked.

Example : The following example takes a label on the form and three checkboxes for bold, italic and underline and three radio buttons for the colors red, green and blue. Whenever checkboxes are checked, label has to become bold italic and underline and whenever radio buttons are selected the color of the label must be changed.

1. Add a win form to the project and design it as follows.



2. Set following properties to the controls on the form.

Label1	Name	: L1
	Text	: Visual C Sharp
CheckBox1	Name	: ChkBold
	Text	: &Bold
CheckBox2	Name	: ChkItalic
	Text	: &Italic
CheckBox3	Name	: ChkUnderline
	Text	: &Underline
RadioButton1	Name	: RbtnRed
	Text	: &Red
RadioButton2	Name	: RbtnGreen
	Text	: &Green

RadioButton3	Name	: RbtnBlue
	Text	: B&lue

3. Create a method with the name ChangeStyle() within the form code as follows to change the style of the label based on status of the checkboxes bold, italic and underline.

```
public void ChangeStyle(bool B, bool I, bool U)
{
        Font F;
        if (B == true && I == true && U == true)
        {
        F = new Font(L1.Font, (FontStyle.Bold | FontStyle.Italic |
                                                    FontStyle.Underline));
        }
        else if (B == true && I == true && U == false)
        {
                F = new Font(L1.Font, (FontStyle.Bold | FontStyle.Italic));
        }
        else if (B == true && I == false && U == true)
        {
        F = new Font(L1.Font, (FontStyle.Bold | FontStyle.Underline));
        }
        else if (B == false && I == true && U == true)
        {
        F = new Font(L1.Font, (FontStyle.Italic | FontStyle.Underline));
        }
        else if (B == true && I == false && U == false)
        {
                F = new Font(L1.Font, FontStyle.Bold);
        }
        else if (B == false && I == true && U == false)
        {
                F = new Font(L1.Font, FontStyle.Italic);
        }
        else if (B == false && I == false && U == true)
        {
                F = new Font(L1.Font, FontStyle.Underline);
        }
        else
        {
                F = new Font(L1.Font, FontStyle.Regular);
        }
        L1.Font = F;
}
```

4. Within the checkedchanged event of the checkboxes call the method ChangeStyle() by passing checked status of all three checkboxes as arguments as follows. To create the checkedchanged event of the checkboxes double click on the checkbox.

```
private void ChkBold_CheckedChanged(object sender, EventArgs e)
{
ChangeStyle(ChkBold.Checked, ChkItalic.Checked, ChkUnderline.Checked);
}
```

```
private void ChkItalic_CheckedChanged(object sender, EventArgs e)
{
ChangeStyle(ChkBold.Checked, ChkItalic.Checked, ChkUnderline.Checked);
}

private void ChkUnderline_CheckedChanged(object sender, EventArgs e)
{
ChangeStyle(ChkBold.Checked, ChkItalic.Checked, ChkUnderline.Checked);
}
```

5. Write the following code within the checkedchanged event of the radio buttons to change the color of the the label whenever a radio button was selected. To create checked changed event for the control double click on the radio button.

```
private void RbtnRed_CheckedChanged(object sender, EventArgs e)
{
      L1.ForeColor = Color.Red;
}

private void RbtnGreen_CheckedChanged(object sender, EventArgs e)
{
      L1.ForeColor = Color.Green;
}

private void RbtnBlue_CheckedChanged(object sender, EventArgs e)
{
      L1.ForeColor = Color.Blue;
}
```

6. Run the application with the shortcut F5.

List Box

List box is used to provide multiple items to the user and allow the user to select one or more items. You can provide items in the list box either at design time or at runtime. List box has the following important properties.

Sno	Property Name	Description
1	**Items**	Used to specify the items to be displayed in list box.
2	**MultiColumn**	Indicates whether items in list box are displayed in multiple columns.
3	**SelectionMode**	Indicates whether user can select a single item or multiple items or no items. This has four possible values; **None, One, MultiSimple** and **MultiExtended**. **None :** not possible to select even one item.

		One : can select only one item. **MultiSimple :** can select multiple items and to select multiple items, no need to hold down either control or shift key. **MultiExtended :** can select multiple items and to select multiple items, need to hold down either control or shift key.
4	**Sorted**	Indicates whether items in list box are sorted in ascending order.
Runtime Properties		
1	**SelectedIndex**	Returns the index of the item selected in the list box.
2	**SelectedItem**	Return the selected item in the list box.
3	**SelectedIndices**	A collection of indices of all selected items in the list box.
4	**SelectedItems**	A collection of all selected items in the list box.

List Box has the following important Events

Sno	Event Name	Description
1	**SelectedIndexChanged**	This is the default event and will be raised when a new item was selected in the list box.

To add items to the list box at runtime, **items** collection of list box is used and **items** collection has the following important members.

