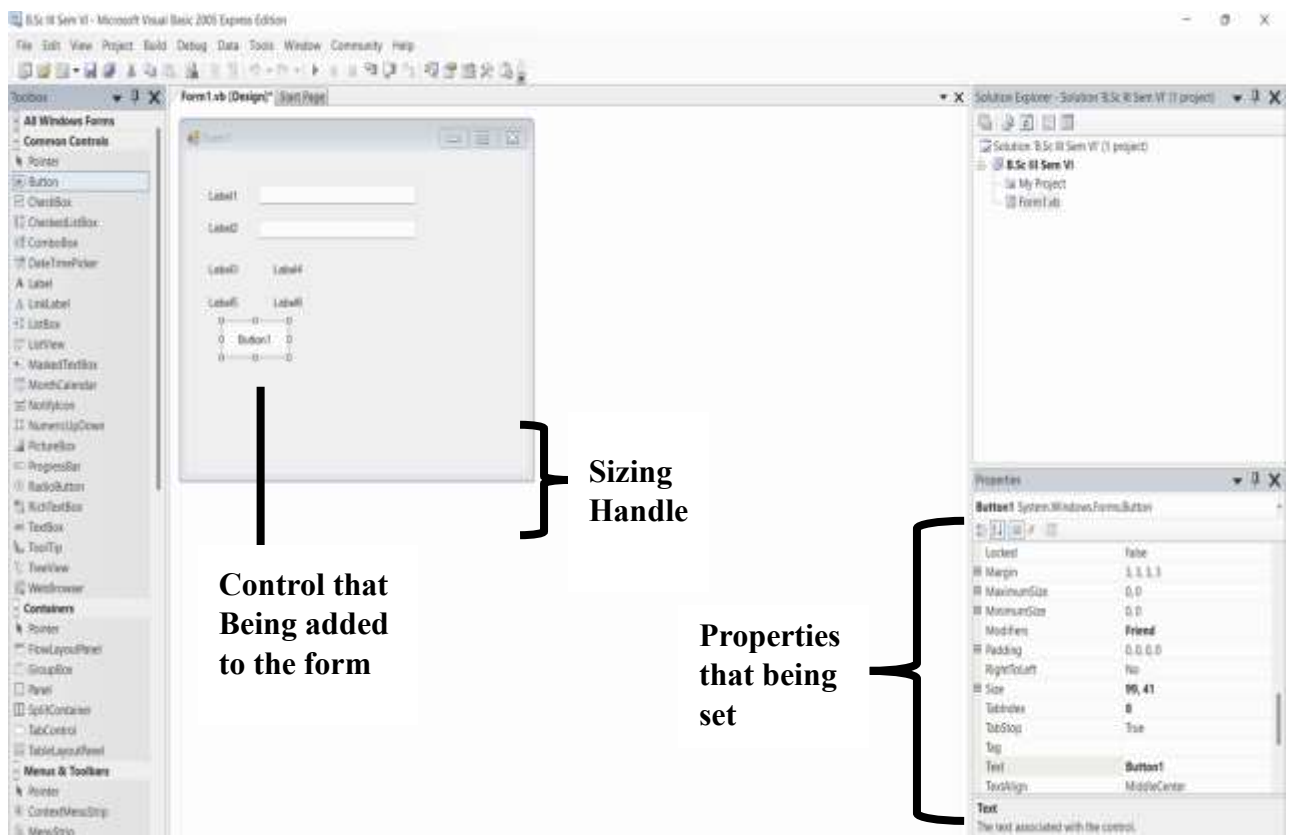


Unit IV : Windows Applications: Forms: Adding Controls to Forms, Handling Events, MsgBox, InputBox , Working with Multiple Forms, Setting the Startup Form, SDI & MDI Forms, Handling Mouse & Keyboard Events, **Common controls:** Text Boxes, Rich Text Boxes, Labels, Buttons, Checkboxes, Radio Buttons, Group Boxes, List Boxes, Checked List Boxes, Combo Boxes, Picture Boxes, Scroll Bars, Tool Tips, Timers, properties – methods.

Adding Controls to form

- To add control to a form, select the control in the Toolbox. Then, click in the form where you want to place the control and drag the pointer on the form to size the control.
- A second method for adding the controls is to simply double-click the control you want to add in the toolbox. This places the control in upper left corner of the form. You can then move and resize the control.
- A third way to add control to form is to drag the control from the toolbox to form. The control is placed wherever you drop it. You can then resize the control.
- After you have added controls to the form, you can work with several controls at once. For example, let's say that you have two labels and two textbox controls on your form and you want to make them all the same size with the same alignment. To do that, first select all four controls by holding down shift key as you click on them or by using mouse pointer to drag around all controls.
- To change the size of a form to accommodate the controls, click on the form and then drag it by one of its handles.
- To set the properties of a form or control, you work with properties window as shown in below fig 1.1. To display the properties for a specific control, click on it in the Form Designer window to select the control.

Control that selected in the toolbox



Control that Being added to the form

Sizing Handle

Properties that being set

[Fig 1.1: A form after Some controls have been added to it]

Event Handling in VB.Net

VB.Net is an event driven programming Language. An event is an action which occurs when clicking on a button, typing some texts or moving the mouse and that calls a function or causing for another event.

The events in a VB.Net program are of two types: user generated events and system generated events. The user-generated events occur when the user actions key press, clicks, mouse movements, etc. are happening. System generated events are notifications in the computer.

VB.Net mainly provide the following two events.

- Mouse Events
- Keyboard Events

VB.Net Mouse Events

Mouse events occur by the actions of the mouse in a Windows form, such as mouse move and mouse click. Mouse event present in the class System.Windows.Forms.MouseEventArgs.

The following are the mouse events in VB.Net.

- **MouseDown** – occurs when a mouse button is pressed.

```
Private Sub Button1_MouseDown(ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handles Button1.MouseDown
    TextBox1.ForeColor = Color.Brown
End Sub
```

- **MouseEnter** – occurs when the mouse pointer enters the control.

```
Private Sub Button1_MouseEnter(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button1.MouseEnter
    TextBox2.ForeColor = Color.Blue
End Sub
```

- **MouseHover** – occurs when the mouse pointer hovers over the control.

```
Private Sub Button1_MouseHover(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button1.MouseHover
    TextBox3.ForeColor = Color.CornflowerBlue
End Sub
```

- **MouseLeave** – occurs when the mouse pointer leaves the control.

```
Private Sub Button1_MouseLeave(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button1.MouseLeave
    TextBox4.ForeColor = Color.DarkGray
End Sub
```

- **MouseMove** – occurs when the mouse pointer moves over the control.

