Advanced Form Controls

GroupBox Controls

A **GroupBox control** is a container control that is used to place Windows Forms child controls in a group. The purpose of a GroupBox is to define user interfaces where we can categorize related controls in a group.

Properties

- **Location, Height, Width, and Size** - The Location property takes a Point that specifies the starting position of the GroupBox on a Form. The Size property specifies the size of the control. We can also use Width and Height property instead of Size property.

```
authorGroup.Location = newPoint(10, 10);
authorGroup.Width = 250;
authorGroup.Height = 200;
```

- **Background and Foreground** - BackColor and ForeColor properties are used to set background and foreground color of a GroupBox respectively.

```
authorGroup.BackColor = Color.LightBlue;
authorGroup.ForeColor = Color.Maroon;
```

- **Name** - Name property represents a unique name of a GroupBox control. It is used to access the control in the code.

```
authorGroup.Name = "GroupBox1";
```

- **Text -** Text property of a GroupBox represents the header text of a GroupBox control. The following code snippet sets the header of a GroupBox control.

```
authorGroup.Text = "Author Details";
```

- **Font -** Font property represents the font of header text of a GroupBox control. If you click on the Font property in Properties window, you will see Font name, size and other font options. The following code snippet sets Font property at run-time.

```
dynamicGroupBox.Font = newFont("Georgia", 12);
```

ListBox Controls

A ListBox control provides a user interface to display a list of items. Users can select one or more items from the list. A ListBox may be used to display multiple columns and these columns may have images and other controls.

Properties

- **Name -** represents a unique name of a ListBox control. It is used to access the control in the code. The following code snippet sets and gets the name and text of a ListBox control:

```
1.  listBox1.Name = "ListBox1";
```

- **Location, Height, Width and Size** - takes a Point that specifies the starting position of the ListBox on a Form. You may also use the Left and Top properties to specify the location of a control from the top-left corner of the Form. The Size property specifies the size of the control. We can also use the Width and Height properties instead of the Size property. The following code snippet sets the Location, Width, and Height properties of a ListBox control:

```
1.  listBox1.Location = new System.Drawing.Point(12, 12);
2.  listBox1.Size = new System.Drawing.Size(245, 200);
```

- **Font -** represents the font of the text of a ListBox control. If you click on the Font property in Properties window, you will see the Font name, size and other font options. The following code snippet sets the Font property at run-time:

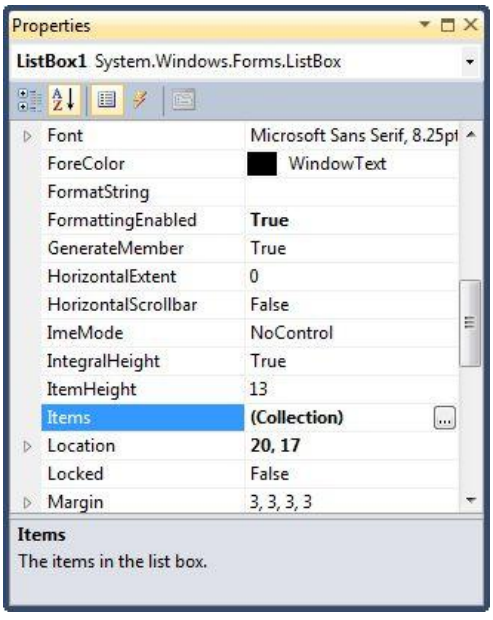
```
1.  listBox1.Font = new Font("Georgia", 16);
```