Sno	Member Name	Description
1	**Add("Item")**	Used to add an item at the end of existing items in the list box.
2	**Insert(Index,"Item")**	Used to insert a new item at specified index.
3	**AddRange(Items)**	Used to add all items in one list box to another.
4	**Remove(item)**	Used to delete an item from list box by specifying the item.
5	**RemoveAt(index)**	Used to delete an item from list box by specifying the index.
6	**Clear()**	Used to delete all items from the list box.
7	**IndexOf(Item)**	Used to get the index of a specified item.

8	**Count**	Returns the total no. of items in list box.
9	**Item**	An array and returns an item at specified index.

Checked List Box

Checked list box is same as list box except that for every item in checked list box, a check box is displayed. All the properties, methods and events available for list box are available for checked list box. The members of items collection of list box are also available for items collection of checked list box. In addition to the members of the list box, checked list box has one additional event **ItemCheck** that will be raised whenever an item in the list box is checked or unchecked.

Combo Box

Combo box is the combination of text box and list box. It provides a list of items same as a list box and allows the user to select an item from the list or specify a new item in text box. In case of a list box it is not possible to specify a new item that is not available in list. Within a combo box you can select only one item. Combo box has the following important properties.

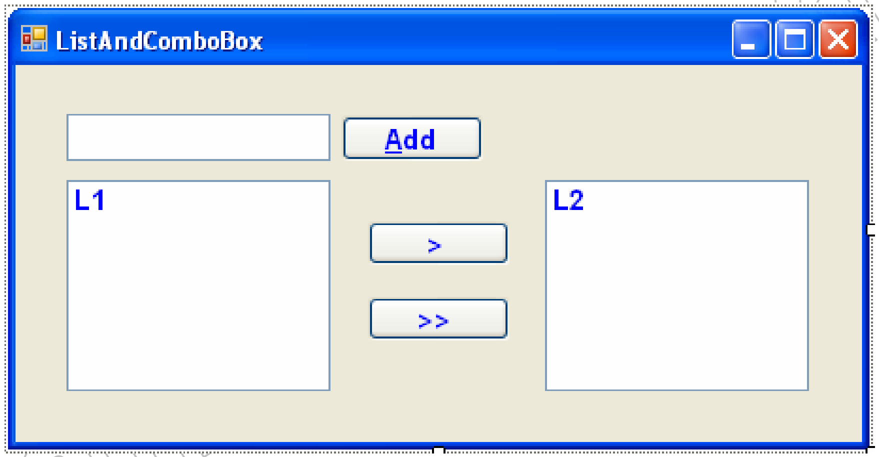
Sno	Property Name	Description
1	**DropDownStyle**	Used to specify the appearance and behavior of the combo box and has three possible values; **Simple, DropDown** and **DropDownList**. Both in simple and dropdown you can select an item from list or enter new item in text box. But in DropDownList you must select an item from list.
2	**Items**	Used to specify the list of items displayed in combo box and has same members as items collection of list box that can be used to add items to combo box at runtime.
3	**Sorted**	Indicates whether items in combo box are sorted in ascending order.
Runtime Properties		
1	**SelectedIndex**	Returns the index of the selected item in combo box.
2	**SelectedItem**	Returns the selected item in the combo box.

Combo Box has the following important events.

Sno	Event Name	Description
1	**SelectedIndexChanged**	This is the default event and will be raised when a new item was selected in the Combo box.
2	**DropDown**	Will be raised when the list portion of the combo box was dropped down.

Example : The following example takes a textbox and two list boxes on the form and allows the user to add items to first list box from text box and to second list box from first list box.

1. Add a win form to the project and design it as follows.



2. Set following properties to the controls on the form.

TextBox1	Name	: TxtItem
Button1	Name	: BtnAdd
	Enabled	: False
	Text	: &Add
ListBox1	Name	: L1
	SelectionMode	: MultiSimple
ListBox2	Name	: L2
Button2	Name	: BtnMove
	Enabled	: False
	Text	: >
Button3	Name	: BtnMoveAll
	Text	: >>

3. Write the following code in textchanged event of the textbox txtitem to enable or disable the add button. To create textchanged event for the textbox txtitem, double click on it.

```
private void TxtItem_TextChanged(object sender, EventArgs e)
{
        if (TxtItem.Text.Trim().Length > 0)
        {
                BtnAdd.Enabled = true;
        }
        else
        {
                BtnAdd.Enabled = false;
        }
}
```

4. Write the following code in click event of add button to add the item in textbox txtitem to listbox L1. to create click event for the add button, double click on it.

```
private void BtnAdd_Click(object sender, EventArgs e)
{
        L1.Items.Add(TxtItem.Text);
        TxtItem.Clear();
        TxtItem.Focus();
}
```

5. Write the following code within the selectedindexchanged event of the listbox L1 to enable or disable the move button. To create selectedindexchanged event for the listbox L1, double click on it.

```
private void L1_SelectedIndexChanged(object sender, EventArgs e)
{
        if (L1.SelectedItems.Count > 0)
        {
                BtnMove.Enabled = true;
        }
        else
        {
                BtnMove.Enabled = false;
        }
}
```