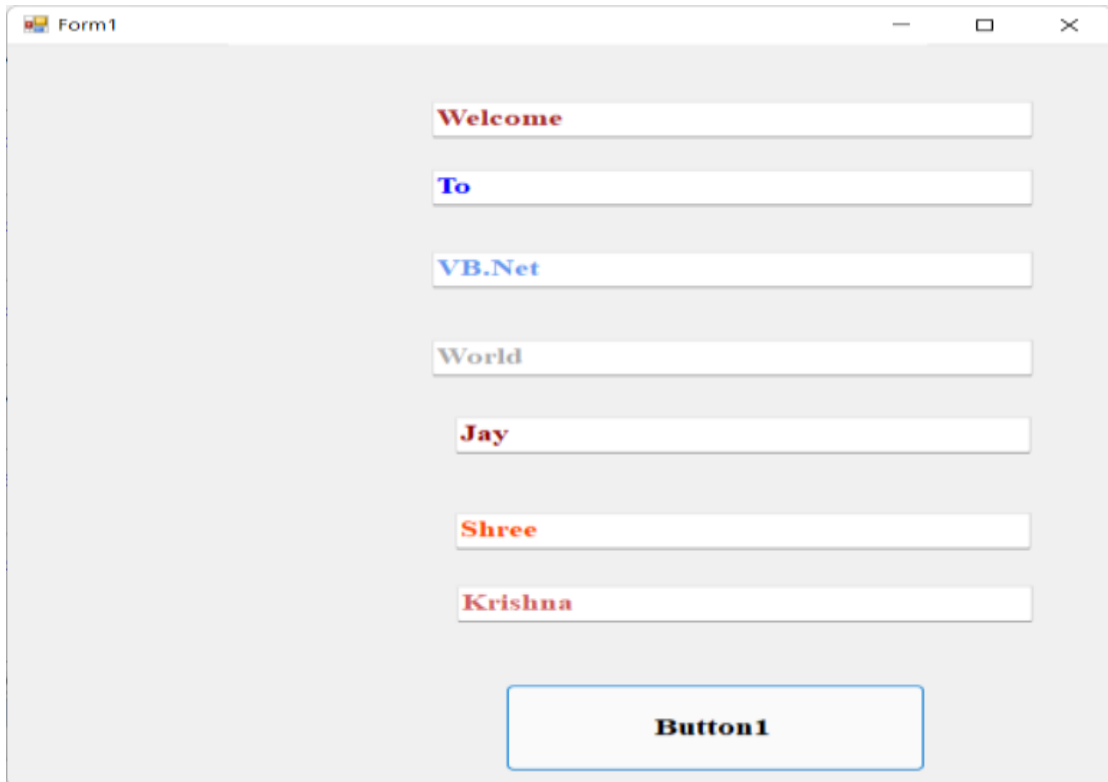
```
Private Sub Button1_MouseMove(ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handles Button1.MouseMove
    TextBox5.ForeColor = Color.DarkRed
End Sub
```

- **MouseUp** – occurs when the mouse pointer is over the control and the mouse button is released.

```
Private Sub Button1_MouseUp(ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handles Button1.MouseUp
    TextBox6.ForeColor = Color.IndianRed
End Sub
```

- **MouseWheel** – occurs when the mouse wheel moves and the control has the focus.

```
Private Sub Button1_MouseWheel(ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handles Button1.MouseWheel
    TextBox7.ForeColor = Color.OrangeRed
End Sub
```



[Fig : Mouse Events Output]

VB.Net Keyboard Events

Keyboard events allow you to validate keystrokes. Keyboard event present in the class System.Windows.Forms.KeyEventArgs. They are raised by the control that has the focus and is receiving input.

Following are the key events:

- KeyDown
- KeyUp
- KeyPress

KeyDown:

It occurs when a key is pressed down on the keyboard, it repeats while the user holds the key depressed and the control has focus.

Example:

```
Private Sub TextBox1_KeyDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyEventArgs) Handles TextBox1.KeyDown

    TextBox1.ForeColor=Color.LightPink
End Sub
```

KeyUp:

It occurs when a key is released on the keyboard, and the control has focus.

Example:

```
Private Sub TextBox1_KeyUp(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyEventArgs) Handles TextBox1.KeyUp

    TextBox1.ForeColor=Color.Aqua
End Sub
```

The KeyDown and KeyUp events get an argument using KeyEventArgs. It has the following properties:

1. Handled : It indicates if the KeyPress event is handled.
2. KeyChar : It stores the character corresponding to the pressed.

KeyPress:

It is raised for character keys while the key is pressed and then released by the user. The KeyPress event is not raised by noncharacter keys. It occurs when a key is released while the control has focus.

Example:

```
Private Sub TextBox1_KeyPress(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyEventArgs) Handles TextBox1.KeyPress

    TextBox1.ForeColor=Color.Aqua
End Sub
```

MessageBox() Function

MessageBox() Function display dialog box which is use to show a pop up message box and prompt the user to **click on command button** before he/she continues. It interrupts the user that means it immediately block further interaction.



Syntax:

```
Msg = MessageBox.Show(prompt, style value, Title)
```

prompt - Required. String expression containing the message to be displayed. The maximum length of a single line prompt is approximately 1024 characters, depending on the width of the characters used.





Style value – Denote the style and type of button to display in messagebox. As in shown below Fig.

- [-] MsgBoxStyle.AbortRetryIgnore
- [-] MsgBoxStyle.ApplicationModal
- [-] MsgBoxStyle.Critical
- [-] MsgBoxStyle.DefaultButton1
- [-] MsgBoxStyle.DefaultButton2
- [-] MsgBoxStyle.DefaultButton3
- [-] MsgBoxStyle.Exclamation
- [-] MsgBoxStyle.Information
- [-] MsgBoxStyle.MsgBoxHelp
- [-] MsgBoxStyle.MsgBoxRight
- [-] MsgBoxStyle.MsgBoxRtlReading
- [-] MsgBoxStyle.MsgBoxSetForeground
- [-] MsgBoxStyle.OkCancel
- [-] MsgBoxStyle.OkOnly
- [-] MsgBoxStyle.Question

Constant	Value	Description
vbOKOnly - or - MsgBoxStyle.OKOnly	0	Display OK button only.
vbOKCancel - or -	1	Display OK and Cancel buttons.

MsgBoxStyle.OKCancel		
vbAbortRetryIgnore - or - MsgBoxStyle.AbortRetryIgnore	2	Display Abort , Retry , and Ignore buttons.
vbYesNoCancel - or - MsgBoxStyle.YesNoCancel	3	Display Yes , No , and Cancel buttons.
vbYesNo - or - MsgBoxStyle.YesNo	4	Display Yes and No buttons.
vbRetryCancel - or - MsgBoxStyle.RetryCancel	5	Display Retry and Cancel buttons.

Determines which icon to display:

Constant	Value	Description	Icon
vbCritical - or - MsgBoxStyle.Critical	16	Display Critical Message icon.	
vbQuestion - or - MsgBoxStyle.Question	32	Display Warning Query (question mark) icon.	
vbExclamation - or - MsgBoxStyle.Exclamation	48	Display Warning Message icon.	
vbInformation - or - MsgBoxStyle.Information	64	Display Information Message icon.	

