US05CCSC03: VISUAL PROGRAMMING Through VB.NET

Unit-III V.B. NET Controls and Error Handling

➢ What is Control ?

A control is an object that can be drown on to the from making control are visible object. The control are to enable or and hence user interaction with the Application. All controls are unique with the its features example of this control are Textbox, Button, Checkbox, Label, Tree view etc. All the control have properties methods and events.

Control class is the base class of all the windows controls. We can work with controls in two ways at Design time and at Run time.

Some of common properties of the controls are given the below.

1. **Back color -** Background Color.
2. **Background Image** -Background Image.
3. **Bottom** -The distance between the Bottom of the control and the top of the its contains.
4. **Context menu** – It gets or sets the shortcut menu for the control.
5. **Cursor property** – It the cursor.
6. **Unable property** - It gets or sets a value indicting if they control is enable.
7. **Font** – To gets or sets the font for the control.
8. **Fore color** – It gets or sets the font color of the control.
9. **Height** – It gets or sets the height of control.
10. **Left** – It gets or sets the X co-ordinate of the control left page in pixels.
11. **Location** – It gets or sets the co-ordinate of the upper left corner of the control.
12. **Locked** – It gets or sets the recise or move the control at design time.
13. **Name property** – It gets or sets name of the control.
14. **Right** – It retunes the distance between the right edge of the control and the left edge of it's container.
15. **Size** – It gets or sets size of the control in pixels.
16. **Tab index** – It gets or sets tab order of this control in it's container.
17. **Tab Stop** – It gets or sets a value specify if the user can Tab or Shift + Tab to this control with the Tab key.
18. **Text** – It gets or sets text for this control.
19. **Top** – It gets or sets the top co-ordinate of the control.
20. **Visible** – It gets or sets a value for visibility of the control.
21. **Width** – Its gets or sets width of the control.

Some common event of the control are as given below :

1. **Click** – It occurs then the control it click
2. **Got Focus** – It occurs when the control received focus.
3. **Key down** – It occurs when the key is pressed the control has focus.
4. **Key pressed –** It occurs when a key is pressed one control has focus.
5. **Key up –** It occurs when a key is released by the control has focus.
6. **Lost focus –** It occurs when the control user losses focus.
7. **Mouse click –** It occurs when the control is click by the mouse.
8. **Mouse down** – It occurs when the mouse pointer is over the control and the mouse button is pressed.
9. **Mouse enter** – It occurs when the mouse pointer enter the controls the mouse over it occurs when the mouse pointer rests on the controls.
10. **Mouse leave** – It occurs when the mouse pointer lives the control.
11. **Mouse move** - It occurs when the mouse pointer is moved over the control.
12. **Mouse up-** It occurs when the mouse pointer is over the control and a mouse button is released.
13. **Mouse while** - It occurs when the mouse while moves when the control has foucs.

Button Control

One of the most popular control in Visual Basic is the Button Control (previously Command Control). They are the controls which we click and release to perform some action. Buttons are used mostly for handling events in code, say, for sending data entered in the form to the database and so on. The default event of the Button is the Click event and the Button class is based on the ButtonBase class which is based on the Control class.

Button Event

The default event of the Button is the Click event. When a Button is clicked it responds with the Click Event. The Click event of Button looks like this in code:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As_ System.EventArgs)
Handles Button1.Click
'You place the code here to perform action when Button is clicked
End Sub
```

Working with Buttons

Well, it's time to work with Buttons. Drag a Button from the toolbox onto the Form. The default text on the Button is Button1. Click on Button1 and select it's properties by pressing F4 on the keyboard or by selecting View->Properties Window from the main menu. That displays the Properties for Button1.

Appearance

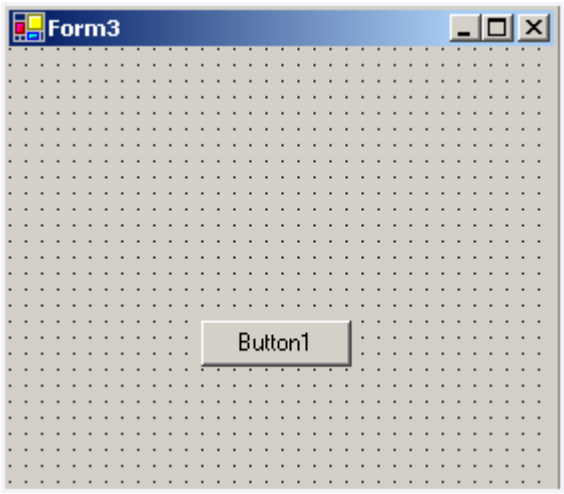
Appearance section of the properties window allows us to make changes to the appearance of the Button. With the help of BackColor and Background Image properties we can set a background color and a background image to the button. We set the font color and font style for the text that appears on button with ForeColor and the Font property. We change the appearance style of the button with the FlatStyle property. We can change the text that appears on button with the Text property and with the TextAlign property we can set where on the button the text should appear from a predefined set of options.

Behavior

Notable Behavior properties of the Button are the Enabled and Visible properties. The Enabled property is set to True by default which makes the button enabled and setting it's property to False makes the button Disabled. With the Visible property we can make the Button Visible or Invisible. The default value is set to True and to make the button Invisible set it's property to False.

Layout

Layout properties are about the look of the Button. Note the Dock property here. A control can be docked to one edge of its parent container or can be docked to all edges and fill the parent container. The default value is set to none. If you want to dock the control towards the left, right, top, bottom and center you can do that by selecting from the button like image this property displays. With the Location property you can change the location of the button. With the Size property you can set the size of the button. Apart from the Dock property you can set it's size and location by moving and stretching the Button on the form itself.



Label Control

Label Control is usually used to display text that cannot be edited by the user. But using the properties or code that is displayed can be changed.

Labels are those controls that are used to display text in other parts of the application. They are based on the Control class.

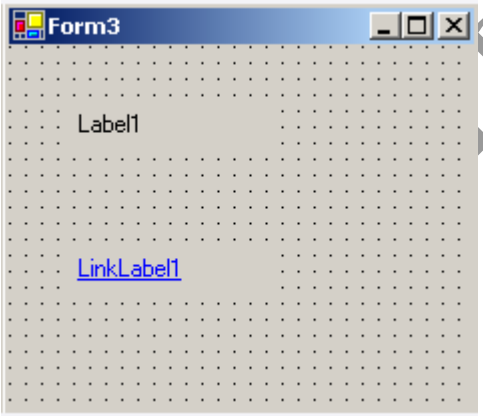
Notable property of the label control is the text property which is used to set the text for the label.

Label Event

The default event of Label is the Click event which looks like this in code:

```
Private Sub Label1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)_
Handles Label1.Click