6. Write the following code within the click event of the move button to move all selected items in listbox L1 to the listbox L2 and delete them from L1. To create click event for the move button, double click on it.

```
private void BtnMove_Click(object sender, EventArgs e)
{
        foreach (object I in L1.SelectedItems)
        {
                L2.Items.Add(I);
        }
        for (int i = L1.SelectedItems.Count - 1; i >= 0; i--)
        {
                L1.Items.Remove(L1.SelectedItems[i]);
        }
}
```

7. Write the following code within the click event of moveall button to move all items in listbox L1 to the listbox L2. To create click event for the button moveall, double click on it.

```
private void BtnMoveAll_Click(object sender, EventArgs e)
{
        L2.Items.AddRange(L1.Items);
        L1.Items.Clear();
}
```

8. Run the application using the shortcut F5.

MonthCalendar

Monthcalendar control is used to display a calendar to the user and allow him to select a date or a range of dates. Monthcalendar has the following important properties.

Sno	Property Name	Decsription
1	**MinDate**	Used to specify the minimum date user can select in monthcalendar control.
2	**MaxDate**	Used to specify the maximum date user can select in monthcalendar control.
3	**ScrollChange**	Used to specify the number of months to navigate when user clicks on arrow keys in the title bar of monthcalendar. To navigate to immediate previous month or immediate next month regardless of scrollchange, click on very first date or very last date in monthcalendar.
4	**SelectionRange**	Used to access the range of dates selected in monthcalendar. This is a structure type and has two members; **start** and **end**. **Start** returns the date at which selection was started and **end** returns the date

		at which selection was ended. When user selects only one date then both **start** and **end** will have the same date.

Monthcalendar have the following important events.

Sno	Event Name	Description
1	**DateChanged**	This is the default event and will be raised when a new date was selected in monthcalendar.
2	**DateSelected**	Will be raised when a new date was selected and immediately after datechanged event.

DateTimePicker

Datetimepicker is used to accept date and time from the user. Datetimepicker appears like combo box and it provides an arrow key by clicking on which it displays monthcalendar to select a date. Within datetimepicker you can select only one date. Datetimepicker has the following important properties.

Sno	Property Name	Decsription
1	**Format**	Used to specify the format of date and time displayed in datetimepicker control. This has four possible values, **Long** for long date, **short** for short date, **Time** for time and **custom** to specify custom format.
2	**CustomFormat**	When **format** property is set to **custom** then to specify custom format, custom format property is used. To specify custom format the following format strings can be used. **dd –** Day of Month **ddd –** Abbreviated form of week (sun, mon,…) **dddd –** Complete name of the week (Sunday, Monday,…) **MM –** Two digital month **MMM –** Abbreviated form of Month (jan, feb, …) **MMMM –** Complete month name (January, February, ,,,) **yy –** Last two digits of the year **yyy –** Last three digits of the year **yyyy –** four digital year **hh –** hours of the day in 12 hour format **HH –** hours of the day in 24 hour format

		mm – minutes **ss –** seconds **tt –** to display AM/PM at the end of the time.
3	**MinDate**	Used to specify the minimum date user can select in datetimepicker control.
4	**MaxDate**	Used to specify the maximum date user can select in datetimepicker control.
5	**Value**	Used to access the date and time entered into datetimepicker control.

Datetimepicker control has the following important events.

Sno	**Event Name**	**Description**
1	**ValueChanged**	This is the default event and will be raised when a date and time in datetimepicker control is changed.

PictureBox

Picturebox is used to display images on the form. Picture box has the following important properties.

Sno	**Property Name**	**Description**
1	**InitialImage**	Used to specify the default image to display in picture box.
2	**Image**	Used to specify the image to display in the picture box.
3	**ErrorImage**	Used to specify the image to display when there is an error in loading the specified image.
4	**SizeMode**	Used to specify how the image will be placed and resized within the picture box. This has three possible values; **normal** for displaying only top left portion of the image when image is larger than picture box, **stretchimage** for stretching the image to the size of picture box when it is larger than or smaller than the picture box and **autosize** for changing the size of picture box to the size of imge.

Picture box has the following important method.

Sno	Method Name	Description
1	**Load()**	Used to load an image in to the picture box.

Default event for the picture box is **click**.

OpenFileDialog

This control is used to display open file dialog box and allow the user to open a file. This control is available in components tab of the toolbox and is invisible at runtime. Openfiledialog control has the following important properties.

