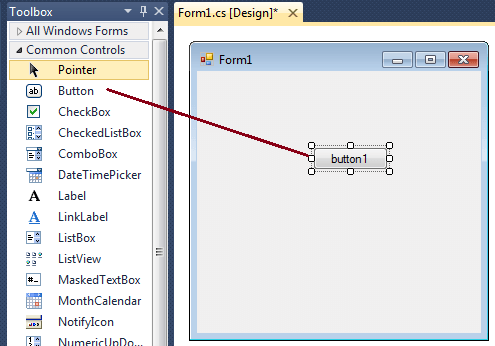
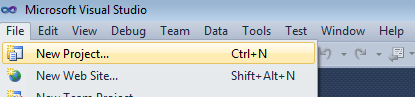
# C# Windows Forms

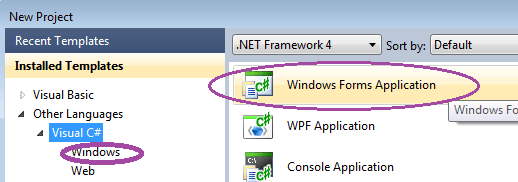
C# programmers have made extensive use of forms to build user interfaces. Each time you create a Windows application, Visual Studio will display a default blank form, onto which you can drag the controls onto your applications main form and adjust their size and position.



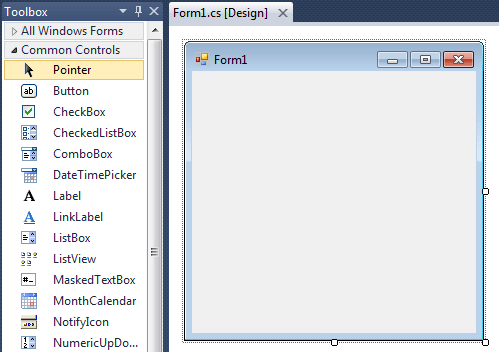
The first step is to start a new project and build a form. Open your Visual Studio and select File->New Project and from the new project dialog box select Other Languages->Visual C# and select Windows Forms Application. Enter a project name at the bottom of the dialouge box and click OK button. The following picture shows how to create a new Form in Visual Studio.



Select Windows Forms Application from New Project dialog box.

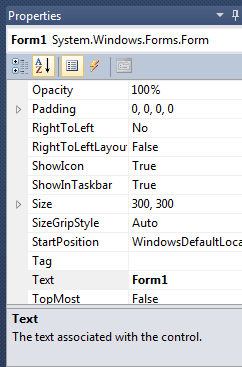


After selecting Windows Forms Application , you can see a default Form (Form1) in your new C# project. The Windows Form you see in Designer view is a visual representation of the window that will open when your application is opened. You can switch between this view and Code view at any time by right-clicking the design surface or code window and then clicking View Code or View Designer. The following picture shows how is the default Form (Form1) looks like.



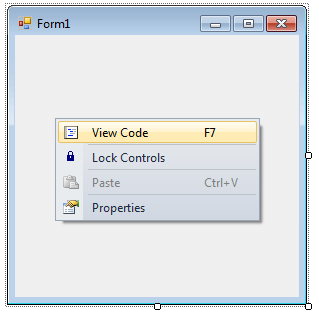
At the top of the form there is a title bar which displays the forms title. Form1 is the default name, and you can change the name to your convenience . The title bar also includes the control box, which holds the minimize, maximize, and close buttons.

If you want to set any properties of the Form, you can use Visual Studio Property window to change it. If you do not see the Properties window, on the View menu, click Properties window. This window lists the properties of the currently selected Windows Form or control, and its here that you can change the existing values.



For example , to change the forms title from Form1 to MyForm, click on Form1 and move to the right side down Properties window, set Text property to MyForm. Then you can see the Title of the form is changed. Likewise you can set any properties of Form through Properties window.

You can also set the properties of the Form1 through coding. For coding, you should right-click the design surface or code window and then clicking View Code.



When you right click on Form then you will get code behind window, there you can write your code

For example , if you want to change the back color of the form to Brown , you can code in the Form1\_Load event like the following

private void Form1\_Load(object sender, EventArgs e)

{

this.BackColor = Color.Brown;

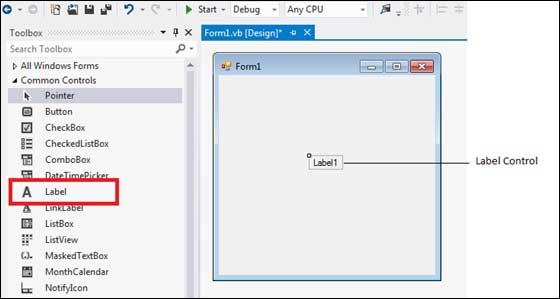
}

Likewise you can change other properties of Form1 through coding.

# Label control:

The Label control represents a standard Windows label. It is generally used to display some informative text on the GUI which is not changed during runtime.

Let's create a label by dragging a Label control from the Toolbox and dropping it on the form.



**Properties of the Label Control**

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

|  |  |
| --- | --- |
| **Sr.No.** | **Property & Description** |
| 1 | **Autosize**  Gets or sets a value specifying if the control should be automatically resized to display all its contents. |
| 2 | **BorderStyle**  Gets or sets the border style for the control. |
| 3 | **FlatStyle**  Gets or sets the flat style appearance of the Label control |
| 4 | **Font**  Gets or sets the font of the text displayed by the control. |
| 5 | **FontHeight**  Gets or sets the height of the font of the control. |
| 6 | **ForeColor**  Gets or sets the foreground color of the control. |
| 7 | **PreferredHeight**  Gets the preferred height of the control. |
| 8 | **PreferredWidth**  Gets the preferred width of the control. |
| 9 | **TabStop**  Gets or sets a value indicating whether the user can tab to the Label. This property is not used by this class. |
| 10 | **Text**  Gets or sets the text associated with this control. |
| 11 | **TextAlign**  Gets or sets the alignment of text in the label. |

**Methods of the Label Control**

The following are some of the commonly used methods of the Label control –

|  |  |
| --- | --- |
| **Sr.No.** | **Method Name & Description** |
| 1 | **GetPreferredSize**  Retrieves the size of a rectangular area into which a control can be fitted. |
| 2 | **Refresh**  Forces the control to invalidate its client area and immediately redraw itself and any child controls. |
| 3 | **Select**  Activates the control. |
| 4 | **Show**  Displays the control to the user. |
| 5 | **ToString**  Returns a String that contains the name of the control. |

**Events of the Label Control**

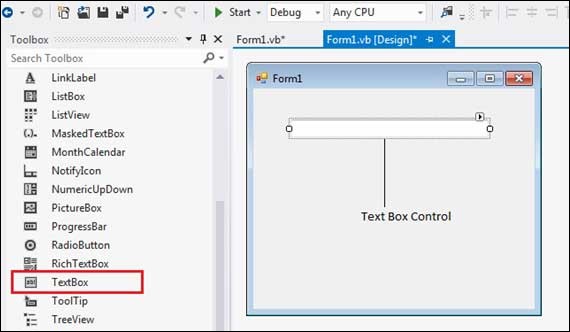
The following are some of the commonly used events of the Label control –

|  |  |
| --- | --- |
| **Sr.No.** | **Event & Description** |
| 1 | **AutoSizeChanged**  Occurs when the value of the AutoSize property changes. |
| 2 | **Click**  Occurs when the control is clicked. |
| 3 | **DoubleClick**  Occurs when the control is double-clicked. |
| 4 | **GotFocus**  Occurs when the control receives focus. |
| 5 | **Leave**  Occurs when the input focus leaves the control. |
| 6 | **LostFocus**  Occurs when the control loses focus. |
| 7 | **TabIndexChanged**  Occurs when the TabIndex property value changes. |
| 8 | **TabStopChanged**  Occurs when the TabStop property changes. |
| 9 | **TextChanged**  Occurs when the Text property value changes. |

# TextBox Control:

Text box controls allow entering text on a form at runtime. By default, it takes a single line of text, however, you can make it accept multiple texts and even add scroll bars to it.

Let's create a text box by dragging a Text Box control from the Toolbox and dropping it on the form.