If you wanted to display OKCancel button along with the information icon, you could have coded the second argument as

```
MsgBoxStyle.Information + MsgBoxStyle.OkCancel

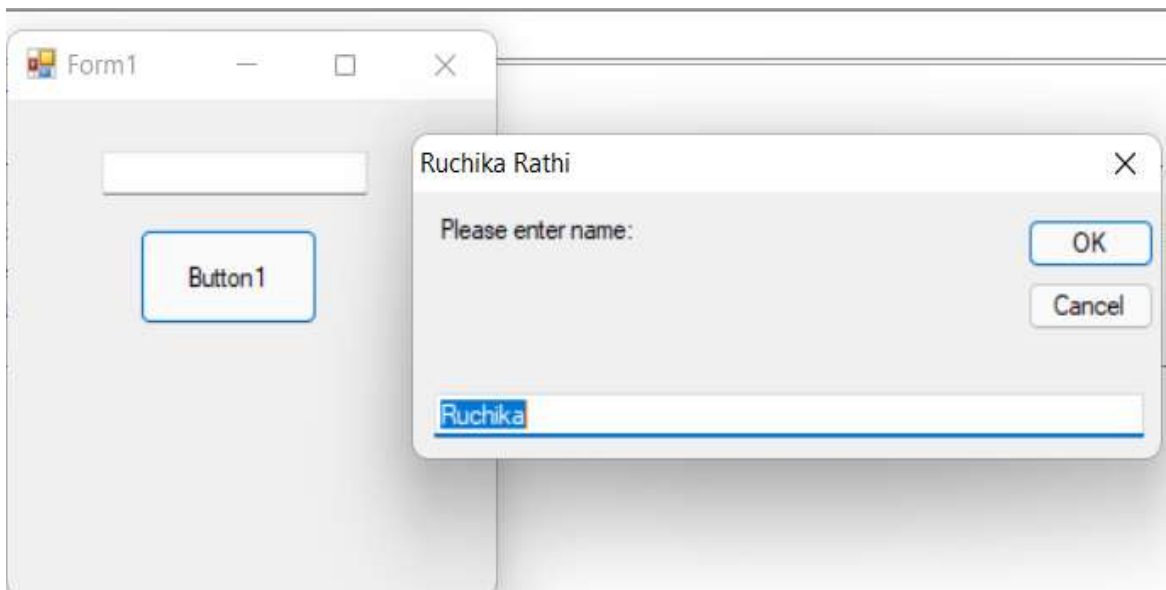
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim msg As String
    msg = MsgBox("Ruchika", MsgBoxStyle.Information +
MsgBoxStyle.OkCancel, "Rathi")
End Sub
End Class
```



Title: The title argument will display the title of the message board.

InputBox Function

This **InputBox** function will display a built in dialog box that can be used to prompt the user for information. This type of message box displays a message and waits for the user to respond by pressing a button. **This function always returns a String** so you often have to convert the value to its correct data type. Even if a numerical value is entered it will be returned as a string.



[Fig: InputBox Function demonstration]

Syntax

`InputBox(Prompt, Title, default_text, x-position, y-position)`

prompt: Required. String expression containing the message to be displayed. The maximum length of a single line prompt is approximately 1024 characters, depending on the width of the characters used.

title: Optional. String expression displayed in the title bar of the dialog box. If you omit title, the application name is placed in the title bar.

default: Optional. String expression displayed in the text box as the default response if no other input is provided. If you omit default, the text box is displayed empty.

xpos: Optional. Numeric expression that specifies, the horizontal distance of the left edge of the dialog box from the left edge of the screen. If xpos is omitted, the dialog box is horizontally centered.

ypos: Optional. Numeric expression that specifies, the vertical distance of the upper edge of the dialog box from the top of the screen. If ypos is omitted, the dialog box is vertically positioned approximately one-third of the way down the screen.

Example:

```
Public Class Form1

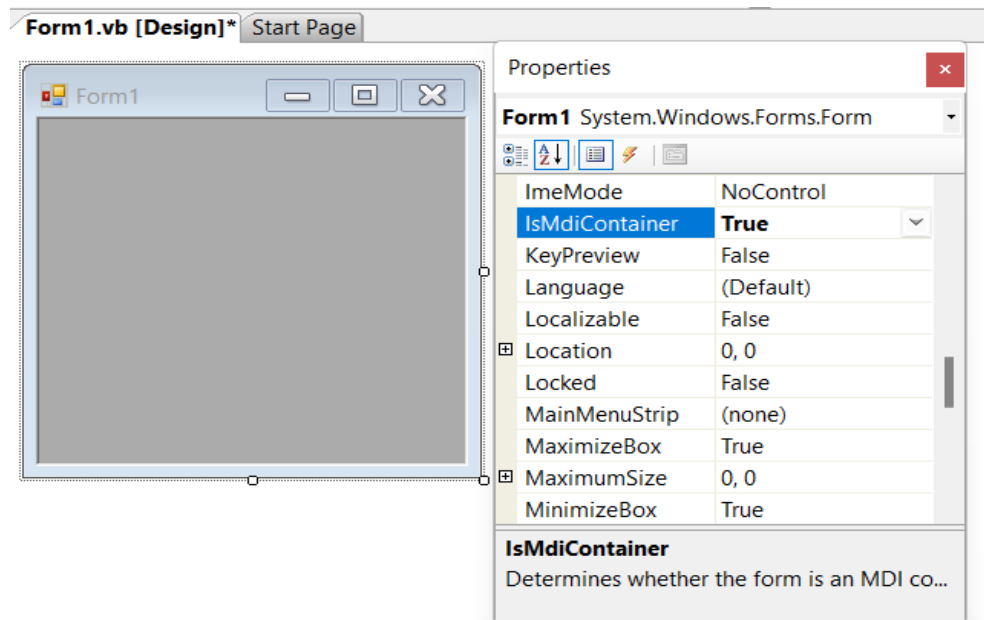
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button1.Click
        Dim msg As String
        msg = InputBox("Please enter name:", "Ruchika Rathi",
"Ruchika", 300, 150)
        TextBox1.Text = msg
    End Sub
End Class
```

The output of the given code will be as shown in above fig[InputBox Function Demonstration].

How to develop a multiple-document interface(MDI)?

The process of creating MDI Application are as follows:

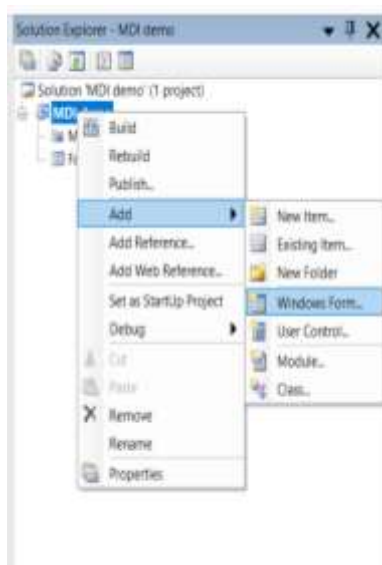
1. To make a form as a parent we have to set property IsMdiContainer as True inside property window as shown in below fig 1.