End Sub
```



Link Label Control

Link Label Control is an extension of Label control to display one or more hyperlinks. This Class is inherited from the Label Class so all the functions of the label control can be used with the link label control.

LinkLabel, new in Visual Basic .NET, is a standard control that lets you embed web-style links in a form.

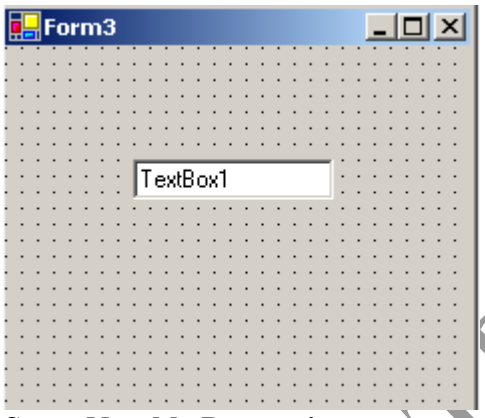
LinkLabel is much more convenient and trouble free than older techniques. But, in sync with .NET architecture, LinkLabel is designed to be used with other objects to do the whole job. You still need to use a separate command to start an email or browser for example. Example code is included below.

The basic idea is to put the email address or web URL into the Text property of a LinkLabel component, then when the label is clicked, the **LinkClicked** event is triggered. There are well over a hundred methods and objects available for the LinkLabel object including properties to handle

everything you might want to do with a link like changing the color, text, position, how it behaves when you click it ... whatever! You can even check mouse buttons and positions and test whether the **Alt**, **Shift**, or **Ctrl** keys are pressed when the link is clicked.

TextBox Control

- This controls looks like a box and accepts input from the user.
- They are some time called and edit field or edit control.
- Typically a textbox control is used to display or accept as input a single line of text.
- Text can display multi line you can do that by setting the textbox multi line property is true.
- Using the scrollbar property's there are four ways to add scrollbar to a textbox value of scrollbar is 0 as nun, 1 as Horizontal 2 as vertical, 3 as both.
- You can use text line properties you can set it to lest justified, right justified and centered.
- You can use text line properties to make a textbox read only setting this properties to true means that the user cannot enter text into the textbox.
- You can disable a textbox by setting its enabled property to false.
- By default a textbox holds up to 32767 char. But when displaying multi line a textbox holds up to 2GB of text.
- To convert a slandered textbox into a password box you just assign some char to the textboxes password char. Property.
- The text property contains in the control to set or get text.



Some Notable Properties:

Some important properties in the Behavior section of the Properties Window for TextBoxes.
Enabled: Default value is True. To disable, set the property to False.
Multiline: Setting this property to True makes the TextBox multiline which allows to accept multiple lines of text. Default value is False.
PasswordChar: Used to set the password character. The text displayed in the TextBox will be the character set by the user. Say, if you enter *, the text that is entered in the TextBox is displayed as *.
ReadOnly: Makes this TextBox readonly. It doesn't allow to enter any text.
Visible: Default value is True. To hide it set the property to False.
TextAlign: Allows to align the text from three possible options. The default value is left and you can set the alignment of text to right or center.
Scrollbars: Allows to add a scrollbar to a Textbox. Very useful when the TextBox is multiline. You have four options with this property. Options are are None, Horizontal, Vertical and Both. Depending on the size of the TextBox anyone of those can be used.

TextBox Event

The default event of the TextBox is the TextChanged Event which looks like this in code:

```
Private Sub TextBox1_TextChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles TextBox1.TextChanged
```

```
End Sub
```

Passwords

If you need a TextBox for accepting a password you should set it's PasswordChar property to a character. The character you choose will be displayed in the TextBox instead of what is actually typed. Good characters to choose include * and ●.

Radio Buttons

A radio button allows the user to choose exactly one of a predefined set of options. Radio buttons are arranged in groups of two or more and displayed on screen a list of circular holes that can contain white space (for unselected) or a dot (for selected). Each radio button can show a caption describing the choice that this radio button represents. This is done by setting the radio button's text property.

A RadioButton control provides a round interface to select one option from a number of options. Radio buttons are usually placed in a group on a container control such as a Panel or a GroupBox and one of them is selected.

Creating a RadioButton

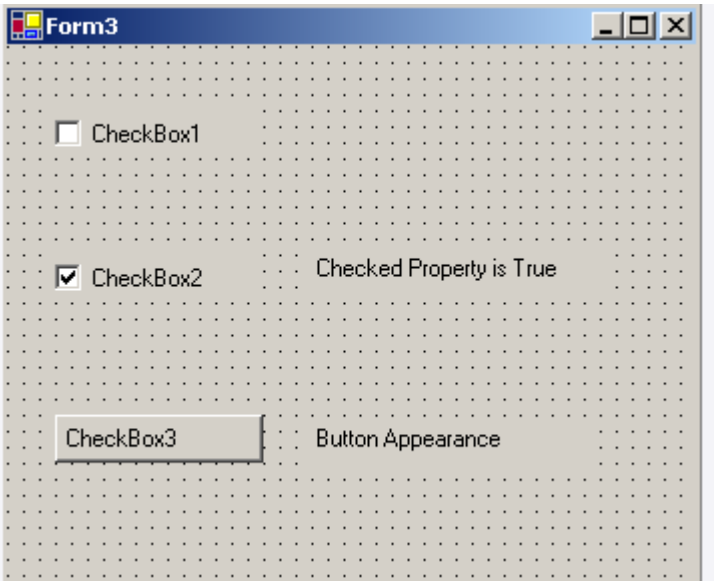
We can create a RadioButton control using a Forms designer at design-time or using the RadioButton class in code at run-time (also known as dynamically).

To create a RadioButton control at design-time, you simply drag and drop a RadioButton control from Toolbox to a Form in Visual Studio. After you drag and drop a RadioButton on a Form, the RadioButton looks like Figure 1. Once a RadioButton is on the Form, you can move it around and resize it using mouse and set its properties and events.

Creating a RadioButton control at run-time is merely work of creating an instance of RadioButton class, set its properties and add RadioButton class to the Form controls.

Checkboxes

CheckBoxes are those controls which gives us an option to select, say, Yes/No or True/False. A checkbox is clicked to select and clicked again to deselect some option. When a checkbox is selected a check (a tick mark) appears indicating a selection. The CheckBox control is based on the TextBoxBase class which is based on the Control class. Below is the image of a Checkbox.



Notable Properties

Important properties of the CheckBox in the Appearance section of the properties window are:
Appearance: Default value is Normal. Set the value to Button if you want the CheckBox to be displayed as a Button.
BackgroundImage: Used to set a background image for the checkbox.
CheckAlign: Used to set the alignment for the CheckBox from a predefined list.
Checked: Default value is False, set it to True if you want the CheckBox to be displayed as checked.
CheckState: Default value is Unchecked. Set it to True if you want a check to appear. When set to Indeterminate it displays a check in gray background.
FlatStyle: Default value is Standard. Select the value from a predefined list to set the style of the checkbox.
Important property in the Behavior section of the properties window is the ThreeState property which is set to False by default. Set it to True to specify if the Checkbox can allow three check states than two.

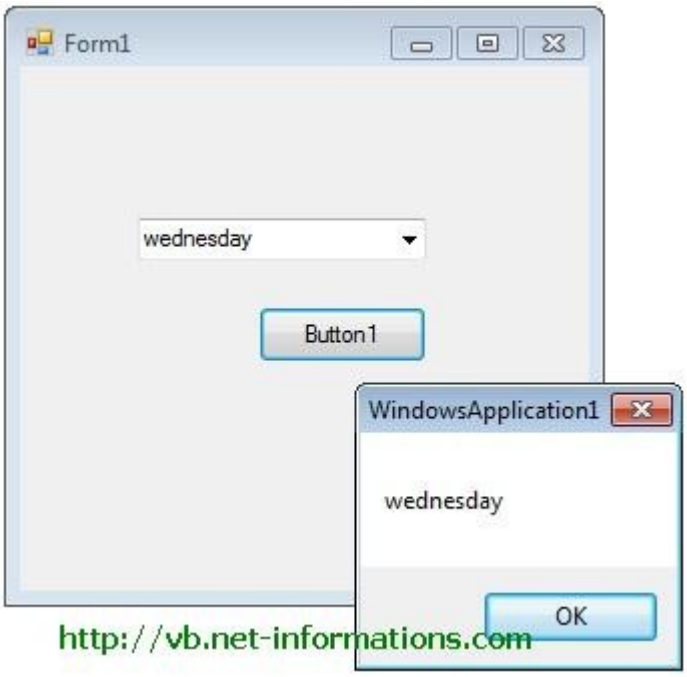
CheckBox Event

The default event of the CheckBox is the CheckedChange event which looks like this in code:

```
Private Sub CheckBox1_CheckedChanged(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles CheckBox1.CheckedChanged

End Sub
```

Combo Box :- The ComboBox control is similar to the ListBox control in the sense that it contains multiple items and the user may select one, but it typically occupies less space onscreen. The ComboBox is practically an expandable ListBox control, which can grow when the user wants to make a selection and retract after the selection is made. Normally, the ComboBox control displays one line with the selected item, as this control doesn't allow multiple item selection. The essential difference, however, between ComboBox and ListBox controls is that the ComboBox allows the user to specify items that don't exist in the list.
VB.Net controls are located in the Toolbox of the development environment, and you use them to create objects on a form with a simple series of mouse clicks and dragging motions. The ComboBox control , which lets the user choose one of several choices.



The user can type a value in the text field or click the button to display a drop down list. In addition to display and selection functionality, the ComboBox also provides features that enable you to efficiently add items to the ComboBox.

```
ComboBox1.Items.Add("Sunday")
```

You can set which item should shown while it displaying in the form for first time.

```
ComboBox1.SelectedItem = ComboBox1.Items(3)
```

When you run the above code it will show the forth item in the combobox. The item index is starting from 0.

If you want to retrieve the displayed item to a string variable , you can code like this

```
Dim var As String
var = ComboBox1.Text
```

You can remove items from a combobox in two ways. You can remove item at a the specified index or giving a specified item by name.

```
ComboBox1.Items.RemoveAt(1)
```

The above code will remove the second item from the combobox.

```
ComboBox1.Items.Remove("Friday")
```

ListBox

The ListBox control displays a list of items from which we can make a selection. We can select one or more than one of the items from the list. The ListBox control is based on the ListControl class which is based on the Control class. The image below displays a ListBox.

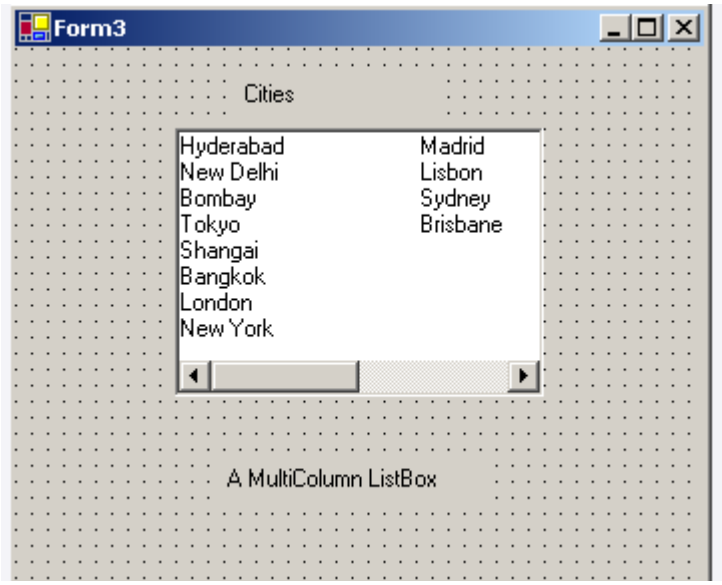
A list box control display a list of items from which the user can select one or more. List box is based you for displaying large number of choices. A scroll bar automatically appears when many items in the list box. By default we can select only single items from the list box.

- Items property gives list of the data in the list box.

- Coding for add items in the list box.

Listbox1. Items. Add (“Brisbane”)

Or form the property windows add items in the list box using the items property when we click on items property of the list box then we can view as editor. Type the list of data as we want. The list will add items design time.



Notable Properties of the ListBox

In the Behavior Section

HorizontalScrollbar: Displays a horizontal scrollbar to the ListBox. Works when the ListBox has MultipleColumns.

MultiColumn: The default value is set to False. Set it to True if you want the list box to display multiple columns.

ScrollAlwaysVisible: Default value is set to False. Setting it to True will display both Vertical and Horizontal scrollbar always.

SelectionMode: Default value is set to one. Select option None if you do not any item to be selected. Select it to MultiSimple if you want multiple items to be selected. Setting it to MultiExtended allows you to select multiple items with the help of Shift, Control and arrow keys on the keyboard.

The value of selection mode are as given bellow.

(1) Multi extend :- multiple items can select + user can use the shift control and arrow keys to make selections.

(2) Multi simple :-- multiple items can select, then selection mode non. no items can select.

(3) One :-- Only one item can select.

- you can use the selected index changed event which is the default event for list boxes to handle the case where the selected item changes in the list box.

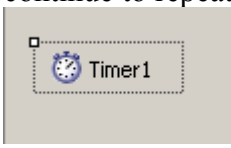
Sorted: Default value is set to False. Set it to True if you want the items displayed in the ListBox to be sorted by alphabetical order.

In the Data Section

Notable property in the Data section of the Properties window is the Items property. The Items property allows us to add the items we want to be displayed in the list box. Doing so is simple, click on the ellipses to open the String Collection Editor window and start entering what you want to be displayed in the ListBox. After entering the items click OK and doing that adds all the items to the ListBox.

Timer Control

A Timer Control is a control that will execute code at intervals. It is not visible at runtime. The interval can be set in the properties in the measurement of milliseconds. The timer control will continue to repeat its code at your interval until the control is disabled.

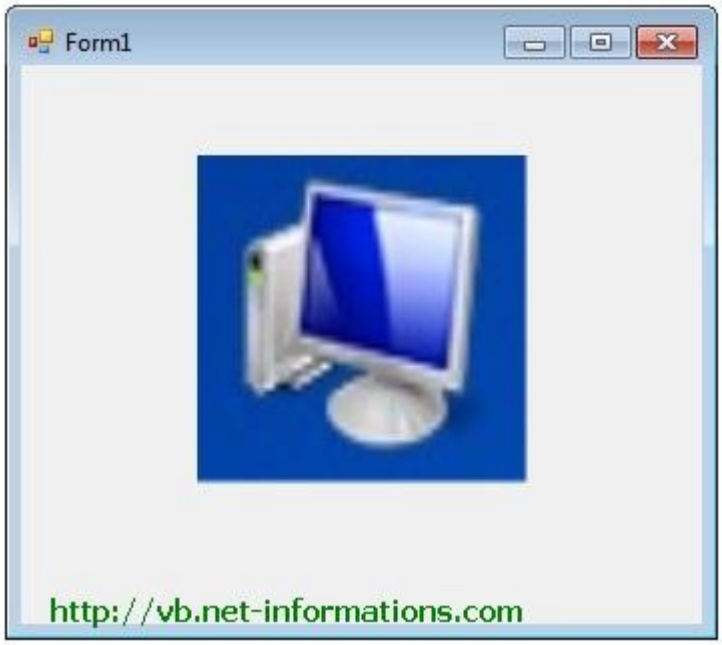


Timer Properties and Events

Visual Basic 6.0	Visual Basic .NET Equivalent
Enabled property	**Enabled**
Index property	No equivalent property. The **Timer** is a component and cannot be a part of a control array.
Interval property	**Interval** **Note** The behavior of the **Interval** property has changed. For more information, see Timer Interval property behavior has changed.
Name property	No equivalent property. For more information, see Name Property Changes in Visual Basic .NET.
Parent property	No equivalent property. The **Timer** is a component and cannot have a parent.
Tag property	No equivalent. For more information, see Tag Property Changes in Visual Basic .NET.

Picture Box :

PictureBox control is used to display image. The images displayed can be any format like Bitmap, JPEG, and GIF, PNG or any other image format files. The PictureBox control is based on the Control class.



You can set the Image property to the Image you want to display, either at design time or at run time. You can programmatically change the image displayed in a picture box, which is particularly useful when you use a single form to display different pieces of information.