**The Properties of the TextBox Control**

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

|  |  |
| --- | --- |
| **Sr.No.** | **Property & Description** |
| 1 | **AcceptsReturn**  Gets or sets a value indicating whether pressing ENTER in a multiline TextBox control creates a new line of text in the control or activates the default button for the form. |
| 2 | **AutoCompleteCustomSource**  Gets or sets a custom System.Collections.Specialized.StringCollection to use when the AutoCompleteSourceproperty is set to CustomSource. |
| 3 | **AutoCompleteMode**  Gets or sets an option that controls how automatic completion works for the TextBox. |
| 4 | **AutoCompleteSource**  Gets or sets a value specifying the source of complete strings used for automatic completion. |
| 5 | **CharacterCasing**  Gets or sets whether the TextBox control modifies the case of characters as they are typed. |
| 6 | **Font**  Gets or sets the font of the text displayed by the control. |
| 7 | **FontHeight**  Gets or sets the height of the font of the control. |
| 8 | **ForeColor**  Gets or sets the foreground color of the control. |
| 9 | **Lines**  Gets or sets the lines of text in a text box control. |
| 10 | **Multiline**  Gets or sets a value indicating whether this is a multiline TextBox control. |
| 11 | **PasswordChar**  Gets or sets the character used to mask characters of a password in a single-line TextBox control. |
| 12 | **ReadOnly**  Gets or sets a value indicating whether text in the text box is read-only. |
| 13 | **ScrollBars**  Gets or sets which scroll bars should appear in a multiline TextBox control. This property has values −   * None * Horizontal * Vertical * Both |
| 14 | **TabIndex**  Gets or sets the tab order of the control within its container. |
| 15 | **Text**  Gets or sets the current text in the TextBox. |
| 16 | **TextAlign**  Gets or sets how text is aligned in a TextBox control. This property has values −   * Left * Right * Center |
| 17 | **TextLength**  Gets the length of text in the control. |
| 18 | **WordWrap**  Indicates whether a multiline text box control automatically wraps words to the beginning of the next line when necessary. |

### The Methods of the TextBox Control

The following are some of the commonly used methods of the TextBox control –

|  |  |
| --- | --- |
| **Sr.No.** | **Method Name & Description** |
| 1 | **AppendText**  Appends text to the current text of a text box. |
| 2 | **Clear**  Clears all text from the text box control. |
| 3 | **Copy**  Copies the current selection in the text box to the **Clipboard**. |
| 4 | **Cut**  Moves the current selection in the text box to the **Clipboard**. |
| 5 | **Paste**  Replaces the current selection in the text box with the contents of the **Clipboard**. |
| 6 | **Paste(String)**  Sets the selected text to the specified text without clearing the undo buffer. |
| 7 | **ResetText**  Resets the Text property to its default value. |
| 8 | **ToString**  Returns a string that represents the TextBoxBase control. |
| 9 | **Undo**  Undoes the last edit operation in the text box. |

**Events of the TextBox Control**

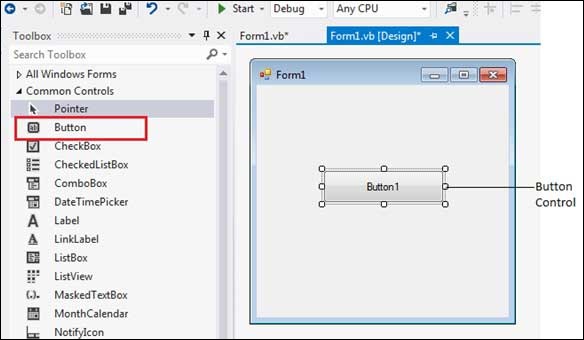
The following are some of the commonly used events of the Text control –

|  |  |
| --- | --- |
| **Sr.No.** | **Event & Description** |
| 1 | **Click**  Occurs when the control is clicked. |
| 2 | **DoubleClick**  Occurs when the control is double-clicked. |
| 3 | **TextAlignChanged**  Occurs when the TextAlign property value changes. |

### Button Controls

The Button control represents a standard Windows button. It is generally used to generate a Click event by providing a handler for the Click event.

Let's create a label by dragging a Button control from the Toolbox ad dropping it on the form.



**Properties of the Button Control**

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

|  |  |
| --- | --- |
| **Sr.No.** | **Property & Description** |
| 1 | **AutoSizeMode**  Gets or sets the mode by which the Button automatically resizes itself. |
| 2 | **BackColor**  Gets or sets the background color of the control. |
| 3 | **BackgroundImage**  Gets or sets the background image displayed in the control. |
| 4 | **DialogResult**  Gets or sets a value that is returned to the parent form when the button is clicked. This is used while creating dialog boxes. |
| 5 | **ForeColor**  Gets or sets the foreground color of the control. |
| 6 | **Image**  Gets or sets the image that is displayed on a button control. |
| 7 | **Location**  Gets or sets the coordinates of the upper-left corner of the control relative to the upper-left corner of its container. |
| 8 | **TabIndex**  Gets or sets the tab order of the control within its container. |
| 9 | **Text**  Gets or sets the text associated with this control. |

**Methods of the Button Control**

The following are some of the commonly used methods of the Button control –

|  |  |
| --- | --- |
| **Sr.No.** | **Method Name & Description** |
| 1 | **GetPreferredSize**  Retrieves the size of a rectangular area into which a control can be fitted. |
| 2 | **NotifyDefault**  Notifies the Button whether it is the default button so that it can adjust its appearance accordingly. |
| 3 | **Select**  Activates the control. |
| 4 | **ToString**  Returns a String containing the name of the Component, if any. This method should not be overridden. |

**Events of the Button Control**

The following are some of the commonly used events of the Button control –

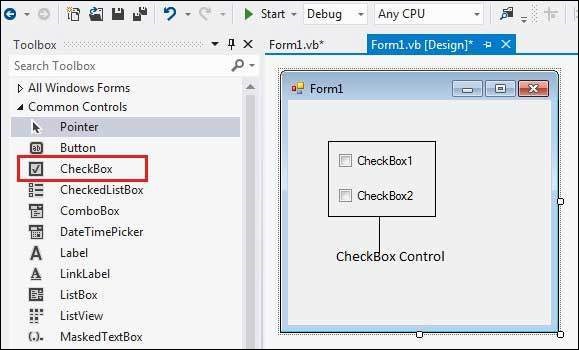
|  |  |
| --- | --- |
| **Sr.No.** | **Event & Description** |
| 1 | **Click**  Occurs when the control is clicked. |
| 2 | **DoubleClick**  Occurs when the user double-clicks the Button control. |
| 3 | **GotFocus**  Occurs when the control receives focus. |
| 4 | **TabIndexChanged**  Occurs when the TabIndex property value changes. |
| 5 | **TextChanged**  Occurs when the Text property value changes. |
| 6 | **Validated**  Occurs when the control is finished validating. |

**GroupBox and Panel Controls**

**CheckBox control:**

The CheckBox control allows the user to set true/false or yes/no type options. The user can select or deselect it. When a check box is selected it has the value True, and when it is cleared, it holds the value False.

Let's create two check boxes by dragging CheckBox controls from the Toolbox and dropping on the form.



The CheckBox control has three states, checked, unchecked and indeterminate. In the indeterminate state, the check box is grayed out. To enable the indeterminate state, the *ThreeState* property of the check box is set to be True.

## Properties of the CheckBox Control

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

|  |  |
| --- | --- |
| **Sr.No.** | **Property & Description** |
| 1 | **Appearance**  Gets or sets a value determining the appearance of the check box. |
| 2 | **AutoCheck**  Gets or sets a value indicating whether the Checked or CheckedState value and the appearance of the control automatically change when the check box is selected. |
| 3 | **CheckAlign**  Gets or sets the horizontal and vertical alignment of the check mark on the check box. |
| 4 | **Checked**  Gets or sets a value indicating whether the check box is selected. |
| 5 | **CheckState**  Gets or sets the state of a check box. |
| 6 | **Text**  Gets or sets the caption of a check box. |
| 7 | **ThreeState**  Gets or sets a value indicating whether or not a check box should allow three check states rather than two. |

## Methods of the CheckBox Control

The following are some of the commonly used methods of the CheckBox control –

|  |  |
| --- | --- |
| **Sr.No.** | **Method Name & Description** |
| 1 | **OnCheckedChanged**  Raises the CheckedChanged event. |
| 2 | **OnCheckStateChanged**  Raises the CheckStateChanged event. |
| 3 | **OnClick**  Raises the OnClick event. |

## Events of the CheckBox Control

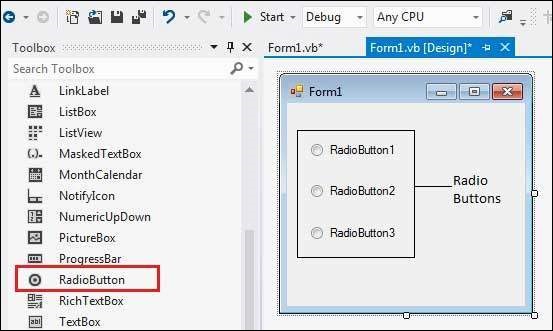
The following are some of the commonly used events of the CheckBox control –

|  |  |
| --- | --- |
| Sr.No. | Event & Description |
| 1 | **AppearanceChanged**  Occurs when the value of the Appearance property of the check box is changed. |
| 2 | **CheckedChanged**  Occurs when the value of the Checked property of the CheckBox control is changed. |
| 3 | **CheckStateChanged**  Occurs when the value of the CheckState property of the CheckBox control is changed. |

**RadioButton:**

The RadioButton control is used to provide a set of mutually exclusive options. The user can select one radio button in a group. If you need to place more than one group of radio buttons in the same form, you should place them in different container controls like a GroupBox control.