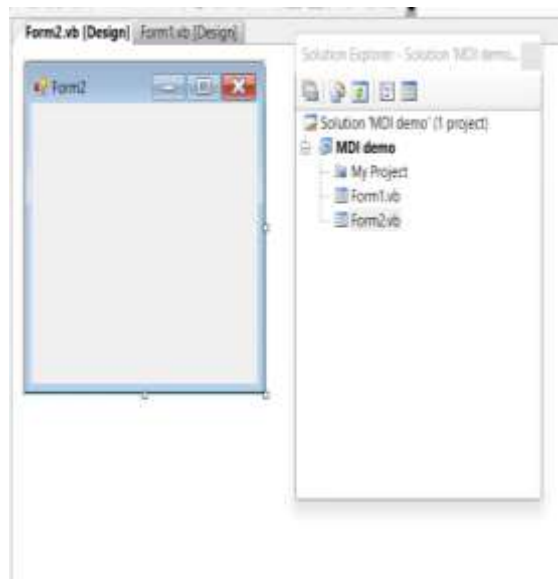
[Fig 1]

2. After that to add more form inside an MDI Forms: Go to Solution Explorer and then right click on **Name of Project(MDI demo in my**

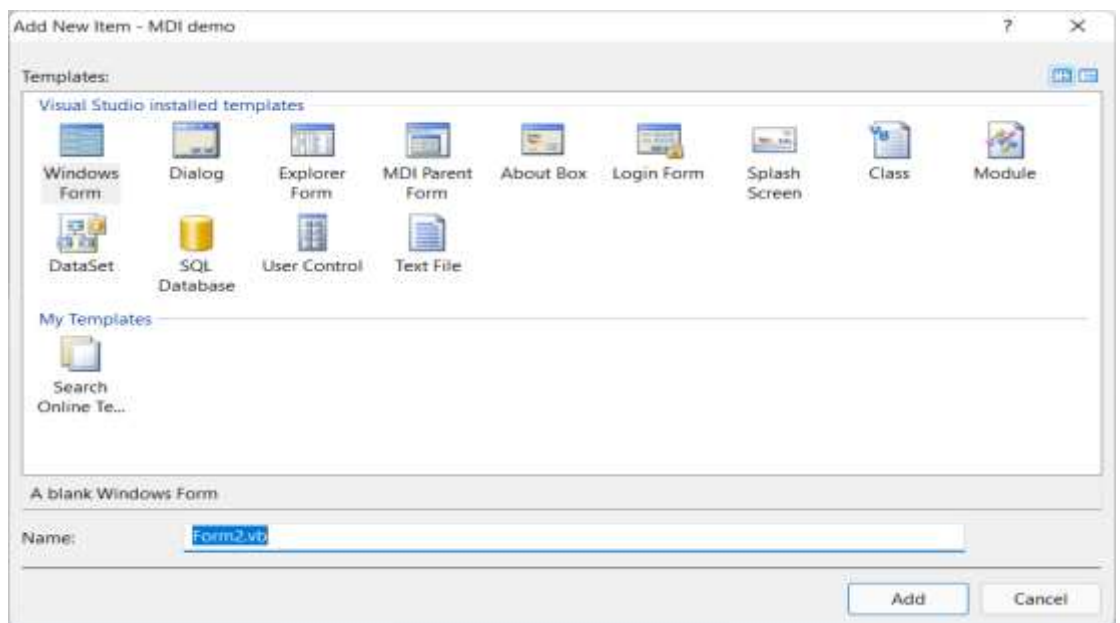
case) as shown in fig 2 below from pop up shown we have to select **Add** inside that we have to select windows form. Then after clicking on Windows form, it will show pop up window form providing name of that form by default the text will be Form2. Then after providing any name to form or to set as it is default name as shown in below fig 3. then click on add then it will add Form2 inside solution explore as shown in fig 4. Now we assuming Form2 as child.



[Fig 2]



[Fig 4]



[Fig 3]

3. Now, to create Menu, add a MenuStrip control to form1(Parent). The control will appear in the Component Designer tray at bottom of the Designer window, and the Menu property of the form will set name of that control as shown in fig 5 below. To add menu items, click on wherever it says “Type Here” in menu designer and select MenuItems from combobox options as shown in fig 6 below. It will add MenuItem1 as shown in fig 7 below. When the user double click on ToolStripMenuItem1 it will open an code window for you for click event occurs after clicking on the particular MenuItems.

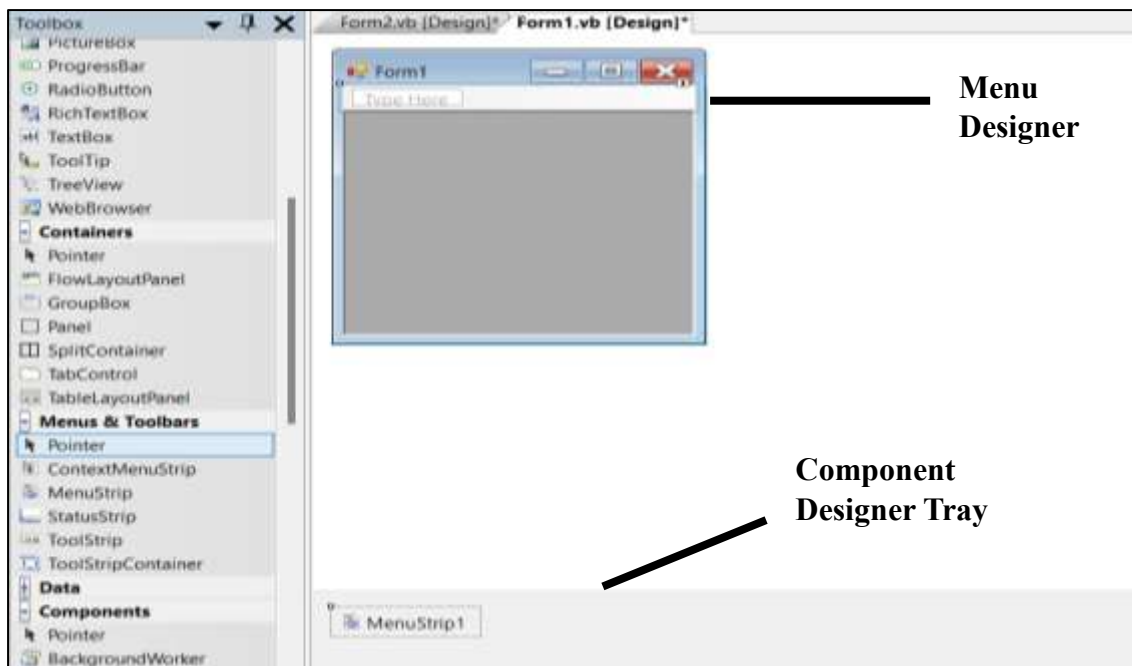


Fig 5

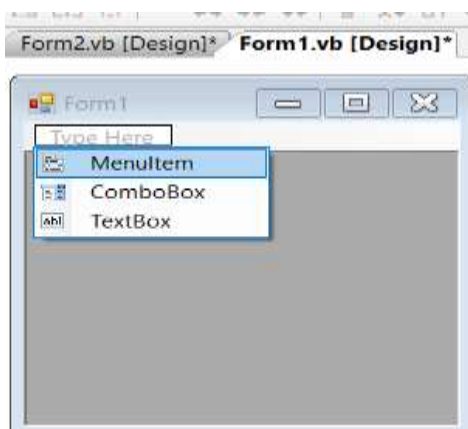


Fig 6

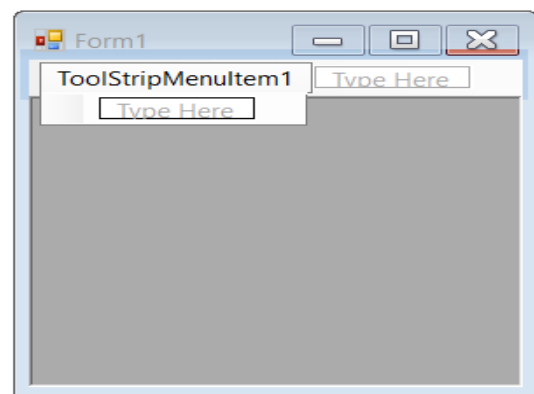


Fig 7

4. When the user double clicks on ToolStripMenuItem1 it will open a code window for you for click event occurs after clicking on the particular MenuItems as shown in fig 8.

```
Public Class Form1
    Private Sub ToolStripMenuItem1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ToolStripMenuItem1.Click
        End Sub
    End Class
```