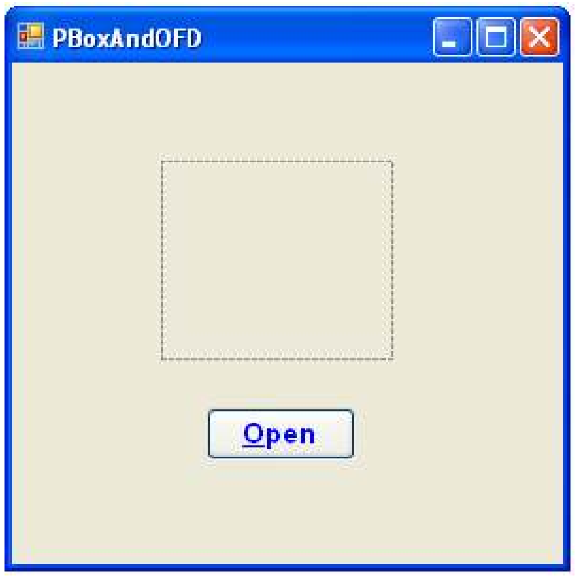
Sno	Property Name	Description
1	**InitialDirectory**	Used to specify the initial directory to display in open filedialog box.
2	**Filter**	Used to specify the filters to display within the open file dialog box. Syntax to specify the filter is as follows. **Word Doments|*.Doc|Text Files|*.Txt|All Files|*.***
3	**FileName**	Returns the name and path of the file selected by the user in openfiledialog box.
4	**MultiSelect**	Indicates whether user can select multiple files in openfiledialog.

Openfiledialog has one important method.

Sno	Method Name	Description
1	**Showdialog()**	Displays the open file dialog box and its return type is **dialogresult**.

Example : The following example takes a picturebox, openfiledialog control and a button control on the form and allows the user to open an image in to picture box by displaying open file dialog with the help of openfiledialog control.

1. Add a Win Form to the project and design it as follows.



2. Set following properties to controls on the form.

PictureBox	Name	: PB
	SizeMode	: StretchImage
Button	Name	: BtnOpen
	Text	: &Open
OpenFileDialog	Name	: Ofd

3. Write the following code within the click event of the open button control to display open file dialog box and display the selected image in picture box. To create click event for open button, double click on it.

```
private void BtnOpen_Click(object sender, EventArgs e)
{
      Ofd.InitialDirectory = @"C:\Windows\";
      Ofd.Filter = "Bitmap Files|*.Bmp|All Files|*.*";
      if (Ofd.ShowDialog() == DialogResult.OK)
      {
            PB.Load(Ofd.FileName);
      }
}
```

4. Run the application using the shortcut F5.

SaveFileDialog

Savefiledialog is used to display save file dialog box that can be used to save a file and it has the following important properties.

Sno	Property Name	Description
1	**AddExtension**	Indicates whether file extension is automatically added while saving the file.
2	**DefaultExt**	Used to specify the default extension to add to the file when **addextension** property is set to **true**.
3	**FileName**	Returns the name and path of the file specified by the user in save file dialog box.
4	**Initial Directory**	Used to specify the initial directory to display in open file dialog box.
5	**OverwritePrompt**	Indicates whether overwrite prompt is displayed when an existing file is being overwritten.
6	**ValidateNames**	Indicates whether file name given by the user is validated.

Savefiledialog also has the method **ShowDialog()** to display the save file dialog box.

FontDialog

Fontdialog is used to display font dialog box that can be used to change the font of a control and has the following important properties.

Sno	Property Name	Description
1	**ShowEffects**	Indicates whether effects like underline, strikethrough and color are displayed
2	**ShowColor**	Indicates whether color option is displayed in font dialog box.
3	**Font**	Returns the font selected by the user in font dialog box.
4	**Color**	Returns the color selected by the user in font dialog box.

FontDialog also has the method **ShowDialog()** to display the font dialog box.

<u>ColorDialog</u>

ColorDialog is used to disply color dialog box to the user from which user can select a color for a control on the form.

Sno	Property Name	Description
1	**AllowFullOpen**	Indicates whether define custom color button is enabled.
2	**FullOpen**	Indicates whether custom color section is by default opened.
3	**Color**	Returns the color selected by the user in color dialog box.

ColorDialog also has the method **ShowDialog()** to display the color dialog box.

<u>Timer</u>

Timer is used to repeatedly execute the code at specified interval of time. Timer is available in components tab of the toolbox and it is invisible at runtime and has the following important properties.

Sno	Property Name	Description
1	**Enabled**	Indicates whether timer responds to events. By default set to false.
2	**Interval**	Used to specify the time interval for the timer in milliseconds where 1000 milliseconds equals 1 second.

Timer has one important event **Tick** that will be raised whenever the time interval set for the timer was elapsed.

Example : The following example creates a form with a timer and a label where label will be scrolling with the help of timer.

1. Add a win form to the project and place a label and a timer control to the form and set following properties to them.

Label	Name	: L1
	Text	: Naresh Technologies, Ameerpet, Hyderabad
.Timer	Name	: Timer1
	Enabled	: True
	Interval	: 300

2. Write the following code in timer's tick event to make the label scrolling. To create tick event for the timer, double click on it.

```
private void timer1_Tick(object sender, EventArgs e)
{
      L1.Text = L1.Text.Substring(1) + L1.Text.Substring(0, 1);
}
```

3. Run the application using the shortcut F5.

Example : The following example makes a label blinking with the help of timer.