```
PictureBox1.Image = Image.FromFile("C:\testImage.jpg")
```

The SizeMode property, which is set to values in the PictureBoxSizeMode enumeration, controls the clipping and positioning of the image in the display area.

```
PictureBox1.SizeMode = PictureBoxSizeMode.StretchImage
```

There are five different PictureBoxSizeMode is available to PictureBox control.

```
AutoSize            - Sizes the picture box to the image.
CenterImage         - Centers the image in the picture box.
Normal              - Places the upper-left corner of the image at upper
                      left in the picture box
StretchImage        - Allows you to stretch the image in code
```

You can change the size of the display area at run time with the ClientSize property.

```
pictureBox1.ClientSize = New Size(xSize, ySize)
```

HSCROLLBAR AND VSCROLLBAR

Windows Forms **ScrollBar** controls are used to provide easy navigation through a long list of items or a large amount of information by scrolling either horizontally or vertically within an application or control. Scroll bars are a common element of the Windows interface, so the **ScrollBar** control is often used with controls that do not derive from the ScrollableControl class. Similarly, many developers choose to incorporate the **ScrollBar** control when authoring their own user controls.

The **HScrollBar** (horizontal) and **VScrollBar** (vertical) controls operate independently from other controls and have their own set of events, properties, and methods. **ScrollBar** controls are not the same as the built-in scroll bars that are attached to text boxes, list boxes, combo boxes, or MDI forms (the **TextBox** control has a ScrollBars property to show or hide scroll bars that are attached to the control).

The **ScrollBar** controls use the Scroll event to monitor the movement of the scroll box (sometimes referred to as the thumb) along the scroll bar. Using the **Scroll** event provides access to the scroll bar value as it is being dragged.

Value Property

The Value property (which, by default, is 0) is an **integer** value corresponding to the position of the scroll box in the scroll bar. When the scroll box position is at the minimum value, it moves to the left-most position (for horizontal scroll bars) or the top position (for vertical scroll bars). When the scroll box is at the maximum value, the scroll box moves to the right-most or bottom position. Similarly, a value halfway between the bottom and top of the range places the scroll box in the middle of the scroll bar.

In addition to using mouse clicks to change the scroll-bar value, a user can also drag the scroll box to any point along the bar. The resulting value depends on the position of the scroll box, but it is always within the range of the Minimum to Maximum properties set by the user.

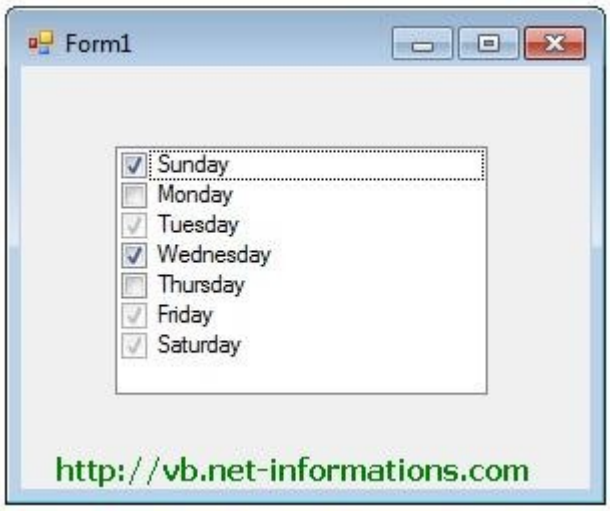
LargeChange and SmallChange Properties

When the user presses the PAGE UP or PAGE DOWN key or clicks in the scroll-bar track on either side of the scroll box, the **Value** property changes according to the value set in the LargeChange property.

When the user presses one of the arrow keys or clicks one of the scroll-bar buttons, the **Value** property changes according to the value set in the SmallChange property.

Checked ListBox Control

The CheckedListBox control gives you all the capability of a list box and also allows you to display a check mark next to the items in the list box.



To add objects to the list at run time, assign an array of object references with the AddRange method. The list then displays the default string value for each object.

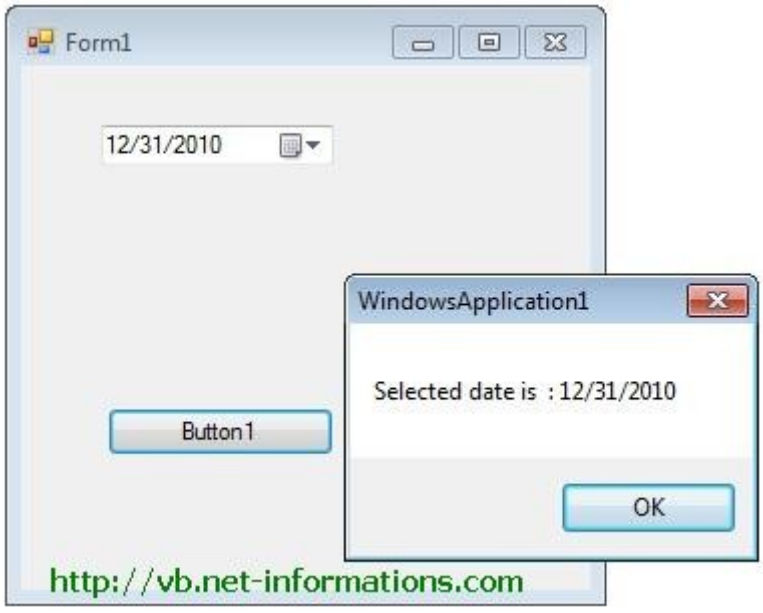
```
Dim days As String() =  {"Sunday", "Monday", "Tuesday"}
checkedListBox1.Items.AddRange(days)
```

You can add individual items to the list with the Add method. The CheckedListBox object supports three states through the CheckState enumeration: Checked, Indeterminate, and Unchecked.