Let's create three radio buttons by dragging RadioButton controls from the Toolbox and dropping on the form.



The *Checked* property of the radio button is used to set the state of a radio button. You can display text, image or both on radio button control. You can also change the appearance of the radio button control by using the *Appearance* property.

**Properties of the RadioButton Control**

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

|  |  |
| --- | --- |
| **Sr.No.** | **Property & Description** |
| 1 | **Appearance**  Gets or sets a value determining the appearance of the radio button. |
| 2 | **AutoCheck**  Gets or sets a value indicating whether the Checked value and the appearance of the control automatically change when the control is clicked. |
| 3 | **CheckAlign**  Gets or sets the location of the check box portion of the radio button. |
| 4 | **Checked**  Gets or sets a value indicating whether the control is checked. |
| 5 | **Text**  Gets or sets the caption for a radio button. |
| 6 | **TabStop**  Gets or sets a value indicating whether a user can give focus to the RadioButton control using the TAB key. |

**Methods of the RadioButton Control**

The following are some of the commonly used methods of the RadioButton control –

|  |  |
| --- | --- |
| **Sr.No.** | **Method Name & Description** |
| 1 | **PerformClick**  Generates a Click event for the control, simulating a click by a user. |

**Events of the RadioButton Control**

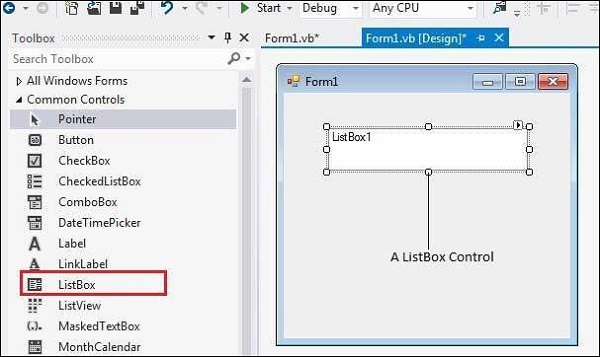
The following are some of the commonly used events of the RadioButton control –

|  |  |
| --- | --- |
| **Sr.No** | **Event & Description** |
| 1 | **AppearanceChanged**  Occurs when the value of the Appearance property of the RadioButton control is changed. |
| 2 | **CheckedChanged**  Occurs when the value of the Checked property of the RadioButton control is changed. |

**ListBox:**

The ListBox represents a Windows control to display a list of items to a user. A user can select an item from the list. It allows the programmer to add items at design time by using the properties window or at the runtime.

Let's create a list box by dragging a ListBox control from the Toolbox and dropping it on the form.



You can populate the list box items either from the properties window or at runtime. To add items to a ListBox, select the ListBox control and get to the properties window, for the properties of this control. Click the ellipses (...) button next to the Items property. This opens the String Collection Editor dialog box, where you can enter the values one at a line.

**Properties of the ListBox Control**

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

|  |  |
| --- | --- |
| **Sr.No.** | **Property & Description** |
| 1 | **AllowSelection**  Gets a value indicating whether the ListBox currently enables selection of list items. |
| 2 | **BorderStyle**  Gets or sets the type of border drawn around the list box. |
| 3 | **ColumnWidth**  Gets of sets the width of columns in a multicolumn list box. |
| 4 | **HorizontalExtent**  Gets or sets the horizontal scrolling area of a list box. |
| 5 | **HorizontalScrollBar**  Gets or sets the value indicating whether a horizontal scrollbar is displayed in the list box. |
| 6 | **ItemHeight**  Gets or sets the height of an item in the list box. |
| 7 | **Items**  Gets the items of the list box. |
| 8 | **MultiColumn**  Gets or sets a value indicating whether the list box supports multiple columns. |
| 9 | **ScrollAlwaysVisible**  Gets or sets a value indicating whether the vertical scroll bar is shown at all times. |
| 10 | **SelectedIndex**  Gets or sets the zero-based index of the currently selected item in a list box. |
| 11 | **SelectedIndices**  Gets a collection that contains the zero-based indexes of all currently selected items in the list box. |
| 12 | **SelectedItem**  Gets or sets the currently selected item in the list box. |
| 13 | **SelectedItems**  Gets a collection containing the currently selected items in the list box. |
| 14 | **SelectedValue**  Gets or sets the value of the member property specified by the ValueMember property. |
| 15 | **SelectionMode**  Gets or sets the method in which items are selected in the list box. This property has values −   * None * One * MultiSimple * MultiExtended |
| 16 | **Sorted**  Gets or sets a value indicating whether the items in the list box are sorted alphabetically. |
| 17 | **Text**  Gets or searches for the text of the currently selected item in the list box. |
| 18 | **TopIndex**  Gets or sets the index of the first visible item of a list box. |

**Methods of the ListBox Control**

The following are some of the commonly used methods of the ListBox control –

|  |  |
| --- | --- |
| **Sr.No.** | **Method Name & Description** |
| 1 | **BeginUpdate**  Prevents the control from drawing until the EndUpdate method is called, while items are added to the ListBox one at a time. |
| 2 | **ClearSelected**  Unselects all items in the ListBox. |
| 3 | **EndUpdate**  Resumes drawing of a list box after it was turned off by the BeginUpdate method. |
| 4 | **FindString**  Finds the first item in the ListBox that starts with the string specified as an argument. |
| 5 | **FindStringExact**  Finds the first item in the ListBox that exactly matches the specified string. |
| 6 | **GetSelected**  Returns a value indicating whether the specified item is selected. |
| 7 | **SetSelected**  Selects or clears the selection for the specified item in a ListBox. |
| 8 | **OnSelectedIndexChanged**  Raises the SelectedIndexChanged event. |
| 8 | **OnSelectedValueChanged**  Raises the SelectedValueChanged event. |

**Events of the ListBox Control**

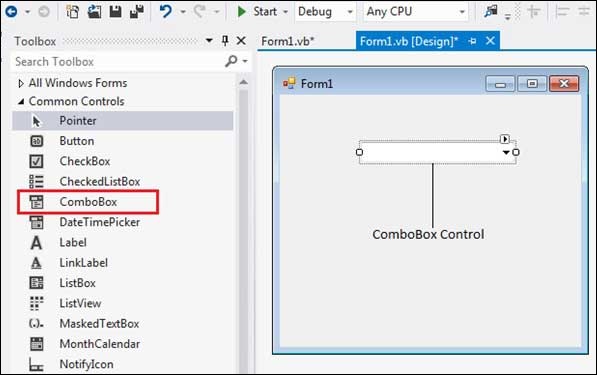
The following are some of the commonly used events of the ListBox control –

|  |  |
| --- | --- |
| **Sr.No.** | **Event & Description** |
| 1 | **Click**  Occurs when a list box is selected. |
| 2 | **SelectedIndexChanged**  Occurs when the SelectedIndex property of a list box is changed. |

**ComboBox Control**

The ComboBox control is used to display a drop-down list of various items. It is a combination of a text box in which the user enters an item and a drop-down list from which the user selects an item.

Let's create a combo box by dragging a ComboBox control from the Toolbox and dropping it on the form.



You can populate the list box items either from the properties window or at runtime. To add items to a ComboBox, select the ComboBox control and go to the properties window for the properties of this control. Click the ellipses (...) button next to the Items property. This opens the String Collection Editor dialog box, where you can enter the values one at a line.