Fig 8

5. Now, to display child form(Form2) inside the parent form(Form1). You have to write some lines of code as shown below inside code window shown in fig 8.

```
Dim frm2 As New Form2
Me.IsMdiContainer=True
frm2.MdiParent=Me
frm2.show()
```

In above code firstly we create an instance(object) of a child form, you must set its MdiParent property to the name of its parent form. Since a child form is usually displayed by its parent form, you can use the Me keyword to identify the parent form. The output of MDI Form2 inside Form1 will be look as shown in Fig 9



Fig 9

SDI(Single Document Interface) Form

An SDI opens each document in its own primary window. Each window has its own menu, toolbar, and entry in the task bar. Therefore, an SDI is not permitted to a parent window. This makes it easier for the user to view the contents of the various windows. **Notepad** is an example of an SDI application.

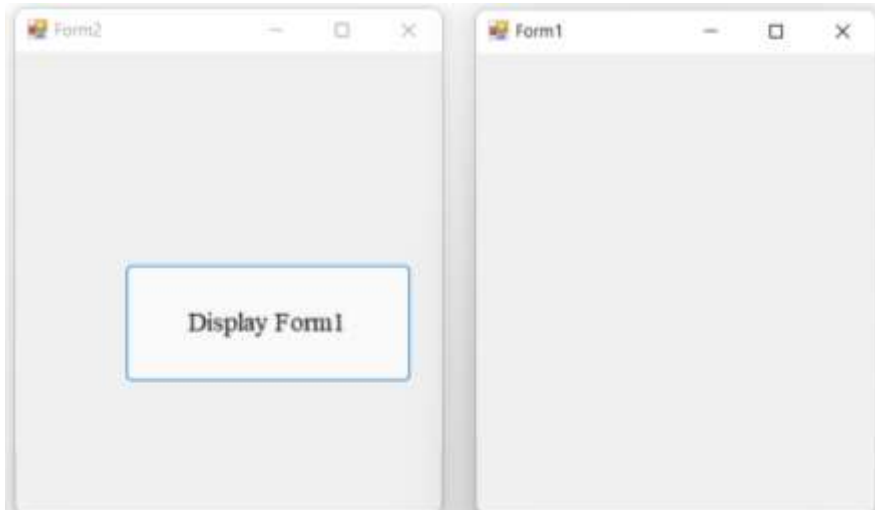


Fig: SDI Form

Multiple Document Interface (MDI):

An MDI lets you open more than one document at the same time. The MDI has a parent window, and any number of child windows. The child windows usually share various parts of the parent window’s interface, including the menu bar, toolbar and status bar. Therefore, an MDI is permitted to the parent window. **Microsoft Visual Studio** is an MDI application.

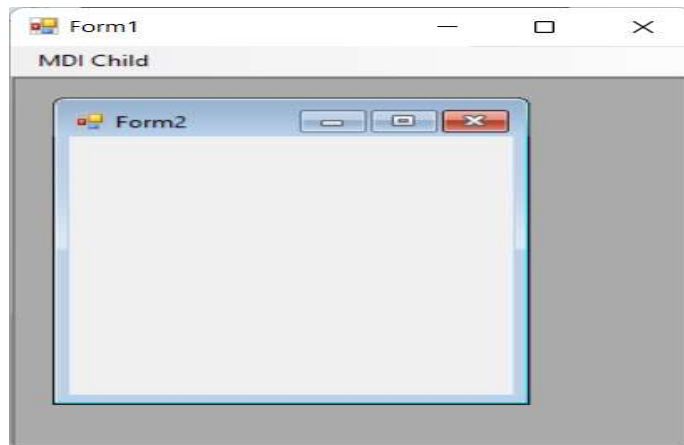


Fig : MDI Form

Difference Between SDI and MDI

- a) MDI stands for “Multiple Document Interface” while SDI stands for “Single Document Interface”.
- b) One document per window is enforced in SDI while child windows per document are allowed in MDI.

- c) MDI is a container control while SDI is not container control.
- d) SDI contains one window only at a time but MDI contains multiple documents at a time appeared as child window.
- e) MDI supports many interfaces means we can handle many applications at a time according to user's requirement. But SDI supports one interface means you can handle only one application at a time.
- f) For switching between documents MDI uses special interface inside the parent window while SDI uses Task Manager for that.
- g) In MDI grouping is implemented naturally but in SDI grouping is possible through special window managers.
- h) For maximizing all documents, parent window is maximized by MDI but in case of SDI, it is implemented through special code or window manager.
- i) Switch focus to the specific document can be easily handled while in MDI but it is difficult to implement in SDI.

Common Controls, Properties and Methods:

ToolBox Control in VB.Net, will act as an object use for interaction with user in the Visual Studio form to create an Windows Application. Every Visual basic control consists of three important elements:

1. Properties which describes the objects,
2. Methods helps objects to do task after an action perform.
3. Events informs an application what happen with object if some action perform.

Control Properties:

All Visual basic objects such as button, textbox, label etc can be moved, resized, or customized by setting their properties. A properties is a value or characteristic held by Visual basic object, such as Caption, ForeColor, BackColor etc.

Properties can be **set at design time** by using the Properties window or at **run time** by using statements in program code.

Object.Property = Value

For ex:

Textbox1.Text = "Hello"

Control Methods:

Method is specific task created as a member of class and helps objects do something. Methods are used to access or manipulate the characteristics of an object or variable.

Windows forms is in the **System.Windows.Forms** namespace, and the form class is **System.Windows.Forms.Form**. The Form class itself is based on the Control class, which means that forms share a lot of the properties and methods that controls do.

TextBox

Textbox are exactly what their name implies: box-like controls in which you can enter text. Windows forms text boxes are used to get input from the user or to display text. The TextBox control is generally used for editable text, although it can also be made read only. Text boxes can display multiple lines, wrap text to the size of the control, and add basic formatting, such as quotation marks and masking characters for passwords.

The text displayed by the control is contained in the Text property. By default, you can enter up to 2,048 characters in a text box. If you set the **MultiLine** property to True to make the control, accept multiple lines of text, you can enter up to 32KB of text. The Text property can be set at design time with the Properties window, at run time in code, or by user input at run time.

You can set or read text from text boxes at run time, and the user can enter and edit text in text boxes as well. You can limit the amount of text entered into a TextBox control by setting the **MaxLength** property to a specific number of characters. TextBox controls also can be used to accept passwords if you use the **PasswordChar** property to mask characters.