1. Add a win form to the project and place a label and a timer on it and set following properties to them.

Label	Name	: L1
	Text	: Naresh Technologies
Timer	Name	: Timer1
	Enabled	: True
	Interval	: 300

2. Write the following code within the tick event of the timer to make the label blinking. To create tick event for the timer double click on it.

```
private void timer1_Tick(object sender, EventArgs e)
{
      L1.Visible = !L1.Visible;
}
```

3. Run the application using the shortcut F5.

ProgressBar

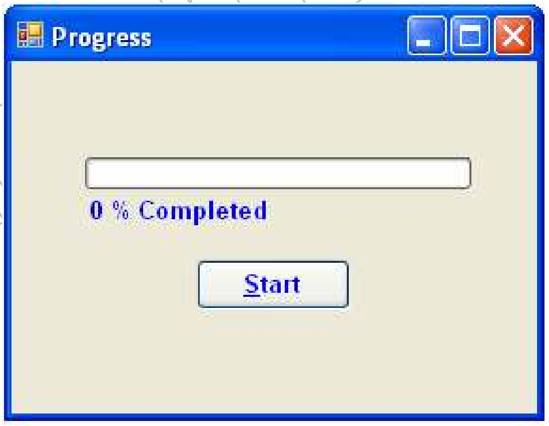
Progressbar is used to display the progress of an operation in your .net application and has the following important properties.

Sno	Property Name	Description
1	**Minimum**	Indicates the lower bound for the progress bar
2	**Maximum**	Indicates the upper bound for the progress bar.
3	**Value**	Returns the current value of the progress bar.
4	**Style**	Indicates the appearance of the progress bar and three possible values; **blocks, continuous** and **marquee**

To write the code for the progress bar, timer control will be used.

Example : The following example takes a progressbar, timer and button controls on the form and displays the progress of an operation that will be completed in 1 minute.

1. Add a win form to the project and design it as follows.



2. Set following properties to the controls on the form.

ProgressBar	Name	: P1
	Maximum	: 60
	Visible	: false
Label	Name	: L1
	Visible	: false

Button	Name	: BtnStart
	Text	: &Start
Timer	Name	: Timer1
	Enabled	: false
	Interval	: 1000

3. Write the following code in Start button click event to make progress bar and label visible and enable timer. To create click event for the start button double click on it.

```
private void BtnStart_Click(object sender, EventArgs e)
{
        P1.Visible = true;
        L1.Visible = true;
        timer1.Enabled = true;
}
```

4. Write the following code within the tick event of the timer control to display the progress of the operation and how much percentage it is completed. To create tick event of the timer, double click on it.

```
private void timer1_Tick(object sender, EventArgs e)
{
        P1.Value++;
        L1.Text = (P1.Value * 100 / P1.Maximum).ToString() + " % Completed";
        if (P1.Value >= P1.Maximum)
        {
                P1.Visible = false;
                L1.Visible = false;
                timer1.Enabled = false;
        }
}
```

5. Run the application using the shortcut F5.

MenuStrip

Menustrip is used to create menu bar for the windows application in .net. To create a menu bar for the windows application, follow the following steps.

1. Place menustrip on the form, which will be placed on component tray and a menu bar is displayed at the top of the form with a text box with the caption “type here”.
2. Click on type here to provide a caption for the menu. For example, **File**.

3. To provide menu items for the File menu, click on “type here” just below the menu and provide a caption for the menu item. Continue the same process to create all required menu items for the menu.
4. To create a separator for the menu, specify the caption for the menu item as “-“.
5. To create another menu, click on “type here” right to the first menu and follow the steps 3 and 4 to create all required menu items for that menu and continue the same process to create all required menus.

Menu items have the following important properties.

Sno	Property Name	Description
1	**Checked**	Indicates whether menu item behaves like check box.
2	**CheckOnClick**	Indicates whether check box type menu item is automatically checked or unchecked.
3	**CheckState**	Indicates whether check box is checked or unchecked.
4	**Image**	Used to specify an image for the menu item.
5	**ShortcutKeys**	Used to provide a shortcut key for the menu item.

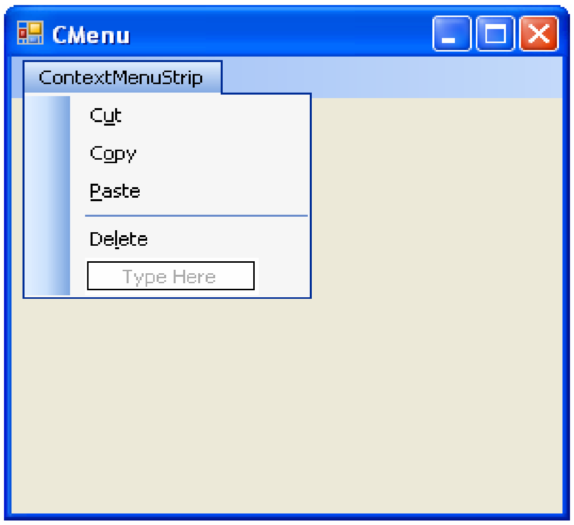
To write the code for menu items, use the **click** event of that menu item.