**Properties of the ComboBox Control**

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

|  |  |
| --- | --- |
| **Sr.No.** | **Property & Description** |
| 1 | **AllowSelection**  Gets a value indicating whether the list enables selection of list items. |
| 2 | **AutoCompleteCustomSource**  Gets or sets a custom System.Collections .Specialized.StringCollection to use when the AutoCompleteSourceproperty is set to CustomSource. |
| 3 | **AutoCompleteMode**  Gets or sets an option that controls how automatic completion works for the ComboBox. |
| 4 | **AutoCompleteSource**  Gets or sets a value specifying the source of complete strings used for automatic completion. |
| 5 | **DataBindings**  Gets the data bindings for the control. |
| 6 | **DataManager**  Gets the CurrencyManager associated with this control. |
| 7 | **DataSource**  Gets or sets the data source for this ComboBox. |
| 8 | **DropDownHeight**  Gets or sets the height in pixels of the drop-down portion of the ComboBox. |
| 9 | **DropDownStyle**  Gets or sets a value specifying the style of the combo box. |
| 10 | **DropDownWidth**  Gets or sets the width of the of the drop-down portion of a combo box. |
| 11 | **DroppedDown**  Gets or sets a value indicating whether the combo box is displaying its drop-down portion. |
| 12 | **FlatStyle**  Gets or sets the appearance of the ComboBox. |
| 13 | **ItemHeight**  Gets or sets the height of an item in the combo box. |
| 14 | **Items**  Gets an object representing the collection of the items contained in this ComboBox. |
| 15 | **MaxDropDownItems**  Gets or sets the maximum number of items to be displayed in the drop-down part of the combo box. |
| 16 | **MaxLength**  Gets or sets the maximum number of characters a user can enter in the editable area of the combo box. |
| 17 | **SelectedIndex**  Gets or sets the index specifying the currently selected item. |
| 18 | **SelectedItem**  Gets or sets currently selected item in the ComboBox. |
| 19 | **SelectedText**  Gets or sets the text that is selected in the editable portion of a ComboBox. |
| 20 | **SelectedValue**  Gets or sets the value of the member property specified by the ValueMember property. |
| 21 | **SelectionLength**  Gets or sets the number of characters selected in the editable portion of the combo box. |
| 22 | **SelectionStart**  Gets or sets the starting index of text selected in the combo box. |
| 23 | **Sorted**  Gets or sets a value indicating whether the items in the combo box are sorted. |
| 24 | **Text**  Gets or sets the text associated with this control. |

**Methods of the ComboBox Control**

The following are some of the commonly used methods of the ComboBox control –

|  |  |
| --- | --- |
| **Sr.No.** | **Method Name & Description** |
| 1 | **BeginUpdate**  Prevents the control from drawing until the EndUpdate method is called, while items are added to the combo box one at a time. |
| 2 | **EndUpdate**  Resumes drawing of a combo box, after it was turned off by the BeginUpdate method. |
| 3 | **FindString**  Finds the first item in the combo box that starts with the string specified as an argument. |
| 4 | **FindStringExact**  Finds the first item in the combo box that exactly matches the specified string. |
| 5 | **SelectAll**  Selects all the text in the editable area of the combo box. |

**Events of the ComboBox Control**

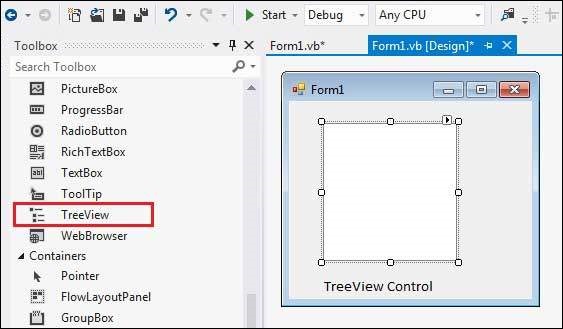
The following are some of the commonly used events of the ComboBox control –

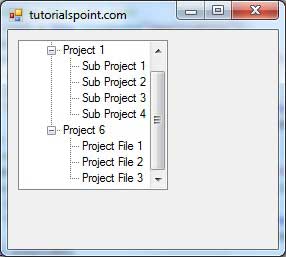
|  |  |
| --- | --- |
| **Sr.No.** | **Event & Description** |
| 1 | **DropDown**  Occurs when the drop-down portion of a combo box is displayed. |
| 2 | **DropDownClosed**  Occurs when the drop-down portion of a combo box is no longer visible. |
| 3 | **DropDownStyleChanged**  Occurs when the DropDownStyle property of the ComboBox has changed. |
| 4 | **SelectedIndexChanged**  Occurs when the SelectedIndex property of a ComboBox control has changed. |
| 5 | **SelectionChangeCommitted**  Occurs when the selected item has changed and the change appears in the combo box. |

**TreeView Control**

The TreeView control is used to display hierarchical representations of items similar to the ways the files and folders are displayed in the left pane of the Windows Explorer. Each node may contain one or more child nodes.

Let's click on a TreeView control from the Toolbox and place it on the form.





**Properties of the TreeView Control**

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

|  |  |
| --- | --- |
| **Sr.No.** | **Property & Description** |
| 1 | **BackColor**  Gets or sets the background color for the control. |
| 2 | **BackgroundImage**  Gets or set the background image for the TreeView control. |
| 3 | **BackgroundImageLayout**  Gets or sets the layout of the background image for the TreeView control. |
| 4 | **BorderStyle**  Gets or sets the border style of the tree view control. |
| 5 | **CheckBoxes**  Gets or sets a value indicating whether check boxes are displayed next to the tree nodes in the tree view control. |
| 6 | **DataBindings**  Gets the data bindings for the control. |
| 7 | **Font**  Gets or sets the font of the text displayed by the control. |
| 8 | **FontHeight**  Gets or sets the height of the font of the control. |
| 9 | **ForeColor**  The current foreground color for this control, which is the color the control uses to draw its text. |
| 10 | **ItemHeight**  Gets or sets the height of each tree node in the tree view control. |
| 11 | **Nodes**  Gets the collection of tree nodes that are assigned to the tree view control. |
| 12 | **PathSeparator**  Gets or sets the delimiter string that the tree node path uses. |
| 13 | **RightToLeftLayout**  Gets or sets a value that indicates whether the TreeView should be laid out from right-to-left. |
| 14 | **Scrollable**  Gets or sets a value indicating whether the tree view control displays scroll bars when they are needed. |
| 15 | **SelectedImageIndex**  Gets or sets the image list index value of the image that is displayed when a tree node is selected. |
| 16 | **SelectedImageKey**  Gets or sets the key of the default image shown when a TreeNode is in a selected state. |
| 17 | **SelectedNode**  Gets or sets the tree node that is currently selected in the tree view control. |
| 18 | **ShowLines**  Gets or sets a value indicating whether lines are drawn between tree nodes in the tree view control. |
| 19 | **ShowNodeToolTips**  Gets or sets a value indicating ToolTips are shown when the mouse pointer hovers over a TreeNode. |
| 20 | **ShowPlusMinus**  Gets or sets a value indicating whether plus-sign (+) and minus-sign (-) buttons are displayed next to tree nodes that contain child tree nodes. |
| 21 | **ShowRootLines**  Gets or sets a value indicating whether lines are drawn between the tree nodes that are at the root of the tree view. |
| 22 | **Sorted**  Gets or sets a value indicating whether the tree nodes in the tree view are sorted. |
| 23 | **StateImageList**  Gets or sets the image list that is used to indicate the state of the TreeView and its nodes. |
| 24 | **Text**  Gets or sets the text of the TreeView. |
| 25 | **TopNode**  Gets or sets the first fully-visible tree node in the tree view control. |
| 26 | **TreeViewNodeSorter**  Gets or sets the implementation of IComparer to perform a custom sort of the TreeView nodes. |
| 27 | **VisibleCount**  Gets the number of tree nodes that can be fully visible in the tree view control. |

**Methods of the TreeView Control**

The following are some of the commonly used methods of the TreeView control –

|  |  |
| --- | --- |
| **Sr.No.** | **Method Name & Description** |
| 1 | **CollapseAll**  Collapses all the nodes including all child nodes in the tree view control. |
| 2 | **ExpandAll**  Expands all the nodes. |
| 3 | **GetNodeAt**  Gets the node at the specified location. |
| 4 | **GetNodeCount**  Gets the number of tree nodes. |
| 5 | **Sort**  Sorts all the items in the tree view control. |
| 6 | **ToString**  Returns a string containing the name of the control. |