Properties of TextBox

Property	Means
MaxLength	Sets/gets the maximum number of characters the user can type into the text box.
Multiline	Sets/gets a value specifying if this is a multiline text box control.
PasswordChar	Sets/gets the character used to mask characters of a password in a single-line text box.
ScrollBars	Sets/gets what scroll bars should appear in a multiline text box.
Text	Sets/gets the current text in the text box.
TextAlign	Sets/gets how text is aligned in a text box control.

WordWrap	Indicates if a multiline text box control automatically wraps words.
----------	--

Methods of TextBox

Methods	Means
AppendText	Appends text to the current text in the text box.
Clear	Clears all text from the text box.
Copy	Copies the selected text in the text box to the Clipboard.
Cut	Moves the selected text in the text box to the Clipboard.
Paste	Replaces the selected text in the text box with the contents of the Clipboard.

Q How to access Text in TextBox? Explain the steps of adding scrollbars and aligning text in textboxes.

Access Text in TextBox:

you set the text in a text box using the Text property, like this:

```
Private Sub Button1_Click_1(ByVal sender As System.Object, _ ByVal e As System.EventArgs) Handles Button1.Click  
  
    TextBox1.Text = "Hello from Visual Basic"  
  
End Sub
```

When the user clicks the command button Button1, the text "Hello from Visual Basic" appears in the text box. And you can retrieve/get text from a text box in the same way:

```
Private Sub Button1_Click_1(ByVal sender As System.Object, _ ByVal e As System.EventArgs) Handles Button1.Click  
  
    Dim strText As String
```

```
strText = TextBox1.Text
```

```
End Sub
```

Steps of adding scrollbars :

If you're using multiline text boxes, it would be even better if you could add scroll bars to let the user enter even more text. If your program's users are going to be entering a lot of text into text boxes, you can avoid the need for huge text boxes by adding scroll bars.

Using the ScrollBars property, there are four ways to add scroll bars to a text box; here are the settings you use for ScrollBars, and the type of scroll bars each setting displays:

0: None

1: Horizontal

2: Vertical

3: Both

Note that in order for the **scroll bars** to actually appear, the **text box's MultiLine** property must be **True**. After you install scroll bars in a text box, the result appears as in Figure below. Now the user can enter much more text simply by scrolling appropriately.

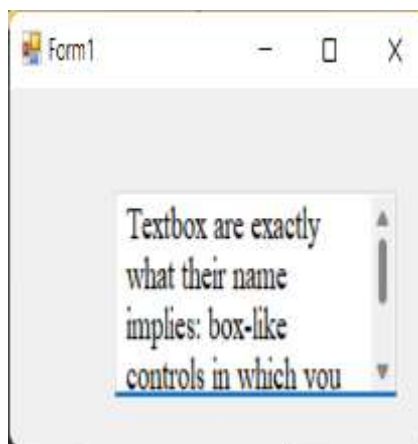


Fig: scroll bars in a text box

Aligning Text in textboxes:

Text boxes have an TextAlign property, for alignment of text (there are three possibilities: **0: left-justified**, **1: right-justified**, and **2: centered**) at design time in all the text boxes. You needed to set a text box's MultiLine property to True before text alignment. Fig below shows a textbox with right-justified.

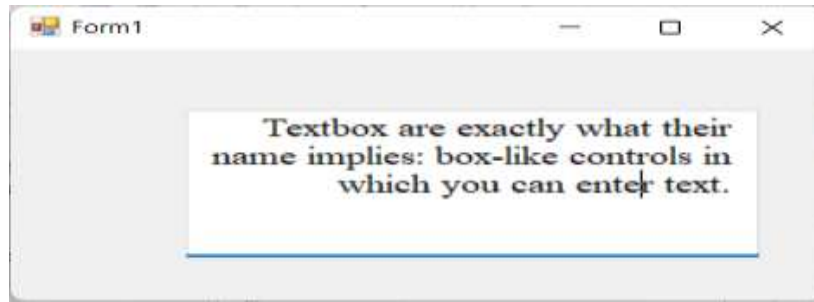


Fig : textbox with right-justified text

Q. Differentiate between Textbox and RichTextbox Control.

Q. Explain Rich TextBox Control with any three properties and any three methods.

The RichTextBox is similar to the TextBox, but it has additional formatting capabilities. Whereas the TextBox control allows the Font property to be set for the entire control, the RichTextBox allows you to set the Font, as well as other formatting properties, for selections within the text displayed in the control.

When considering the difference between the TextBox and RichTextBox controls, compare Notepad and WordPad, the two text editors that are included with Windows. Notepad, based on the TextBox control, has no formatting applicable to selections within the text. WordPad, on the other hand, is practically a full-fledged word processor, with such features as bullets, hanging indents, fonts, and character styles applicable to selections within the text.

The default file format for the RichTextBox is the Rich Text Format (RTF), a file format that can carry formatting information.

The RichTextBox control inherits all the properties, methods, and events from the TextBoxBase class, so it behaves much like the TextBox control. In addition, it has many properties, methods, and events specific to itself.

Properties of Rich TextBox

Properties	Means
AutoSize	Sets/gets a value specifying if the size of the rich text box automatically adjusts when the font changes.
AutoWordSelection	Sets/gets a value specifying if automatic word selection is enabled.
BulletIndent	Sets/gets the indentation used in the rich text box when the bullet style is applied to the text.
SelectedText	Sets/gets the selected text within the rich text box.
SelectionAlignment	Sets/gets the alignment to apply to the current selection or insertion point.
SelectionBullet	Sets/gets a value specifying if the bullet style is applied to the current selection or insertion point.

SelectionColor	Sets/gets the text color of the current text selection or insertion point.
SelectionFont	Sets/gets the font of the current text selection or insertion point.
SelectionHangingIndent	Sets/gets the distance between the left edge of the first line of text in the selected paragraph and the left edge of the next lines in the same paragraph.
SelectionIndent	Sets/gets the distance in pixels between the left edge of the rich text box and the left edge of the current text selection or text added after the insertion point.
SelectionStart	Sets/gets the starting point of text selected in the text box.

Method of Rich Textbox

Methods	Means
AppendText	Appends text to the current text of the rich text box. Example <i>RichTextBox1.AppendText("Type additional text here")</i>
Clear	Clears all text from the RichTextBox control. Example <i>RichTextBox1.Clear()</i>
Find	Searches for a specified text within the RichTextBox. Example <i>Dim index As Integer = RichTextBox1.Find("searchText")</i>
LoadFile	Loads the contents of the RichTextBox to a file. Example <i>RichTextBox1.LoadFile("C:\Path\to\File.rtf")</i>
SaveFile	saves the contents of the RichTextBox to a file. Example <i>RichTextBox1.SaveFile("C:\Path\to\NewFile.rtf")</i>

Labels

You use labels for just what they sound like—to label other parts of your application. Labels usually are used to display text that cannot be edited by the user. Your code can change the text displayed by a label.

The caption for a label is stored in the Text property. Because you can change that caption in code, labels can act a little like non-editable text boxes, displaying text and messages to the user. The TextAlign (formerly Alignment) property allows you to set the alignment of the text within the label.