ContextMenuStrip

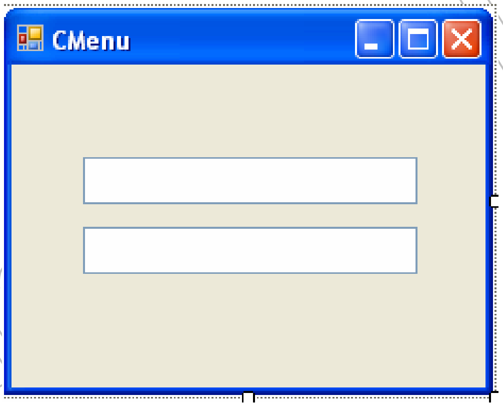
Contextmenustrip is used to create a context menu for the controls. Contextmenu will be displayed only when user right clicks on the control. Creating context menu is same as creating menu bar except that in menu bar multiple menus exists where a context menu contains only one menu. Properties of menu items in context menu are also same as properties of menu items in menu bar. Contextmenustrip has one important property **sourcecontrol** that returns the control on which user right clicks to display the context menu. Use the click event of menu items to write the code for context menu.

Example : The following example creates a context menu for textbox control to perform operations like cut, copy, paste and delete.

1. Add a win form to the project and create a context menu using contextmenustrip control as follows.



2. Design the form with two textboxes and set following properties to those textboxes.



ContextMenuStrip	Name	: CM
TextBox1	ContextMenuStrip	: CM
TextBox2	ContextMenuStrip	: CM

3. Write the following code within the click event of the menu items in contextmenustrip to perform cut, copy, paste and delete operations on the textbox on which user right clicks.

```
private void cutToolStripMenuItem_Click(object sender, EventArgs e)
{
        TextBox T1 = CM.SourceControl as TextBox;
        T1.Cut();
}
```

```
private void copyToolStripMenuItem_Click(object sender, EventArgs e)
{
      TextBox T1 = CM.SourceControl as TextBox;
      T1.Copy();
}

private void pasteToolStripMenuItem_Click(object sender, EventArgs e)
{
      TextBox T1 = CM.SourceControl as TextBox;
      T1.Paste();
}

private void deleteToolStripMenuItem_Click(object sender, EventArgs e)
{
      TextBox T1 = CM.SourceControl as TextBox;
      T1.SelectedText = "";
}
```

4. Run the application using the shortcut F5.

ToolStrip

Toolstrip control is used to create a tool bar for the windows application. In general a toolbar provides the shortcuts for the commands in menu bar. Toolstrip has the following important properties.

Sno	Property Name	Description
1	**AllowItemReorder**	Indicates whether items within the tool bar can be reordered by holding down the control key.
2	**ShowItemTooltips**	Indicates whether tool tips are displayed for the items in tool bar.
3	**Items**	Collection of items to be displayed in tool bar.
4	**Dock**	Used to specify the position of the tool bar on the form i.e. either **Top, Bottom, Left** or **Right**.

The following items can be added to a toolstrip control.

Sno	Item	Description
1	**Button**	Most of the items in a toolstrip are buttons. For a button, you can set the properties like **Name, Image,** and **TooltipText**. **Image** property is used to specify the image to display on the button and **TooltipText** property is used to specify the tool tip to display for

		the button. To write code for a button, use the **click** event of the button.
2	**DropDown Button**	This is button by clicking on which a menu is displayed. In case of this dropdown button, when user clicks on either button or arrow key, it displays the menu. For this you can set the properties like **Name, Image, TooltipText** and **DropDownItems**. **DropDownItems** property is used to specify the menu items to display when user clicks on dropdown button. To write the code for this item, use the click event of the menu items.
3	**SplitButton**	This is also a button that displays a menu when user clicks on the button. But in case of split button, menu will be displayed when user clicks on arrow key and executes the code written for the first menu item when user clicks on the button. For this you can set the properties like **Name, Image, TooltipText** and **DropDownItems**. **DropDownItems** property is used to specify the menu items to display when user clicks on dropdown button. To write the code for this item, use the click event of the menu items.
4	**ComboBox**	When you want to provide multiple items in tool strip from user can select one item, then use combo box. Important properties for combo box are **Name, Items** and **TooltipText**. **Items** property is used to specify the list of items to be displayed in combo box. **Selectedindexchanged** event is used to write the code for combo box.
5	**ProgressBar**	Used to display the progress of an operation within the tool strip instead of on the form. Properties that can be set for the progressbar are **Name, Minimum, Maximum, Value** and **TooltipText**. Timer control has to be used to write the code for the progress bar.
6	**Separator**	Used to provide a separator within the toolstrip.
7	**TextBox**	Used for displaying a text box on the toolstrip. Properties that can be set for a text box are **Name**

		and **TooltipText**. **TextChanged** event is used to write the code for the text box.
8	**Label**	Used to display static text on the toolstrip.