```
CheckedListBox1.Items.Add("Sunday", CheckState.Checked)
CheckedListBox1.Items.Add("Monday", CheckState.Unchecked)
CheckedListBox1.Items.Add("Tuesday", CheckState.Indeterminate)
```

DateTimePicker Control

The DateTimePicker control allows you to display and collect date and time from the user with a specified format.



The DateTimePicker control prompts the user for a date or time using a graphical calendar with scroll arrows. The most important property of the DateTimePicker is the Value property, which holds the selected date and time.

```
DateTimePicker1.Value = "12/31/2010"
```

The Value property is set to the current date by default. You can use the Text property or the appropriate
member of Value to get the date and time value.

```
Dim idate As String
idate = DateTimePicker1.Value
```

The control can display one of several styles, depending on its property values. The values can be displayed in four
formats, which are set by the Format property: Long, Short, Time, or Custom.

```
DateTimePicker1.Format = DateTimePickerFormat.Short
```

⇨ **Property of date time picker : -**

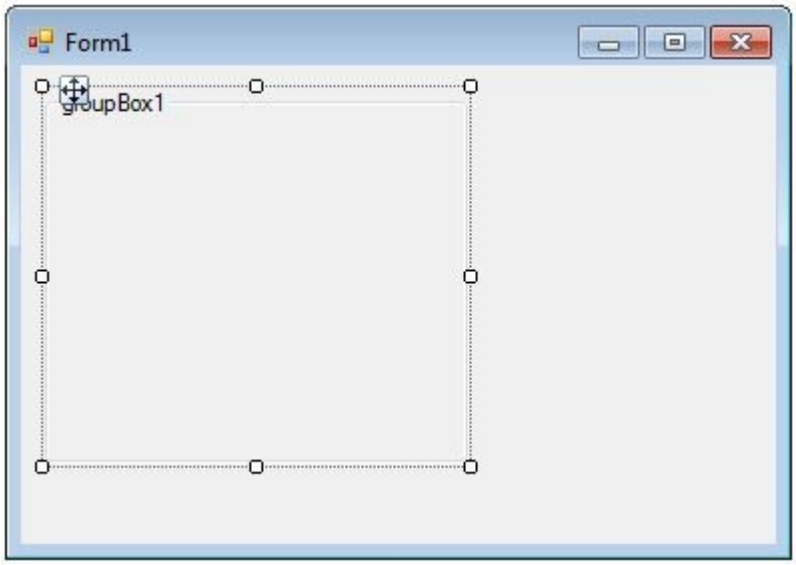
(1) It's specifying the minimum date value of the date time picker control not that this field is read only.
(2) **Min date time** : It's specifying the minimum date value of the date time picker control note that this field is read only.
(3) **Calendar font property** : - It gets or sets the font style for the calendar.
(4) **Check property** : It gets or sets whether the value property holds a valid date time value.
(5) **Customs format** : It gets or sets a custom date time format string.
(6) **Dropdown align** : It gets or sets the alignment of the dropdown calendar on the date time control.
(7) **Format** : It gets or sets the format of date and time.
(8) **Max date** : It gets or sets the maximum selectable data and time.
(9) **Min date** : It gets or sets the minimum selectable data and time.
(10) **Show check box** :It gets or sets if a check box should appear to the left a selected.
(11) **Show up down property** - It gets or sets if and up down controls should be use to adjust date time values.
(12) **Text property : -**It gets or sets the text in the control.
(13**) Value property :** It gets or sets the date time value.

*** Event of date time picker control : -**

1.) **Value change :-** It occurs when the value properties changes.
2.) **Format change event :** -It occurs when the format property values as change.
3.) **Dropdown event :-** It occurs when the dropdown calendar occurs.
4.) **Close up event :** It occurs when the dropdown calendar this appear.

Group Box Control

We can create a GroupBox control using a Forms designer at design-time or using the GroupBox class in code at run-time (also known as dynamically).
To create a GroupBox control at design-time, you simply drag and drop a GroupBox control from Toolbox to a Form in Visual Studio. After you drag and drop a GroupBox on a Form, the GroupBox looks like Figure 1. Once a GroupBox is on the Form, you can move it around and resize it using mouse and set its properties and events.



RichTextBox

Can easily save Richtext files with Colours and Fonts. Has all the qualities of a textbox but with built in save method(richtextbox1.savefile(pathname)) and openfile(Richtextbox1.openfile(filename)).

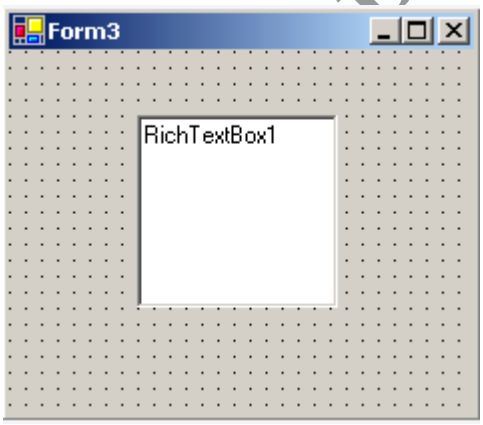
The **RichTextBox** control is the core of a full-blown word processor. It provides all the functionality of a TextBox control; it can handle multiple typefaces, sizes, and attributes, and offers precise control over the margins of the text (see Figure 4.16). You can even place images in your text on a RichTextBox control (although you won't have the kind of control over the embedded images that you have with Microsoft Word).

The fundamental property of the **RichTextBox** control is its Rtf property. Similar to the Text property of the TextBox control, this property is the text displayed on the control. Unlike the Text property, however, which returns (or sets) the text of the control but doesn't contain formatting information, the Rtf property returns the text along with any formatting information. Therefore, you can use the RichTextBox control to specify the text's formatting, including paragraph indentation, font, and font size or style.

RichTextBoxes are similar to TextBoxes but they provide some advanced features over the standard TextBox. RichTextBox allows formatting the text, say adding colors, displaying particular font types and so on. The RichTextBox, like the TextBox is based on the TextBoxBase class which is based on the Control class. These RichTextBoxes came into existence because many word processors these days allow us to save text in a rich text format. With RichTextBoxes we can also create our own word processors. We have two options when accessing text in a RichTextBox, text and rtf (rich text format). Text holds text in normal text and rtf holds text in rich text format. Image of a RichTextBox is shown below.

It's an advance version of text box different between text box and rich text box is like notepad and WordPad. It provides more advance text formatting options it can load RTF, TXT promote files for reading or editing the word pad save it's file in RTF format. The full form of RTF is rich text format . rich textbox allows formatting the text say adding color, displaying particular font typed selected text effect. Bullets ,alignment , indents and show on by default rich textbox is m

multi line by default maximum length of it to 2147483647 some property of rich textbox are bullet indent zoom factor selection font , selection color alignment etc. Methods of rich textbox are like load file, save file , cut ,copy , paste, undo ,redo , replace etc. Best example of rich textbox is word pad.



RichTextBox Event

The default event of RichtextBox is the TextChanged event which looks like this in code:

```
Private Sub RichTextBox1_TextChanged(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles RichTextBox1.TextChanged

End Sub
```

Dialogs : Common dialog of control :

- VB.NET contains built in dialog box which allow us to create our own file o open , file save , font and color dialogs control.

- The common dialogs controls are give bellow.

1.) Color dialogue
2.) Fount dialogue
3.) Open file dialogue
4.) Save file
5.) Print dialogue
6.) Print preview dialogue
7.) Page setup dialogue

-To display common dialogue controls at run-time we need to called dot show Dialogue method of the dialogue control . shows dialogue method is require to called open any dialogue control.

For example

```
If color dialogueone . show ialogue = windows .forms. dialogueresult. ok then
        any me . back color = color dialogue 1 . color
end 1f.
```

OpenFileDialog

An OpenFileDialog displays the standard "Open" dialog. It lets the user browse for a file.