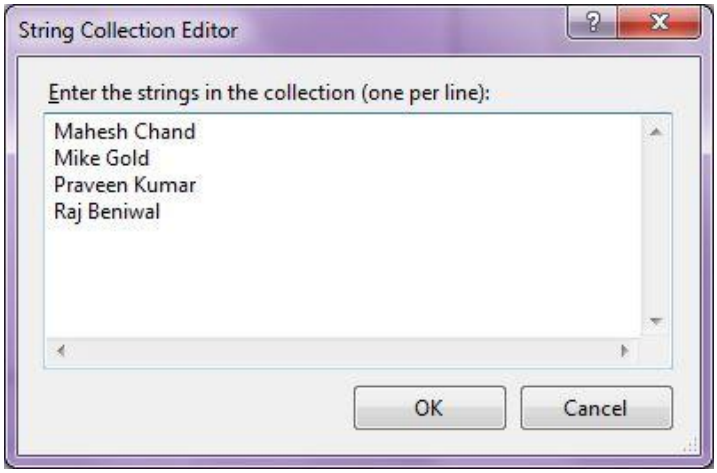
- **Background and Foreground -** used to set the background and foreground colors of a ListBox respectively. If you click on these properties in the Properties window, then the Color Dialog pops up.

```
1.  listBox1.BackColor = System.Drawing.Color.Orange;
2.  listBox1.ForeColor = System.Drawing.Color.Black;
```

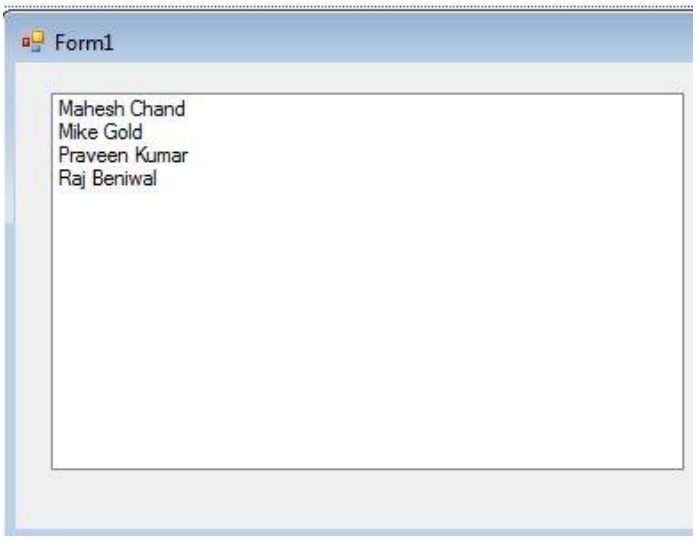
- **ListBox Items** - The Items property is used to add and work with items in a ListBox. We can add items to a ListBox at design-time from the Properties Window by clicking on the Items Collection.

When you click on the Collections, the String Collection Editor window will pop up where you can type strings. Each line added to this collection will become a ListBox item.





The ListBox looks after adding an item.



You can add the same items at run-time by using the following code snippet:

```
1.  listBox1.Items.Add("Mahesh Chand");
2.  listBox1.Items.Add("Mike Gold");
3.  listBox1.Items.Add("Praveen Kumar");
4.  listBox1.Items.Add("Raj Beniwal");
```

Getting All Items

To get all items, we use the Items property and loop through it to read all the items. The following code snippet loops through all items and adds item contents to a StringBuilder and displays it in a MessageBox:

```
1.  private void GetItemsButton_Click(object sender, EventArgs e)
2.  {
3.      System.Text.StringBuilder sb = new System.Text.StringBuilder();
4.      foreach (object item in listBox1.Items)
5.      {
```

```
            sb.Append(item.ToString());
            sb.Append(" ");
        }
        MessageBox.Show(sb.ToString());
    }
```

Selected Text and Item

The Text property is used to set and get text of a ListBox. The following code snippet sets and gets the current text of a ListBox:

```
MessageBox.Show(listBox1.Text);
```

We can also get text associated with the currently selected item using the Items property:

```
string selectedItem = listBox1.Items[listBox1.SelectedIndex].ToString();
```

Why is the value of ListBox.SelectedText Empty?

The SelectedText property gets and sets the selected text in a ListBox only when a ListBox has focus on it. If the focus moves away from a ListBox then the value of SelectedText will be an empty string. To get the current text in a ListBox when it does not have focus, use the Text property.

Selection Mode and Selecting Items

The SelectionMode property defines how items are selected in a ListBox. The SelectionMode value can be one of the following four SelectionMode enumeration values:

- *None*: No item can be selected.
- *One*: Only one item can be selected.
- *MultiSimple*: Multiple items can be selected.
- *MultiExtended*: Multiple items can be selected, and the user can use the SHIFT, CTRL, and arrow keys to make selections.