StatusStrip

Statusstrip is used to create a status bar for the form that can display the status of a command. Properties of statusstrip are same as toolstrip. I.e. **AllowItemReorder, ShowItemTooltips** and **Items**. For the status strip you can add the items like **Label, DropDownButton, SplitButton** and **ProgressBar**. Properties and other details of these items are same as items of toolstrip control. In most cases status bar contains only labels and progressbars. But not dropdown buttons and split buttons.

RichTextBox

Richtextbox is a text box with rich formatting features than in normal text box and has the following important properties.

Sno	**Property Name**	**Description**
1	**BulletIndent**	Used to specify the indentation for the bullets.
2	**DetectUrls**	Indicates whether urls are automatically detected and highlighted as hyperlink.
3	**Lines**	An Array of lines in rich text box.
4	**RightMargin**	Used to specify the rightmargin for selected text in rich text box.
5	**SelectedText**	Returns the selected text in rich text box.
6	**SelectionAlignment**	Used to specify the alignment for the selected text.
7	**SelectionBackColor**	Used to specify the background color for the selected text.
8	**SelectionBullet**	Indicates whether to make the selected text as bulleted text.
9	**SelectionCharOffset**	Used to make selected text as superscript or subscript. To make superscript, set a positive value and to make subscript, set a negative value.
10	**SelectionColor**	Used to set a color for the selected text.
11	**SelectionFont**	Used to specify font for the selected text.

12	**SelectionIndent**	Used to specify the left indentation for the selected text.
13	**SelectionRightIndent**	Used to specify the right indentation for selected text.
14	**SelectionLength**	Returns the number of characters selected.
15	**SelectionStart**	Returns the index of the character from which selection was started.
16	**Text**	Returns the total text.
17	**WordWrap**	Indicates whether wordwrap is on or off.
18	**ZoomFactor**	Used to specify the zoom for the rich text box.
19	**CanUndo**	Indicates whether undo works.
20	**UndoActionName**	Returns the undo action name.
21	**CanRedo**	Indicates whether redo works.
22	**RedoActionName**	Returns the redo action name.

Rich Text box has the following important methods.

Sno	Method Name	Description
1	**Clear()**	Clears the rich text box.
2	**Cut()**	Used move selected text to windows clipboard.
3	**Copy()**	Used to copy the selected text to windows clipboard.
4	**Paste()**	Used to paste the content of windows clipboard at cursor position.
5	**LoadFile()**	Used to open a file in to rich text box.
6	**SaveFile()**	Used to save the content of rich text box to a file.
7	**Undo()**	Used to cancel the last change made to rich text box.
8	**Redo()**	Used to repeat the last change made to rich text box.
9	**Find()**	Used to find given text within the rich text box.

ImageList

Imagelist is used as a central repository for storing images that can be displayed for other controls like button, label, listview and treeview and it has two important properties, **ImageSize** and **Images**. **ImageSize** property is used to specify the size of the images stored in imagelist control and **Images** is collection of images in imagelist control. To display an image from image list control to a control like button, use **ImageList** property to associate imagelist control to button and then use **ImageIndex** property to select an image.

TreeView

TreeView is used to display the data in a hierarchy as in case of left pane of windows explorer and has the following important properties.

Sno	Property Name	Description
1	**ImageList**	Used to associate an image list control to treeview.
2	**Indent**	Used to specify the indentation for child node from parent node.
3	**ImageIndex**	Used to specify the index of the image in imagelist control to be displayed as default image for every node.
4	**SelectedImageIndex**	Used to specify the index of the image in imagelist control to be displayed as default image for every selected node.
5	**Nodes**	Collection of nodes to be displayed in treeview.
6	**SelectedNode**	Returns the currently selected node in treeview.

Treeview has the following important methods.

Sno	Method Name	Description
1	**CollapseAll()**	Collapses all nodes in treeview.
2	**ExpandAll()**	Expands all nodes in treeview.
3	**GetNodaAt()**	Retrieves the node at specified point.
4	**GetNodeCount()**	Returns the total number of nodes in treeview.
5	**Sort()**	Used to sort the nodes in treeview.

Treeview has the following important events.

Sno	Event Name	Description
1	**AfterSelect**	This is the default event and will be raised after a node was selected.
2	**BeforeSelect**	Will be raised before a node was selected.

3	**AfterExpand**	Will be raised after a node was expanded.
4	**BeforeExpand**	Will be raised before a node was expanded.
5	**AfterCollapse**	Will be raised after a node was collapsed.
6	**BeforeCollapse**	Will be raised before a node was collapsed.
7	**AfterLabelEdit**	Will be raised after the label of a node was edited.
8	**BeforeLabelEdit**	Will be raised before the label of a node was edited.
9	**NodeMouseClick**	Will be raised when user clicks on the node with any button of the mouse.