**Events of the TreeView Control**

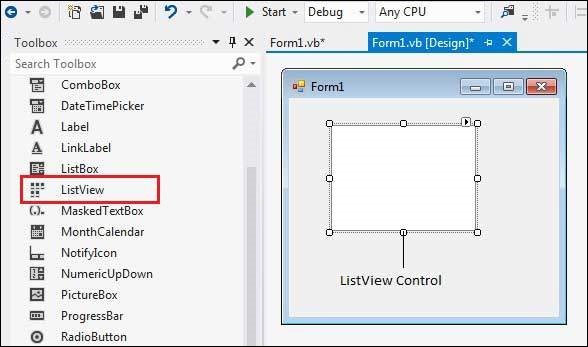
The following are some of the commonly used events of the TreeView control –

|  |  |
| --- | --- |
| **Sr.No.** | **Event & Description** |
| 1 | **AfterCheck**  Occurs after the tree node check box is checked. |
| 2 | **AfterCollapse**  Occurs after the tree node is collapsed. |
| 3 | **AfterExpand**  Occurs after the tree node is expanded. |
| 4 | **AfterSelect**  Occurs after the tree node is selected. |
| 5 | **BeforeCheck**  Occurs before the tree node check box is checked. |
| 6 | **BeforeCollapse**  Occurs before the tree node is collapsed. |
| 7 | **BeforeExpand**  Occurs before the tree node is expanded. |
| 8 | **BeforeLabelEdit**  Occurs before the tree node label text is edited. |
| 9 | **BeforeSelect**  Occurs before the tree node is selected. |
| 10 | **ItemDrag**  Occurs when the user begins dragging a node. |
| 11 | **NodeMouseClick**  Occurs when the user clicks a TreeNode with the mouse. |
| 12 | **NodeMouseDoubleClick**  Occurs when the user double-clicks a TreeNode with the mouse. |
| 13 | **NodeMouseHover**  Occurs when the mouse hovers over a TreeNode. |
| 14 | **PaddingChanged**  Occurs when the value of the Padding property changes. |
| 15 | **Paint**  Occurs when the TreeView is drawn. |
| 16 | **RightToLeftLayoutChanged**  Occurs when the value of the RightToLeftLayout property changes. |
| 17 | **TextChanged**  Occurs when the Text property changes. |

**ListView Control:**

The ListView control is used to display a list of items. Along with the TreeView control, it allows you to create a Windows Explorer like interface.

Let's click on a ListView control from the Toolbox and place it on the form.



The *ListView* control displays a list of items along with icons. The Item property of the ListView control allows you to add and remove items from it. The *SelectedItem* property contains a collection of the selected items. The *MultiSelect* property allows you to set select more than one item in the list view. The *CheckBoxes* property allows you to set check boxes next to the items.

**Properties of the ListView Control**

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

|  |  |
| --- | --- |
| **Sr.No** | **Property & Description** |
| 1 | **Alignment**  Gets or sets the alignment of items in the control. |
| 2 | **AutoArrange**  Gets or sets whether icons are automatically kept arranged. |
| 3 | **BackColor**  Gets or sets the background color. |
| 4 | **CheckBoxes**  Gets or sets a value indicating whether a check box appears next to each item in the control. |
| 5 | **CheckedIndices**  Gets the indexes of the currently checked items in the control. |
| 6 | **CheckedItems**  Gets the currently checked items in the control. |
| 7 | **Columns**  Gets the collection of all column headers that appear in the control. |
| 8 | **GridLines**  Gets or sets a value indicating whether grid lines appear between the rows and columns containing the items and subitems in the control. |
| 9 | **HeaderStyle**  Gets or sets the column header style. |
| 10 | **HideSelection**  Gets or sets a value indicating whether the selected item in the control remains highlighted when the control loses focus. |
| 11 | **HotTracking**  Gets or sets a value indicating whether the text of an item or subitem has the appearance of a hyperlink when the mouse pointer passes over it. |
| 12 | **HoverSelection**  Gets or sets a value indicating whether an item is automatically selected when the mouse pointer remains over the item for a few seconds. |
| 13 | **InsertionMark**  Gets an object used to indicate the expected drop location when an item is dragged within a ListView control. |
| 14 | **Items**  Gets a collection containing all items in the control. |
| 15 | **LabelWrap**  Gets or sets a value indicating whether item labels wrap when items are displayed in the control as icons. |
| 16 | **LargeImageList**  Gets or sets the ImageList to use when displaying items as large icons in the control. |
| 17 | **MultiSelect**  Gets or sets a value indicating whether multiple items can be selected. |
| 18 | **RightToLeftLayout**  Gets or sets a value indicating whether the control is laid out from right to left. |
| 19 | **Scrollable**  Gets or sets a value indicating whether a scroll bar is added to the control when there is not enough room to display all items. |
| 20 | **SelectedIndices**  Gets the indexes of the selected items in the control. |
| 21 | **SelectedItems**  Gets the items that are selected in the control. |
| 22 | **ShowGroups**  Gets or sets a value indicating whether items are displayed in groups. |
| 23 | **ShowItemToolTips**  Gets or sets a value indicating whether ToolTips are shown for the ListViewItem objects contained in theListView. |
| 24 | **SmallImageList**  Gets or sets the ImageList to use when displaying items as small icons in the control. |
| 25 | **Sorting**  Gets or sets the sort order for items in the control. |
| 26 | **StateImageList**  Gets or sets the ImageList associated with application-defined states in the control. |
| 27 | **TopItem**  Gets or sets the first visible item in the control. |
| 28 | **View**  Gets or sets how items are displayed in the control. This property has the following values:   * LargeIcon − displays large items with a large 32 x 32 pixels icon. * SmallIcon − displays items with a small 16 x 16 pixels icon * List − displays small icons always in one column * Details − displays items in multiple columns with column headers and fields * Tile − displays items as full-size icons with item labels and sub-item information. |
| 29 | **VirtualListSize**  Gets or sets the number of ListViewItem objects contained in the list when in virtual mode. |
| 30 | **VirtualMode**  Gets or sets a value indicating whether you have provided your own data-management operations for the ListView control. |

**Methods of the ListView Control**

The following are some of the commonly used methods of the ListView control –

|  |  |
| --- | --- |
| **Sr.No.** | **Method Name & Description** |
| 1 | **Clear**  Removes all items from the ListView control. |
| 1 | **ToString**  Returns a string containing the string representation of the control. |

**Events of the ListView Control**

The following are some of the commonly used events of the ListView control –

|  |  |
| --- | --- |
| **Sr.No.** | **Event & Description** |
| 1 | **ColumnClick**  Occurs when a column header is clicked. |
| 2 | **ItemCheck**  Occurs when an item in the control is checked or unchecked. |
| 3 | **SelectedIndexChanged**  Occurs when the selected index is changed. |
| 4 | **TextChanged**  Occurs when the Text property is changed. |

**Mouse Handling Events**

Events are basically a user action like key press, clicks, mouse movements, etc., or some occurrence like system generated notifications. Applications need to respond to events when they occur.

Clicking on a button, or entering some text in a text box, or clicking on a menu item, all are examples of events. An event is an action that calls a function or may cause another event. Event handlers are functions that tell how to respond to an event.