Properties	Means
AutoSize	Sets/gets a value specifying if the control should be automatically resized to display all its contents.
BorderStyle	Sets/gets the border style for the control.
FlatStyle	Sets/gets the flat style appearance of the label control.
Image	Sets/gets the image that is displayed on a Label.
ImageAlign	Sets/gets the alignment of an image that is displayed in the control.
TextAlign	Sets/gets the alignment of text in the control.

Link Labels

Link labels are new in VB .NET. They're based on the Label class, but also let you support Web-style hyperlinks to the Internet and other Windows forms. In other words, you can use a link label control for everything that you can use a label control for, and you can also make part of the text in this control a link to a Visual Basic object or Web page.

Properties	Means
ActiveLinkColor	Sets/gets the color for an active link.
DisabledLinkColor	Sets/gets the color for a disabled link.
LinkArea	Sets/gets the range in the text to treat as a link.

LinkBehavior	Sets/gets a value that represents the behavior of a link.
LinkColor	Sets/gets the color for a normal link.
Links	Gets the collection of links in the LinkLabel control.
LinkVisited	Sets/gets a value specifying if a link should be displayed as though it had been visited
VisitedLinkColor	Sets/gets the color used for links that that have been visited.

Button Control

Buttons provide the most popular way of creating and handling an event in our code. Buttons can be clicked with the mouse or with the Enter key if the button has the focus.

Button control is used to perform an action. Whenever user clicks on a button click event associated with the button is fired and action associated with the event is executed.

You can set the AcceptButton or CancelButton property of a form to let users click a button by pressing the Enter or Esc keys-even if the button does not have focus. And when you display a form using the ShowDialog method, you can use the DialogResult property of a button to specify the return value of ShowDialog.

You also can change the button's appearance, giving it an image or aligning text and images in it as you like. You can even make it look flat for a "Web" look, setting the FlatStyle property to FlatStyle.Flat. Or, you can set the FlatStyle property to FlatStyle.Popup, which means it looks flat until the mouse pointer passes over it, when the button pops up to give it the standard Windows button appearance.

Properties

Properties	Means
DialogResult	Gets/sets the value returned to the parent form when the button is clicked. Often used when you're creating dialog boxes.
BorderStyle	Sets/gets the border style for the control.

FlatStyle	Sets/gets the flat style appearance of the label control.
Image	Sets/gets the image that is displayed on a Label.
ImageAlign	Sets/gets the alignment of an image that is displayed in the control.
TextAlign	Sets/gets the alignment of text in the control.

Method

Method	Means
PerformClick	Causes a Click event for a button

Checkboxes:

The checkbox is a control that lets the user select or deselect alternative from the list of choices. A checkmark or tick will show up on the windows form when a checkbox is chosen. You use a checkbox to give the user an option, such as true/false or yes/no. The checkbox control can display an image or text or both.

By default, checkboxes are two-state controls; you use the Checked property to get or set the value of a two-state checkbox. However, if you set the checkbox's ThreeState property to True, you make the checkbox into a three-state control. You use the CheckState property to get or set the value of the three-state checkbox.

The three states are:

Checked— A check appears in the checkbox.

Unchecked— No check appears in the checkbox.

Indeterminate— A check appears in the checkbox on a gray background.

Properties:

Properties	Means
Appearance	Gets/sets the appearance of a checkbox.
AutoCheck	Specifies if the Checked or CheckState values and the checkbox's appearance are automatically changed when the checkbox is clicked.

CheckAlign	Gets/sets the horizontal and vertical alignment of a checkbox in a checkbox control.
Checked	Gets/sets a value indicating if the checkbox is in the checked state.
ThreeState	Specifies if the checkbox will allow three check states rather than two.

Radio Button:

Radio buttons, also called option buttons, are similar to checkboxes—the user can select and deselect them—except for two things: they are round where checkboxes are square, and you usually use radio buttons together in groups.

In fact, that's the functional difference between checkboxes and radio buttons—checkboxes can work independently, but radio buttons are intended to work in groups. When you select one radio button in a group, the others are automatically deselected.

Properties:

Properties	Means
Appearance	Gets/sets the appearance of a radio button.
AutoCheck	Gets/sets a value indicating whether the Checked value and the appearance of the control automatically change when the radio button is clicked.
TextAlign	Gets/sets the alignment of the text in a radio button.
Checked	Gets/sets a value indicating whether the radio button is checked.

GroupBox

Group boxes are used to provide a grouping for other controls. Group boxes display frames around their contained controls and can display text in a caption. Generally, GroupBox control is use as a container for radio button. When RadioButton are grouped using GroupBox, user can select one RadioButton from each GroupBox.

Creating Group Boxes

You can create group boxes at design time or run time. After you've created the group boxes, you can drag other controls into them. Note that although you can **set the caption** for group boxes with the **Text** property, **group boxes do not** have either a **BorderStyle property**, nor do they support **scroll bars**.

ListBoxes

List boxes display a list of items from which the user can select one or more. If there are too many items to display at once, a scroll bar automatically appears to let the user scroll through the list. In Visual Basic .NET, each item in a list box is itself an object.

You also can scroll list boxes horizontally when you set the **MultiColumn** property to **True**. Alternatively, when the **ScrollAlwaysVisible** property is set to **True**, a scroll bar always appears.

The **SelectedIndex** property returns an integer value that corresponds to the selected item. If the first item in the list is selected, then the **SelectedIndex** value is **0**. You can change the selected item by changing the **SelectedIndex** value in code; the corresponding item in the list will appear highlighted on the Windows form. If no item is selected, the **SelectedIndex** value is **-1**. You also can set which items are selected with the **SetSelected** method in code. The **SelectedItem** property is similar to **SelectedIndex**, but returns the object corresponding to the item itself (which is usually a string value).

The items in list boxes are stored in the Items collection; the **Items.Count** property holds the number of items in the list. (The value of the **Items.Count** property is always one more than the largest possible **SelectedIndex** value because **SelectedIndex** is zero-based.) To add or delete items in a **Listbox** control, you can use the **Items.Add**, **Items.Insert**, **Items.Clear**, or **Items.Remove** methods. You also can add a number of objects to a list box at once with the **AddRange** method. Or you can add and remove items to the list by using the **Items** property at design time.

You also can use the **BeginUpdate** and **EndUpdate** methods. These enable you to add a large number of items to the **Listbox** without the list box being redrawn each time an item is added to the list. The **FindString** and **FindStringExact** methods enable you to search for an item in the list that contains a specific search string.

You also can support multiple selections in list boxes. The **SelectionMode** property determines how many list items can be selected at a time; you can set this property to **None**, **One**, **MultiSelect**, or **MultiExtended**:

- **MultiExtended**— Multiple items can be selected, and the user can use the Shift, Ctrl, and arrow keys to make selections.
- **MultiSimple**— Multiple items can be selected.
- **None**— No items may be selected.
- **One**— Only one item can be selected.

When you support multiple selections, you use the Items property to access the items in the list box, the SelectedItems property to access the selected items, and the SelectedIndices property to access the selected indices.