ListView

Listview is used to display a list of items in four different views like **largeicon, smallicon, list** and **details** like right side pane in windows explorer. Listview has the following important properties.

Sno	**Property Name**	**Description**
1	**Alignment**	Used to specify the alignment for the items in listview i.e. either **Top** or **Left**.
2	**AutoArrange**	Indicates whether items are arranged automatically.
3	**Columns**	Collection of columns to be displayed details view of the listview.
4	**Items**	Collection of items to be displayed in listview.
5	**MultiSelect**	Indiates whether multiple items can be selected.
6	**SelectedItems**	Collection of all selected items.
7	**Sorting**	Indicates in which order the items in listview are being sorted.
8	**View**	Used to specify the view of the listview i.e. **largeicon, smallicon, list** or **Details**.
9	**LargeImagelist**	Used to associate an imagelist control that contains large sized images.
10	**SmallImagelist**	Used to associate an imagelist control that contains small sized images.

ListView has the following Important Methods.

Sno	**Method Name**	**Description**

1	**GetItemAt()**	Returns the item at specified point.
2	**Clear()**	Used clear all selected items.
3	**Sort()**	Used to sort the items in listview.

ListView has the following important events.

Sno	**Event Name**	**Description**
1	**SelectedIndexChanged**	Will be raised when a new item was selected in listview.
2	**AfterLabelEdit**	Will be raised after the label of a node was edited.
3	**BeforeLabelEdit**	Will be raised before the label of a node was edited.

MDI(Multiple Document Interface) Applications

MDI application is a windows application in which there is a main form, which called as MDI parent form and it contains the links to every other form in the application. When you are creating a windows application, then it is compulsory to create it as MDI application so that every form in the application can be accessed by the end user.

MDI Parent Form

To make a form as MDI Parent Form, set **IsMdiContainer** property of that form **true**. when you set this property true, then automatically the background color of the form will be changed indicating it is MDI parent form.

MDI Child Form

To display a form as child form within the MDI parent form, while displaying the form, set its **MdiParent** property to the name of the MDI parent form. When the form is displayed as MDI child form, then it is not possible to move it outside the MDI parent form and when MDI parent form is closed, then automatically all child forms in it will be closed.

Maintaining List of Opened Child Forms

To maintain a list of all opened child forms in MDI parent form in a menu, set **MdiWindowListItem** property of **MenuStrip** control to the name of the menu in which you want to maintain the opened child form list.

Arranging Child Forms Within The MDI Parent Form

To arrange the opened child forms within the MDI parent form, use the **LayoutMdi()** method of the MDI parent form. Within the parent form you can arrange the child forms either **Horizontal** or **Vertical** or **Cascade**.

Example

1. Create a windows application with the name **MDI** and make form1 as MDI parent form by setting its **IsMdiContainer** property **true**.
2. Add three more forms to the project with the names **Form2, Form3** and **Form4** and place a label on those forms with the caption Form2, Form3 and Form4 respectively.
3. Create a menu bar on Form1 with two menus **File** and **Windows** and set their names as **MnuFile** and **MnuWindows** respectively..
4. Within the **file menu** create four menu items **Form2, Form3, Form4** and **Exit** with the names **MnuForm2, MnuForm3, MnuForm4** and **MnuExit** respectively.

5. Within the **windows menu** create three menu items **Horizontal, Vertical** and **Cascade** with the names **MnuHorizontal, MnuVertical** and **MnuCascade**.
6. Within the properties of menustrip control, set **MdiWindowListItem** property to the name of windows menu, **MnuWindows**.
7. Write the following code within the click event of the menu items **Form2, Form3, Form4** and **Exit**.

```
Private sub MnuForm2_Click(…)
        Dim F2 as new Form2()
        F2.MdiParent=me
        F2.Show()
End Sub

Private sub MnuForm3_Click(…)
        Dim F3 as new Form3()
        F3.MdiParent=me
        F3.Show()
End Sub

Private sub MnuForm4_Click(…)
        Dim F4 as new Form4()
        F4.MdiParent=me
        F4.Show()
End Sub

Private Sub MnuExit_Click(…)
        Me.Close()
End Sub
```

8. Write the following code within the click events of menu items **MnuHorizontal, MnuVertical** and **MnuCascade** to arrange the child forms within MDI parent form.

```
Private Sub MnuHorizontal_Click(…)
        Me.LayoutMdi(MdiLayout.TileHorizontal)
End Sub
```

```
Private Sub MnuVertical_Click(…)
        Me.LayoutMdi(MdiLayout.TileVertical)
End Sub

Private Sub MnuCascade_Click(…)
        Me.LayoutMdi(MdiLayout.Cascade)
End Sub
```