C#.Net is an event-driven language. There are mainly two types of events −

* Mouse events
* Keyboard events

**Handling Mouse Events**

Mouse events occur with mouse movements in forms and controls. Following are the various mouse events related with a Control class −

* **MouseDown** − it occurs when a mouse button is pressed
* **MouseEnter** − it occurs when the mouse pointer enters the control
* **MouseHover** − it occurs when the mouse pointer hovers over the control
* **MouseLeave** − it occurs when the mouse pointer leaves the control
* **MouseMove** − it occurs when the mouse pointer moves over the control
* **MouseUp** − it occurs when the mouse pointer is over the control and the mouse button is released
* **MouseWheel** − it occurs when the mouse wheel moves and the control has focus

The event handlers of the mouse events get an argument of type **MouseEventArgs**. The MouseEventArgs object is used for handling mouse events. It has the following properties −

* **Buttons** − indicates the mouse button pressed
* **Clicks** − indicates the number of clicks
* **Delta** − indicates the number of detents the mouse wheel rotated
* **X** − indicates the x-coordinate of mouse click
* **Y** − indicates the y-coordinate of mouse click

**Handling Keyboard Events**

Following are the various keyboard events related with a Control class −

* **KeyDown** − occurs when a key is pressed down and the control has focus
* **KeyPress** − occurs when a key is pressed and the control has focus
* **KeyUp** − occurs when a key is released while the control has focus

The event handlers of the KeyDown and KeyUp events get an argument of type **KeyEventArgs**. This object has the following properties −

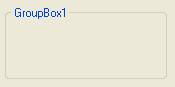
* **Alt** − it indicates whether the ALT key is pressed
* **Control** − it indicates whether the CTRL key is pressed
* **Handled** − it indicates whether the event is handled
* **KeyCode** − stores the keyboard code for the event
* **KeyData** − stores the keyboard data for the event
* **KeyValue** − stores the keyboard value for the event
* **Modifiers** − it indicates which modifier keys (Ctrl, Shift, and/or Alt) are pressed
* **Shift** − it indicates if the Shift key is pressed

The event handlers of the KeyDown and KeyUp events get an argument of type **KeyEventArgs**. This object has the following properties −

* **Handled** − indicates if the KeyPress event is handled
* **KeyChar** − stores the character corresponding to the key pressed

**GroupBox :**

GroupBox control is used to group other controls of VB.NET.  
GroupBox control having a frame to indicate boundary and a text to indicate header or title.  
Generally GroupBox control is used as a container for Radio Button. When Radio Buttons are grouped using GroupBox, user can select one RadioButton from each GroupBox.



**Properties of GroupBox Control**

|  |  |
| --- | --- |
| Property | Purpose |
| BackColor | It is used to get or set background color of the GroupBox. |
| BackgroundImage | It is used to get or set background Image of the GroupBox. |
| BackgroundImageLayout | It is used to get or set background Image layout of the GroupBox. It has one of the following values: None, Tile, Centre, Stretch, Zoom |
| Font | It is used to get or set font Style, Font Size, Font Face of the text contained in GroupBox Control. |
| ForeColor | It is used to get or set color of the text contained in GroupBox Control. |
| Enabled | It is used to specify weather GroupBox Control is enabled or not. It has Boolean value. Default value is true. |
| Visible | It is used to specify weather GroupBox Control is visible or not at run time. It has Boolean value. Default value is true. |
| Text | It is used to get or set Title or Header Text of the GroupBox Control. |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

**Methods of GroupBox Control**

|  |  |
| --- | --- |
| Method | Purpose |
| Show | It is used to show GroupBox Control. |
| Hide | It is used to Hide GroupBox Control at run time. |
| Focus | It is used to set cursor or focus on GroupBox Control. |

**ToolTip:**

**ToolTip** Control is used to display brief description of the control whenever focus is set on that control. Brief description contains information such as for which purpose this control is used or which type of input should be accepted by that control.

**Properties of ToolTip Control**

Various Properties of ToolTip Control are:

|  |  |
| --- | --- |
| Property Name | Description |
| **Name** | It is used to specify name of the ToolTip Control. |
| **Active** | It is used to determine weather ToolTip Control will be Active or Not. It has boolean value. Default value is true. ToolTip will appear only when this property is set to true. |
| **InitialDelay** | It is used to specify amount of time the mouse pointer should remain on the control to display ToolTip text. |
| **AutoPopDelay** | It is used to specify amount of time the ToolTip text remains visible if mouse pointer remains on the control. |
| **IsBallon** | It is used to determine weather ToolTip will be displayed in the form of ballon or not. It has boolean value. Default value is false. |
| **ToolTipIcon** | It is used to set icon to be displayed in the ToolTip. It can have following values: **None,Info,Error,Warning**. Default value is None. |
| **ToolTipTitle** | It is used to set title to be displayed in each ToolTip. |
| **UseAnimation** | It is used to determine weather animations are used when ToolTip is display and Hide. It has boolean value.Default value is true. |
| **ShowAlways** | It is used to determine weather ToolTip will be displayed always even if the parent window is not active. It has boolean value. Default value is false. |
| **StripAmpersands** | It is used to determine weather ampersand (&) will be displayed in the ToolTip text or not. It has boolean value. Default value is false. If it is set to true then ampersand (&) will not display in the ToolTip text. |

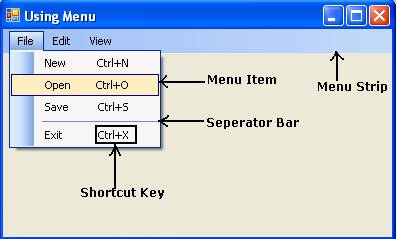
**Methods of ToolTip Control**

Various Methods of ToolTip Control are:

|  |  |
| --- | --- |
| Method Name | Description |
| **SetToolTip** | It is used to specify ToolTip text to be display when mouse pointer is on particular control. **Syntax:** ToolTip1.SetToolTip(ControlName,"ToolTipText") |
| **GetToolTip** | It is used to retrieve ToolTip text associated with particular control. **Syntax:** ToolTip1.GetToolTip(ControlName) |
| **RemoveAll** | It is used to remove all ToolTip text associated with ToolTip Control. **Syntax:** ToolTip1.RemoveAll() |

**Menu Control:**

Menu is one of the most common elements of Graphical User Interface.  
Menu is a one type of control that represents a group of choices to the user and allows user to select any of them according to their requirement.  
Using menu user can organizes various options or commands as per their functionality   
It helps programmer to organize large number of options in a short and easy way.  
It can be attached only with form either SDI or MDI.  
It is displayed immediately under the title bar of the form as shown below:



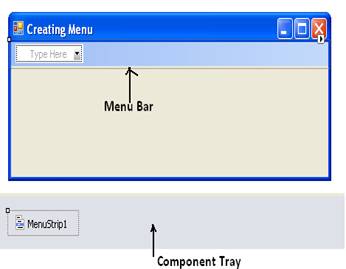
User can assign access key to each menu item for quick access to the menu item through key board.  
User can assign shortcut key to each menu items for accessing them through keyboard.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

**Creating Menu**

In order to create a menu double click on the MenuStrip control in the Tool Box as shown below:  
Create Menu

As you double click on MenuStrip control a menu bar is added just under the title bar of the form and MenuStrip is added in the component tray under the form as shown below:



Once Menu Bar is added to the form you can add, edit and delete menu items and submenu as per your requirements.   
In order to create Menu Item just double click on “Type Here” TextBox and type name of the Menu Item. As you enter name of Menu item “Type Here” TextBox will appear to the right and bottom of the Menu Item as shown below:



You can repeat same procedure in order to create Menu Item and Sub Menu as per your requirement.

**Properties of MenuStrip Control in VB.NET**

|  |  |
| --- | --- |
| Property | Purpose |
| AllowItemReorder | It is used to specify weather user can reorder menu items by holding Alt key or not. It has Boolean value. Default value is false. |
| BackColor | It is used to get or set back color of the MenuStrip. |
| Enabled | It is used to specify weather MenuStrip is enabled or not at run time. It has Boolean value. Default value is true. |
| Font | It is used to set Font Face, Font Style, Font Size and Effects of the text associated with Menu Items of MenuStrip Control. |
| Items | It represents collection of Menu Items contained in Menu Strip control. |
| LayoutStyle | It is used to get or set Layout Style of Menu Strip Control. It has following 5 options: (1) Stack with Overflow  (2) Horizontal Stack With Overflow (3) Vertical Stack With Overflow (4) Flow (5) Table |
| ShowItemToolTips | It is used to specify weather Tool Tip text will be displayed for each menu item or not when mouse is over that menu item. It has Boolean value. Default value is false. |
| TextDirection | It is used to get or set value which determines direction of text in each menu Item. It has following 3 options: (1) Horizontal:   http://www.thecodegallery.com/VBNET/images/MenuStrip5.jpg (2) Vertical90:   http://www.thecodegallery.com/VBNET/images/MenuStrip9.jpg (3) Vertical270: http://www.thecodegallery.com/VBNET/images/MenuStrip10.jpg |
| Visible | It is used to specify weather MenuStrip is visible or not at run time. It has Boolean value. Default value is true. |

**Month and Calendar:**

The calendar control is a functionally rich web control, which provides the following capabilities:

* Displaying one month at a time
* Selecting a day, a week or a month
* Selecting a range of days
* Moving from month to month
* Controlling the display of the days programmatically