```
Dim DialogResult As Int32 = OpenFileDialog1.ShowDialog()

Dim strFileName = OpenFileDialog1.FileName()
```

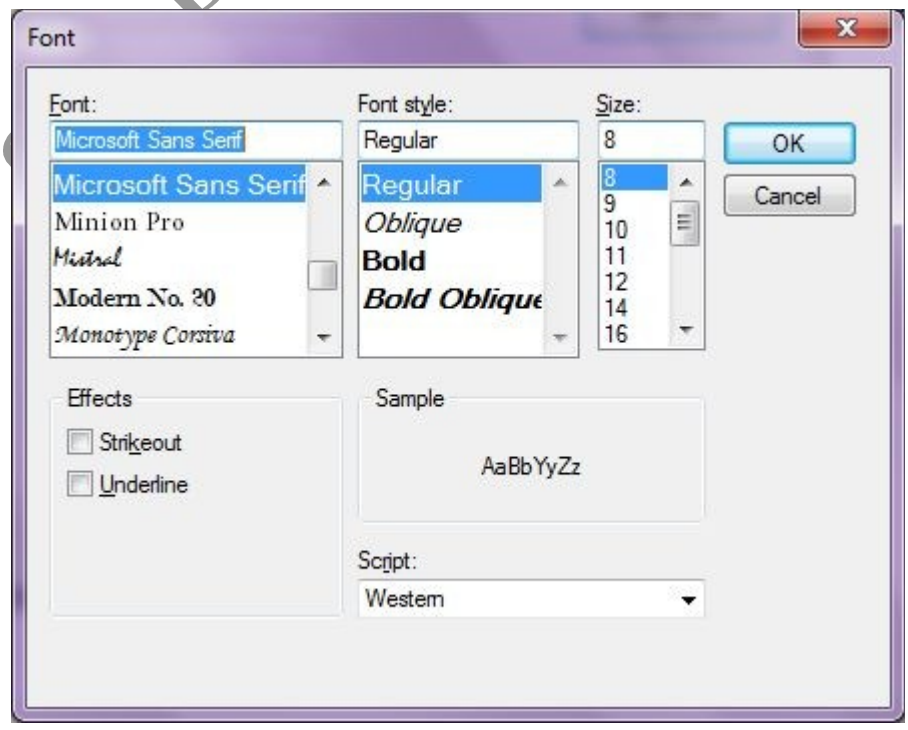
SaveFileDialog

A SaveFileDialog displays the standard "Save" dialog. It lets the user browse for a directory to save files and enter a filename. It can (optionally) automatically append extensions to the filename.

FontDialog

A FontDialog lets the user to select a font from a list of installed fonts.

A FontDialog control is used to select a font from available fonts installed on a system. A typical Font Dialog looks like Figure 1 where you can see there is a list of fonts, styles, size and other options. Please note a FontDialog may have different fonts on different system depending on what fonts are install on a system.



Creating a FontDialog

We can create a FontDialog control using a Forms designer at design-time or using the FontDialog class in code at run-time (also known as dynamically). Unlike other Windows Forms controls, a FontDialog does not have and not need visual properties like others. You use a FontDialog to list all the fonts and select one of them and usually apply selected fonts on controls or some contents.

Design-time

To create a FontDialog control at design-time, you simply drag and drop a FontDialog control from Toolbox to a Form in Visual Studio. After you drag and drop a FontDialog on a Form, the FontDialog looks like Figure 2.

Adding a FontDialog to a Form adds following two lines of code.

```
Friend WithEvents FontDialog1 As System.Windows.Forms.FontDialog
Me.FontDialog1 = New System.Windows.Forms.FontDialog()
```

Run-time

Creating a FontDialog control at run-time is merely a work of creating an instance of FontDialog class, set its properties and add FontDialog class to the Form controls.

First step to create a dynamic FontDialog is to create an instance of FontDialog class. The following code snippet creates a FontDialog control object.

```
Dim fontDlg As New FontDialog
```

ShowDialog method of FontDialog displays the FontDialog. The following code snippet calls the ShowDialog method.

```
fontDlg.ShowDialog()
```

FontDialog Properties

Show Properties

A FontDialog control can have a color drop down that allows users to select a color of the selected font. A FontDialog can also have Apply and Help buttons as you can see in Figure 3.

Figure 3

ShowColor, ShowApply, and ShowHelp properties can be set to true to show these options. The ShowEffects property represents whether the dialog box contains controls that allow the user to specify strikethrough, underline, and text color options.

The following code snippet sets these properties to true.

```
fontDlg.ShowColor = True
fontDlg.ShowApply = True
fontDlg.ShowEffects = True
fontDlg.ShowHelp = True
```

Maximum and Minimum Font Size

We can restrict the maximum and minimum font size by setting MaxSize and MinSize properties. The following code snippet sets these properties to 40 and 22 respectively.

```
fontDlg.MaxSize = 40
fontDlg.MinSize = 22
```

Font and Color

The Font and Color properties represent the selected font and color in a FontDialog. The following code snippet gets these properties and sets font and color of a TextBox and a Label controls.

```
If (fontDlg.ShowDialog() = DialogResult.OK) Then
        TextBox1.Font = fontDlg.Font
        Label1.Font = fontDlg.Font
        TextBox1.BackColor = fontDlg.Color
        Label1.ForeColor = fontDlg.Color
End If
```

ColorDialog

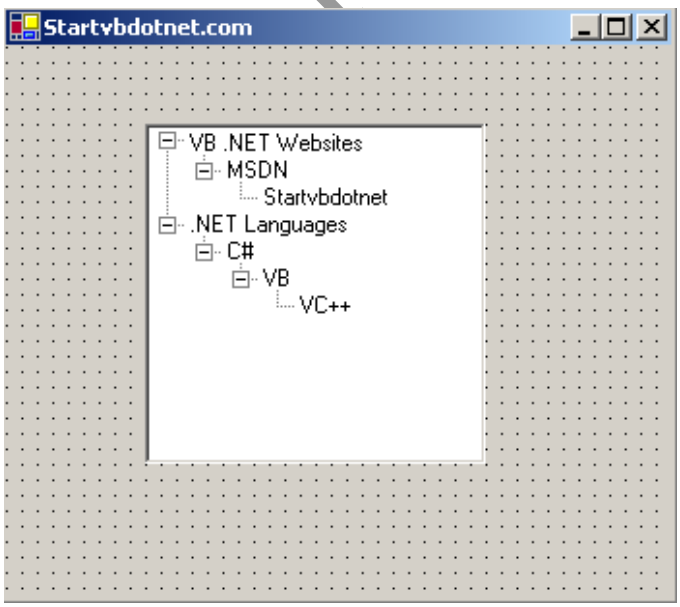
A ColorDialog lets the user select predefined color or specify a custom color.

The Color dialog box, is one of the simplest dialog boxes. Its Color property returns the color selected by the user or sets the initially selected color when the user opens the dialog box.

The following statements set the initial color of the ColorDialog control, display the dialog box, and then use the color selected in the control to fill the form. First, place a ColorDialog control in the form and then insert the following statements in a button's Click event handler:

```
Private Sub Button1 Click(...) Handles Button1.Click
        ColorDialog1.Color = Me.BackColor
If ColorDialog1.ShowDialog = Windows.Forms.DialogResult.OK Then
            Me.BackColor = ColorDialog1.Color
End If
End Sub
```

TreeView_ The tree view control is used to display a hierarchy of nodes (both parent, child). You can expand and collpase these nodes by clicking them. This control is similar to Windows Explorer which displays a tree view in it's left pane to list all the folders on the hard disk. Below is the image of a Tree View control.

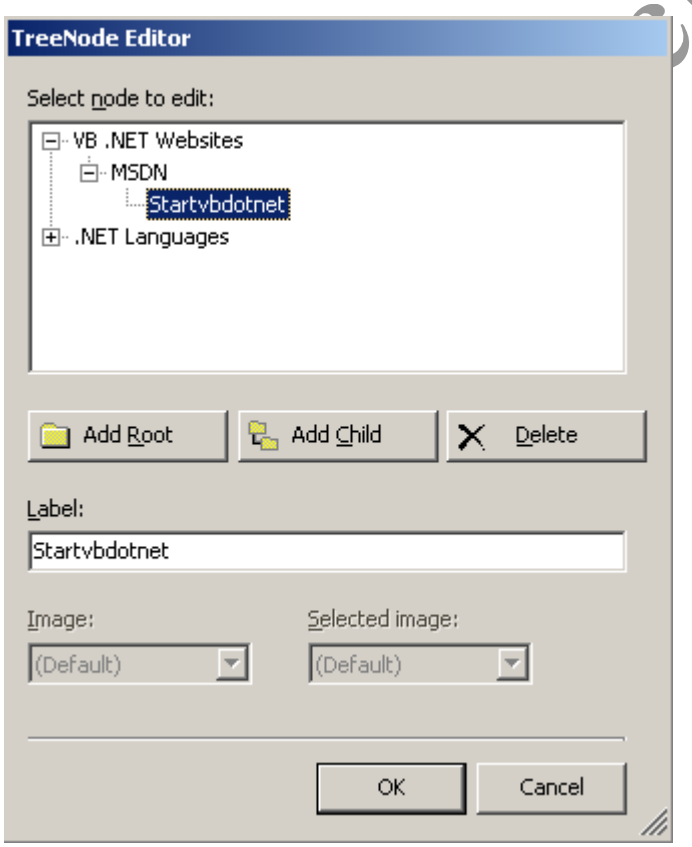


Notable Properties of TreeView

Bounds: Gets the actual bound of the tree node
Checked: Gets/Sets whether the tree node is checked
FirstNode: Gets the first child tree node
FullPath: Gets the path from the root node to the current node
ImageIndex: Gets/Sets the image list index of the image displayed for a node
Index: Gets the location of the node in the node collection
IsEditing: Gets whether the node can be edited
IsExpaned: Gets whether the node is expaned
IsSelected: Gets whether the node is selected
LastNode: Gets the last child node
NextNode: Gets the next sibling node
NextVisibleNode: Gets the next visible node
NodeFont: Gets/Sets the font for nodes
Nodes: Gets the collection of nodes in the current node
Parent: Gets the parent node of the current node
PrevNode: Gets the previous sibling node
PrevVisibleNode: Gets the previous visible node
TreeView: Gets the node's parent tree view

Working with Tree Views

Drag a Tree View control on to a form and to add nodes to it select the nodes property in the properties window, which displays the TreeNode editor as shown below.



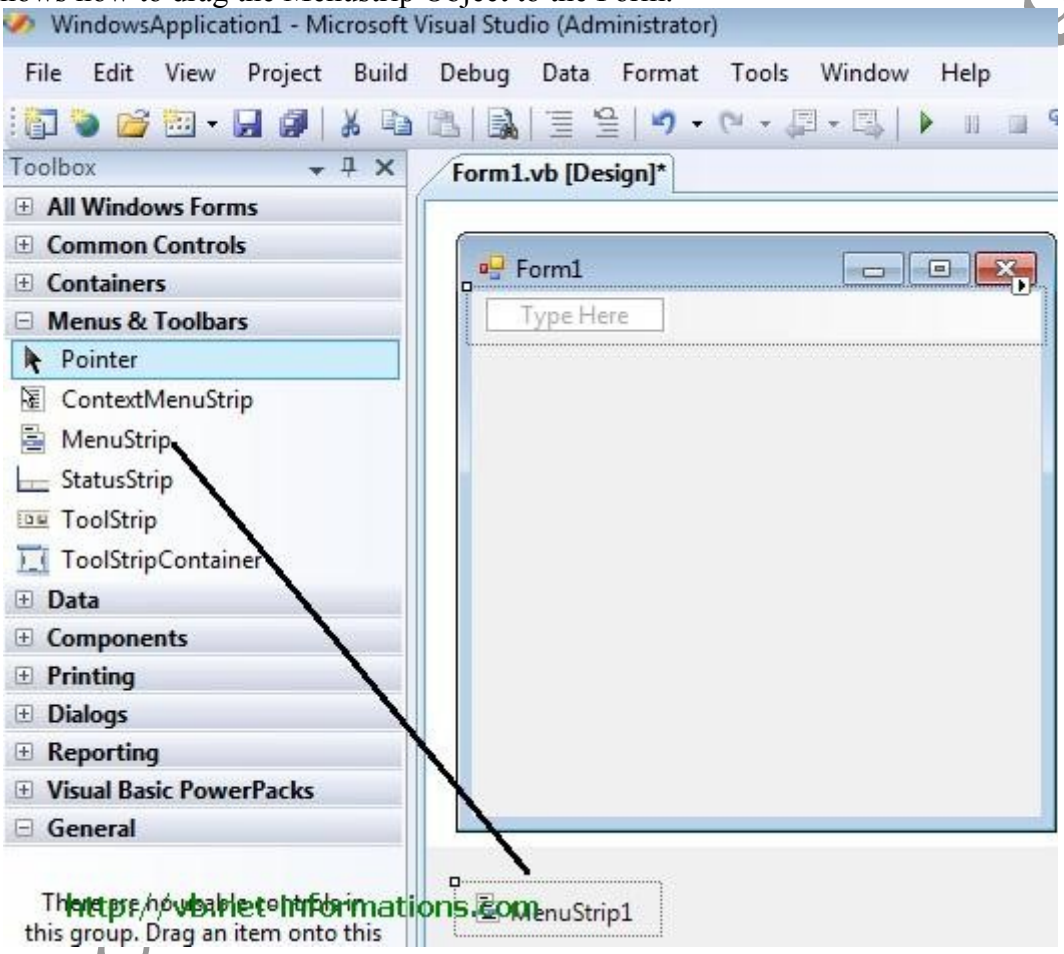
To start adding nodes, you should click the Add Root button, which adds a top-level node. To add child nodes to that node, you should select that node and use the Add Child button. To set text for a node, select the node and set it's text in the textbox as shown in the image above.

Assuming you added some nodes to the tree view, drag two Labels (Label1, Label2) from the toolbox on to the form. The following code displays the node you select on Label2 and the path to that node on Label1. The code looks like this:

Menu Control

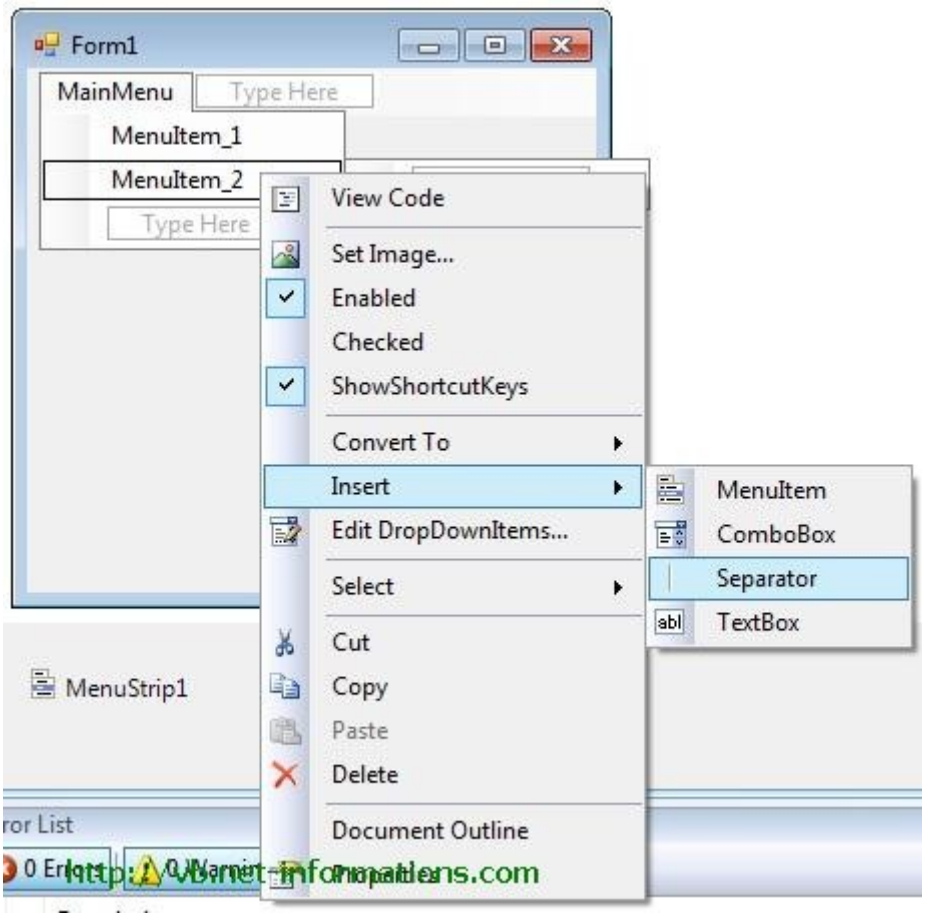
A menu is located on the menu bar and contains a list of related commands. MainMenu is the container for the Menu structure of the form and menus are made of MenuItem objects that represent individual parts of a menu.

You can create a main menu object on your form using the MainMenu control. The following picture shows how to drag the Menustrip Object to the Form.



After drag the Menustrip on your form you can directly create the menu items by type a value into the "Type Here" box on the menubar part of your form. From the following picture you can understand how to create each menu items on mainmenu Object.

If you need a separator bar , right click on your menu then go to insert->Separator.



After creating the Menu on the form , you have to double click on each menu item and write the programs there depends on your requirements. The following Vb.Net program shows how to show a messagebox when clicking a menu item.

Error Handling

The error handling construct in Visual Studio .NET is known as structured exception handling. The constructs used may be new to Visual Basic users, but should be familiar to users of C++ or Java.

Structured exception handling is straightforward to implement, and the same concepts are applicable to either VB.NET or C#. Throughout this section, example code will be shown in both languages.

VB .NET allows backward compatibility by also providing unstructured exception handling, via the familiar On Error GoTo statement and Err object, although this model is not discussed in this section.

Exceptions

Exceptions are used to handle error conditions in Visual Studio .NET. They provide information about the error condition.

An exception is an instance of a class which inherits from the System.Exception base class. Many different types of exception class are provided by the .NET Framework, and it is also possible to create your own exception classes. Each type extends the basic functionality of the System.Exception class by allowing further access to information about the specific type of error that has occurred.

An instance of an Exception class is created and thrown when the .NET Framework encounters an error condition. You can deal with exceptions by using the Try, Catch Finally construct.

Try, Catch, Finally

This construct allows you to catch errors that are thrown within your code. An example of this construct is shown below. An attempt is made to rotate an envelope, which throws an error.