Properties:

Properties	Means
ColumnWidth	Gets/sets column width; use with multicolumn list boxes.
HorizontalScrollbar	Gets/sets if a horizontal scroll bar is displayed in the list box.
Items	Returns a collection of the items of the list box.
MultiColumn	Gets/sets if the list box supports multiple columns.
ScrollAlwaysVisible	Gets/sets if a vertical scroll bar is always shown.
SelectedIndex	Gets/sets the index of the list box's currently selected item.
SelectedItem	Gets/sets the selected item in the list box.
SelectionMode	Gets/sets the mode with which items are selected.
Sorted	Gets/sets if the items in the list box are sorted. The sort is alphabetical.
Text	Gets the text of the selected item in the list box.

Methods

Methods	Means
BeginUpdate	Turns off visual updating of the list box until the EndUpdate method is called.

ClearSelected	Unselects all the items in a list box.
EndUpdate	Resumes visual updating of the list box.
FindString	Finds the first item in the list box that begins with the indicated string.
FindStringExact	Finds the first item in the list box that matches the indicated string exactly.

CheckListBoxes

CheckListBox is the combination of CheckBox and ListBox that means that CheckListBox inherit from CheckBox as well ListBox. So, all the properties and methods of ListBox and CheckBox are applicable to CheckListBox.

ComboBoxes

ComboBox represents list of items in a drop down list type structure from which user can **select only one item** at a time. ComboBox control comes with built in TextBox so user can enter new item if it is not available in the list of ComboBox. It **occupies less space** on the form because of its **drop down structure**.

By default, a combo box displays a text box with a hidden drop-down list. The DropDownStyle property determines the style of combo box to display.

DropDownStyle property include:

- i. **DropDown** (the default)-Includes a drop-down list and a text box. The user can select from the list or type in the text box.
- ii. **Simple**-Includes a text box and a list, which doesn't drop down. The user can select from the list or type in the text box. The size of a simple combo box includes both the edit and list portions. By default, a simple combo box is sized so that none of the list is displayed. Increase the Height property to display more of the list.
- iii. **DropDownList**-This style allows selection only from the drop-down list. This is a good one to keep in mind when you want to restrict the user's input, but if you want to use this one, you also should consider simple list boxes.

Properties of ComboBox

Properties	Means

DropDownStyle	Gets/sets the style of the combo box.
MaxDropDownItems	Gets/sets the maximum number of items visible in the drop down part of a combo box.
Items	Returns a collection of the items of the list box.
MaxLength	Gets/sets the maximum number of characters in the combo box's text box.
SelectedIndex	Gets/sets the index of the list box's currently selected item.
SelectedItem	Gets/sets the selected item in the list box.
Sorted	Gets/sets if the items in the list box are sorted. The sort is alphabetical.
Text	Gets the text of the selected item in the list box.

Methods of ComboBox

Methods	Means
BeginUpdate	Turns off visual updating of the list box until the EndUpdate method is called.
ClearSelected	Unselects all the items in a list box.
EndUpdate	Resumes visual updating of the list box.
FindString	Finds the first item in the list box that begins with the indicated string.
FindStringExact	Finds the first item in the list box that matches the indicated string exactly.

PictureBox

The PictureBox control is used for displaying images on the form. The Image property of the control allows you to set an image both at design time or at run time.

Properties of the PictureBox Control

Properties	Means
ErrorImage	Gets or specifies an image to be displayed when an error occurs during the image-loading process or if the image load is cancelled.
Image	Gets or sets the image that is displayed in the control.
ImageLocation	Gets or sets the path or the URL for the image displayed in the control.
SizeMode	Determines the size of the image to be displayed in the control. This property takes its value from the PictureBoxSizeMode enumeration, which has values – Normal — Standard picture box behavior (the upper-left corner of the image is placed at upper left in the picture box). StretchImage — Allows you to stretch the image in code. AutoSize — Fits the picture box to the image. CenterImage — Centers the image in the picture box.
Text	Gets or sets the text for the picture box.

Method of PictureBox

Method	Means
FromFile	This method is versatile and can load images from bitmap (.bmp), icon (.ico) or metafile (.wmf), JPEG (.jpg), GIF (.gif) files, and other types of files. Example:

	<code>PictureBox1.Image= Image.FromFile("Path\image.jpg")</code>
--	--

ScrollBar

The ScrollBar controls display vertical and horizontal scroll bars on the form. This is used for navigating through large amount of information. There are two types of scroll bar controls: **HScrollBar** for horizontal scroll bars and **VScrollBar** for vertical scroll bars. These are used independently from each other.

You use the Minimum and Maximum (formerly Min and Max) properties to set the range of values the user can select using the scroll bar. The LargeChange property sets the scroll increment that happens when the user clicks in the scroll bar but outside the scroll box. The SmallChange property sets the scroll increment when the user clicks the scroll arrows at each end of the scroll bar.

The **default values** for the **Minimum, Maximum, SmallChange, and LargeChange** values are **0, 100, 1, and 10 respectively**. You get the actual setting of a scroll bar with its Value property.

Properties of the ScrollBar Control

The following are some of the commonly used properties of the ScrollBar control –

Properties	Means
AutoSize	Gets or sets a value indicating whether the ScrollBar is automatically resized to fit its contents.
BackColor	Gets or sets the background color for the control.
ForeColor	Gets or sets the foreground color of the scroll bar control.
LargeChange	Gets or sets a value to be added to or subtracted from the Value property when the scroll box is moved a large distance.
Maximum	Gets or sets the upper limit of values of the scrollable range.
Minimum	Gets or sets the lower limit of values of the scrollable range.

SmallChange	Gets or sets the value to be added to or subtracted from the Value property when the scroll box is moved a small distance.

Timers

Timers are also very useful controls, because they let you create periodic events. Timers are no longer controls but components, and they do not appear in a window at run time. At design time, they appear in the component tray underneath the form you've added them to. There's a timer at work behind the scenes in the Timers i.e., in backend.

Properties of Timer

Properties	Means
Enabled	Gets/sets whether the timer is running.
Interval	Gets/sets the time (in milliseconds) between timer ticks. Example: Timer1.Interval = 2000

Methods of Timer

Method	Means
Start	Starts the timer. Example: Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click Timer1.Start() End Sub

Stop	<p>Stops the timer.</p> <p>Example:</p> <pre>Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click Timer1.Stop() End Sub</pre>
------	--

Tool Tip

All Windows users know what tool tips are—they're those small windows that appear with explanatory text when you let the mouse rest on a control or window. That's what tool tips are used for—to give quick help when the mouse rests on an item.

To connect a tool tip with a control, you use its **SetToolTip method**. To connect the tool tip to Button1, you can use this code:

```
ToolTip1.SetToolTip(Button1, "This is a button")
```

You also can use the **GetToolTip method** to get information about a tool tip object.

The important properties for tool tip controls are **Active**, which must be **set to True** for the tool tip to appear, and **AutomaticDelay**, which **sets the length of time** that the tool tip is shown, how long the user must point at the control for the tool tip to appear, and how long it takes for subsequent tool tip windows to appear. **IsBalloon** property set to **True** indicates whether tool tip will appear **balloon window** as shown below fig.

```
Public Class Form1
```

```
    Private Sub Button1_GotFocus(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button1.GotFocus
        ToolTip1.IsBalloon = True
        ToolTip1.SetToolTip(Button1, "This is a button")
        ToolTip1.GetToolTip(Button1)
```

```
    End Sub
End Class
```




Fig: Tool Tip at work when Gotfocus event occurs