**Properties and Events of the Calendar Control**

The calendar control has many properties and events, using which you can customize the actions and display of the control. The following table provides some important properties of the Calendar control:

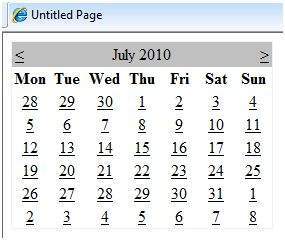
|  |  |
| --- | --- |
| **Properties** | **Description** |
| Caption | Gets or sets the caption for the calendar control. |
| CaptionAlign | Gets or sets the alignment for the caption. |
| CellPadding | Gets or sets the number of spaces between the data and the cell border. |
| CellSpacing | Gets or sets the space between cells. |
| DayHeaderStyle | Gets the style properties for the section that displays the day of the week. |
| DayNameFormat | Gets or sets format of days of the week. |
| DayStyle | Gets the style properties for the days in the displayed month. |
| FirstDayOfWeek | Gets or sets the day of week to display in the first column. |
| NextMonthText | Gets or sets the text for next month navigation control. The default value is >. |
| NextPrevFormat | Gets or sets the format of the next and previous month navigation control. |
| OtherMonthDayStyle | Gets the style properties for the days on the Calendar control that are not in the displayed month. |
| PrevMonthText | Gets or sets the text for previous month navigation control. The default value is <. |
| SelectedDate | Gets or sets the selected date. |
| SelectedDates | Gets a collection of DateTime objects representing the selected dates. |
| SelectedDayStyle | Gets the style properties for the selected dates. |
| SelectionMode | Gets or sets the selection mode that specifies whether the user can select a single day, a week or an entire month. |
| SelectMonthText | Gets or sets the text for the month selection element in the selector column. |
| SelectorStyle | Gets the style properties for the week and month selector column. |
| SelectWeekText | Gets or sets the text displayed for the week selection element in the selector column. |
| ShowDayHeader | Gets or sets the value indicating whether the heading for the days of the week is displayed. |
| ShowGridLines | Gets or sets the value indicating whether the gridlines would be shown. |
| ShowNextPrevMonth | Gets or sets a value indicating whether next and previous month navigation elements are shown in the title section. |
| ShowTitle | Gets or sets a value indicating whether the title section is displayed. |
| TitleFormat | Gets or sets the format for the title section. |
| Titlestyle | Get the style properties of the title heading for the Calendar control. |
| TodayDayStyle | Gets the style properties for today's date on the Calendar control. |
| TodaysDate | Gets or sets the value for today's date. |
| UseAccessibleHeader | Gets or sets a value that indicates whether to render the table header <th> HTML element for the day headers instead of the table data <td> HTML element. |
| VisibleDate | Gets or sets the date that specifies the month to display. |
| WeekendDayStyle | Gets the style properties for the weekend dates on the Calendar control. |

The Calendar control has the following three most important events that allow the developers to program the calendar control. They are:

|  |  |
| --- | --- |
| **Events** | **Description** |
| SelectionChanged | It is raised when a day, a week or an entire month is selected. |
| DayRender | It is raised when each data cell of the calendar control is rendered. |
| VisibleMonthChanged | It is raised when user changes a month. |

**Working with the Calendar Control**

Putting a bare-bone calendar control without any code behind file provides a workable calendar to a site, which shows the months and days of the year. It also allows navigation to next and previous months.



Calendar controls allow the users to select a single day, a week, or an entire month. This is done by using the SelectionMode property. This property has the following values:

|  |  |
| --- | --- |
| **Properties** | **Description** |
| Day | To select a single day. |
| DayWeek | To select a single day or an entire week. |
| DayWeekMonth | To select a single day, a week, or an entire month. |
| None | Nothing can be selected. |

**LinkLabel:**

LinkLabel Control is designed such that it provides the functionality of Hyperlink in window application.   
It is derived from label Control so it also provides all the functionality of Label control.

**Properties of Linklabel Control**

|  |  |
| --- | --- |
| Property | Purpose |
| LinkColor | It is used to get or set Fore color of the Hyperlink in its default state. |
| ActiveLinkColor | It is used to get or set Fore color of the Hyperlink when user clicks it. |
| DisabledLinkColor | It is used to get or set Fore color of the Hyperlink when LinkLabel is disabled. |
| |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | | |
| VisitedLinkColor | It is used to get or set Fore color of the Hyperlink when LinkVisited property of LinkLabel is set to true. |
| LinkVisited | It is used to specify weather Hyperlink is already visited or not. It has Boolean value. Default value is false. |
| Text | It is used to get or set text associated with LinkLabel Control. |
| TextAlign | It is used to get or set alignment of the text associated with LinkLabel Control. |
| ForeColor | It is used to get or set Fore Color of the text associated with LinkLabel Control. |
| BackColor | It is used to get or set Background color of the LinkLabel Control. |
| Enabled | It is used to specify weather LinkLabel control is enabled or not at runtime. It has Boolean value true or false. Default value is true. |
| Visible | It is used to specify weather LinkLabel control is visible or not at runtime. It has Boolean value true or false. Default value is true. |

**Methods of Linklabel Control**

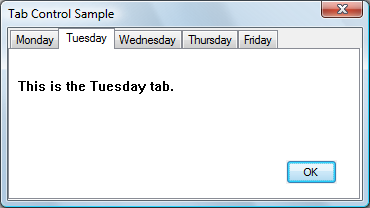
|  |  |
| --- | --- |
| Method | Purpose |
| Show | It is used to Show LinkLabel Control at runtime. |
| Hide | It is used to Hide LinkLabel Control at runtime. |
| Focus | It is used to set input focus on LinkLabel Control. |

**Events of Linklabel Control**

|  |  |
| --- | --- |
| Event | Purpose |
| Link Clicked | It is the default event of LinkLabel Control. It fires each time a user click on a hyperlink of LinkLabel Control. |

**TabControl:**

Windows TabControl is a useful control that allows you display multiple dialogs tabs on a single form by switching between the tabs. A tab acts as another Form that can host other controls.



**Properties**

|  |  |
| --- | --- |
| **Alignment** | Gets or sets the area of the control (for example, along the top) where the tabs are aligned. |
| **Appearance** | Gets or sets the visual appearance of the control's tabs. |
| **DefaultSize** | Overridden. |
| **DisplayRectangle** | Overridden. Gets the display area of the control's tab pages. |
| **DrawMode** | Gets or sets the way that the control's tabs are drawn. |
| **HotTrack** | Gets or sets a value indicating whether the control's tabs change in appearance when the mouse passes over them. |
| **ImageList** | Gets or sets the images to display on the control's tabs. |
| **ItemSize** | Gets or sets the size of the control's tabs. |
| **MultiLine** | Gets or sets a value indicating whether more than one row of tabs can be displayed. |
| **Padding** | Gets or sets the amount of space around each item on the control's tab pages. A Point that specifies the amount of space to pad each item with. The default is (6,3). |
| **RowCount** | Gets the number of rows that are currently being displayed in the control's tab strip. |
| **SelectedIndex** | Gets or sets the index of the currently selected tab page. |
| **SelectedTab** | Gets or sets the currently selected tab page. |
| **ShowToolTips** | Gets or sets a value indicating whether a tab's ToolTip is shown when the mouse passes over the tab. |
| **SizeMode** | Gets or sets the way that the control's tabs are sized. |
| **TabCount** | Gets the number of tabs in the tab strip. |
| **TabPages** | Gets the collection of tab pages in this tab control. |

**Methods**

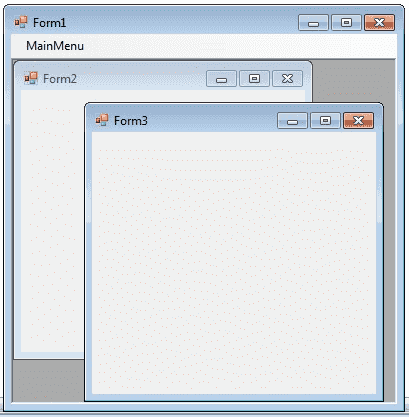
|  |  |
| --- | --- |
| **DeselectTab** | Overloaded. Makes the tab following the specified tab the current tab. |
| **Dispose** | Overloaded. Releases the resources used by the TabControl. |
| **GetControl** | Gets the TabPage control at the specified location. |
| **GetItems** | Overloaded. Gets the TabPage controls that belong to the TabControl. |
| **GetTabRect** | Returns the bounding rectangle for a specified tab in this tab control. |
| **GetToolTipText** | Gets the ToolTip for the specified TabPage. |
| **RemoveAll** | Removes all the tab pages and additional controls from this tab control. |
| **SelectTab** | Overloaded. Makes the specified tab the current tab. |
| **ToString** | Overridden. Returns a string that represents the TabControl control. |
| **UpdateTabSelection** | Sets the Visible property to true for the appropriate TabPage control in the TabPages collection. |