```
Try
   Dim env As IEnvelope = New EnvelopeClass()
   env.PutCoords(0D, 0D, 10D, 10D)
   Dim trans As ITransform2D = env
   trans.Rotate(env.LowerLeft, 1D)
Catch ex As System.Exception
   MessageBox.Show("Error: " + ex.Message)
Finally
   ' Perform any tidy up code.
End Try
```

The Try block is placed around the code which may fail. If an error is thrown within the Try block, the point of execution will switch to the first Catch block.

The Catch block handles a thrown error. A Catch block is executed when the Type of a thrown error matches the Type of error specified by the Catch block. You can have more than one Catch block to handle different kinds of errors. The code shown below checks first if the exception thrown is a DivideByZeroException.

Try, Catch and Finally keywords.

The Try, Catch and Finally keywords are usually refered to as Error Handling. Handling exceptions with these methods will stop your application from recieving runtime errors (errors encountered

during running the application) caused by exceptions.

When you handle these errors, VB.NET gives you the ability to get a wealth of information about the exception and ways to let your application correspond according to them. Before we get into getting exception information, lets talk the complete basics.

- Try - The Try keyword will tell the compiler that we want to handle an area for errors. The Try keyword must have an Catch method, and must be encapsulated in an End Try statement.
- Catch - The Catch keyword tells the application what to do if an error occurs. It will stop the application from crashing, and let the application do the rest.
- Finally - The Finally keyword will occur regardless if the Try method worked successfully or a Catch method was used. The Finally keyword is optional.

Here is an example of a basic Try..Catch method:

```
Try
     ' Create a new integer variable.
     Dim anInteger As Integer = 0
     'Divide zero by zero to produce a DivideByZeroException
     'exception.
     anInteger \= 0
Catch ex As Exception
     ' Show an error telling the user an error occured.
     MessageBox.Show("An error occured")
Finally
     ' Tell the user that it has got to the Finally statement.
     MessageBox.Show("This has finally happened. :)")
End Try
```