**Events**

|  |  |
| --- | --- |
| **BackgroundImageChanged** | Occurs when the value of the BackgroundImage property changes. |
| **BackgroundImageLayoutChanged** | Occurs when the value of the BackgroundImageLayout property changes. |
| **Deselected** | (Added in 2.0) Occurs when a tab is deselected. |
| **Deselecting** | (Added in 2.0) Occurs before a tab is deselected, enabling a handler to cancel the tab change. |
| **DrawItem** | Occurs when the TabControl needs to paint each of its tabs if the DrawMode property is set to OwnerDrawFixed. |
| **ForeColorChanged** | Occurs when the value of the ForeColor property changes. |
| **Selected** | (Added in 2.0) Occurs when a tab is selected. |
| **SelectedIndexChanged** | Occurs when the SelectedIndex property is changed. |
| **Selecting** | (Added in 2.0) Occurs before a tab is selected, enabling a handler to cancel the tab change. |
| **TabIndexChanged** | Occurs when the TabIndex property value changes. (Inherited from Control.) |
| **TextChanged** | Occurs when the value of the Text property changes. |
| **Validating** | Occurs when the control is validating. This is fired again when the form is closed ? |

**MDI Form:**

A Multiple Document Interface (MDI) programs can display multiple child windows inside them. This is in contrast to single document interface (SDI) applications, which can manipulate only one document at a time. Visual Studio Environment is an example of Multiple Document Interface (MDI) and notepad is an example of an SDI application. MDI applications often have a Window menu item with submenus for switching between windows or documents.



Any windows can become an MDI parent, if you set the **IsMdiContainer** property to True.

**IsMdiContainer** = **true**;

The following C# program shows a MDI form with two child forms. Create a new C# project, then you will get a default form Form1 . Then add two mnore forms in the project (Form2 , Form 3) . Create a Menu on your form and call these two forms on menu click event.

**MDI Form demonstration with menu**

using System;

using System.Drawing;

using System.Windows.Forms;

namespace WindowsFormsApplication1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

**IsMdiContainer** = true;

}

private void menu1ToolStripMenuItem\_Click(object sender, EventArgs e)

{

Form2 frm2 = new Form2();

frm2.Show();

**frm2.MdiParent = this;**

}

private void menu2ToolStripMenuItem\_Click(object sender, EventArgs e)

{

Form3 frm3 = new Form3();

frm3.Show();

**frm3.MdiParent = this;**

}

}

}

**C# - Delegate**

A function can have one or more parameters of different data types, but what if you want to pass a function itself as a parameter? How does C# handle the callback functions or event handler? The answer is - **delegate**.

A delegate is like a pointer to a function. It is a reference type data type and it holds the reference of a method. All the delegates are implicitly derived from System.Delegate class.

A delegate can be declared using **delegate** keyword followed by a function signature as shown below.

**Delegate Syntax:**

<access modifier> delegate <return type> <delegate\_name>(<parameters>)

The following example declares a Print delegate.

**Example: Declare delegate**

public delegate void Print(int value);

The Print delegate shown above, can be used to point to any method that has same return type & parameters declared with Print. Consider the following example that declares and uses Print delegate.

Example: C# delegate

class Program

{

// declare delegate

public delegate void Print(int value);

static void Main(string[] args)

{

// Print delegate points to PrintNumber

Print printDel = PrintNumber;

// or

// Print printDel = new Print(PrintNumber);

printDel(100000);

printDel(200);

// Print delegate points to PrintMoney

printDel = PrintMoney;

printDel(10000);

printDel(200);

}

public static void PrintNumber(int num)

{

Console.WriteLine("Number: {0,-12:N0}",num);

}

public static void PrintMoney(int money)

{

Console.WriteLine("Money: {0:C}", money);

}

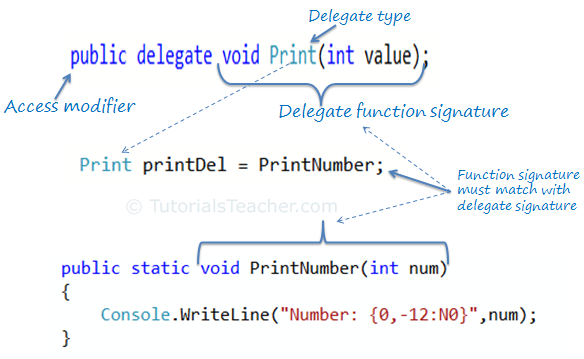
}

Output:

Number: 10,000   
Number: 200   
Money: $ 10,000.00   
Money: $ 200.00

In the above example, we have declared Print delegate that accepts *int* type parameter and returns void. In the Main() method, a variable of Print type is declared and assigned a PrintNumber method name. Now, invoking Print delegate will in-turn invoke PrintNumber method. In the same way, if the Print delegate variable is assigned to the PrintMoney method, then it will invoke the PrintMoney method.

The following image illustrates the delegate.

[](https://www.tutorialsteacher.com/Content/images/csharp/delegate.png)delegate in C#

Optionaly, a delegate object can be created using the new operator and specify a method name, as shown below:

Print printDel = new Print(PrintNumber);

**Invoking Delegate**

The delegate can be invoked like a method because it is a reference to a method. Invoking a delegate will in-turn invoke a method which id refered to. The delegate can be invoked by two ways: using () operator or using the Invoke() method of delegate as shown below.

Example: Invoking a delegate

Print printDel = PrintNumber;

printDel.Invoke(10000);

//or

printDel(10000);

Number: 10000   
Number: 10000

**Pass Delegate as a Parameter**

A method can have a parameter of a delegate type and can invoke the delegate parameter.

Example: Delegate Parameter

public static void PrintHelper(Print delegateFunc, int numToPrint)

{

delegateFunc(numToPrint);

}

In the above example, PrintHelper method has a delegate parameter of Print type and invokes it like a function:*delegateFunc(numToPrint)*.

The following example shows how to use PrintHelper method that includes delegate type parameter.

Example: Delegate parameter

class Program

{

public delegate void Print(int value);

static void Main(string[] args)

{

PrintHelper(PrintNumber, 10000);

PrintHelper(PrintMoney, 10000);

}

public static void PrintHelper(Print delegateFunc, int numToPrint)

{

delegateFunc(numToPrint);

}

public static void PrintNumber(int num)

{

Console.WriteLine("Number: {0,-12:N0}",num);

}

public static void PrintMoney(int money)

{

Console.WriteLine("Money: {0:C}", money);

}

}

Output:

Number: 10,000   
Money: $ 10,000.00

**Multicast Delegate**

The delegate can points to multiple methods. A delegate that points multiple methods is called a multicast delegate. The "+" operator adds a function to the delegate object and the "-" operator removes an existing function from a delegate object.

Example: Multicast delegate

public delegate void Print(int value);

static void Main(string[] args)

{

Print printDel = PrintNumber;

printDel += PrintHexadecimal;

printDel += PrintMoney;

printDel(1000);

printDel -=PrintHexadecimal;

printDel(2000);

}

public static void PrintNumber(int num)

{

Console.WriteLine("Number: {0,-12:N0}",num);

}

public static void PrintMoney(int money)

{

Console.WriteLine("Money: {0:C}", money);

}

public static void PrintHexadecimal(int dec)

{

Console.WriteLine("Hexadecimal: {0:X}", dec);

}

Output:

Number: 1,000   
Hexadecimal: 3EB   
Money: $ 1,000.00   
Number: 2,000   
Money: $2,000.00

As you can see in the above example, Print delegates becomes a multicast delegate because it points to three methods - PrintNumber, PrintMoney & PrintHexadecimal. So invoking printDel will invoke all the methods sequentially.

Points to Remember :

1. Delegate is a function pointer. It is reference type data type.
2. Syntax: *public delegate void <function name>(<parameters>)*
3. A method that is going to assign to delegate must have same signature as delegate.
4. Delegates can be invoke like a normal function or Invoke() method.
5. Multiple methods can be assigned to the delegate using "+" operator. It is called multicast delegate.