To select an item in a ListBox, we can use the SetSelect method that takes an item index and a true or false value where the true value represents the item to be selected.

The following code snippet sets a ListBox to allow multiple selection and selects the second and third items in the list:

```
listBox1.SelectionMode = SelectionMode.MultiSimple;
listBox1.SetSelected(1, true);
listBox1.SetSelected(2, true);
```

We can clear all selected items by calling the ClearSelected method, as in:

```
listBox1.ClearSelected();

```

How to disable item selection in a ListBox?

Just set the SelectionMode property to None.

Sorting Items

If the Sorted property is set to true then the ListBox items are sorted. The following code snippet sorts the ListBox items:

```
listBox1.Sorted = true;
```

Find Items

The FindString method is used to find a string or substring in a ListBox. The following code snippet finds a string in a ListBox and selects it if found:

```
private void FindItemButton_Click(object sender, EventArgs e)
{
    listBox1.ClearSelected();

    int index = listBox1.FindString(textBox1.Text);

    if (index < 0)
    {
        MessageBox.Show("Item not found.");
        textBox1.Text = String.Empty;
    }
    else
    {
        listBox1.SelectedIndex = index;
    }
}
```

Radio Button

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.

Properties

- **Location, Height, Width, and Size -** takes a Point that specifies the starting position of the RadioButton on a Form. You may also use Left and Top properties to specify the location of a control from the left top corner of the Form. The Size property specifies the size of the control. We can also use Width and Height property instead of Size property. The following code snippet sets Location, Width, and Height properties of a RadioButton control.

```
dynamicRadioButton.Location = new Point(20, 150);
dynamicRadioButton.Height = 40;
dynamicRadioButton.Width = 300;
```

- **Background, Foreground, BorderStyle -** BackColor and ForeColor properties are used to set background and foreground color of a RadioButton respectively. If you click on these properties in Properties window, the Color Dialog pops up.

```
dynamicRadioButton.BackColor = Color.Red;
dynamicRadioButton.ForeColor = Color.Blue;
```

- **Name -** represents a unique name of a RadioButton control. It is used to access the control in the code. The following code snippet sets and gets the name and text of a RadioButton control.

```
dynamicRadioButton.Name = "DynamicRadioButton";
string name = dynamicRadioButton.Name;
```

- **Text and TextAlign -** represents the current text of a RadioButton control. The TextAlign property represents text alignment that can be Left, Center, or Right. The following code snippet sets the Text and TextAlign properties and gets the size of a RadioButton control.

```
dynamicRadioButton.Text = "I am a Dynamic RadioButton";
dynamicRadioButton.TextAlign = ContentAlignment.MiddleCenter;
```

- **Font -** represents the font of text of a RadioButton control. If you click on the Font property in Properties window, you will see Font name, size, and other font options. The following code snippet sets the Font property at run-time.

```
dynamicRadioButton.Font = new Font("Georgia", 16);
```

- **Read RadioButton Contents -** The simplest way of reading a RadioButton control's content is using the Text property. The following code snippet reads contents of a RadioButton in a string.

```
string RadioButtonContents = dynamicRadioButton.Text;
```

- **Appearance -** can be used to set the appearance of a RadioButton to a Button or a RadioButton. The Button look does not have a round select option. The following property makes a RadioButton look like a Button control.

```
dynamicRadioButton.Appearance = Appearance.Button;
```

- **Check Mark Alignment -** used to align the check mark in a RadioButton. By using CheckAlign and TextAlign properties, we can place text and check mark to any position on a RadioButton we want. The following code snippet aligns radio button round circle and text to middle-right and creates a RadioButton.

```
dynamicRadioButton.CheckAlign = ContentAlignment.MiddleRight;
dynamicRadioButton.TextAlign = ContentAlignment.MiddleRight;
```

- **Image in RadioButton -** used to set the background as an image. The Image property needs an Image object. The Image class has a static method called

FromFile that takes an image file name with full path and creates an Image object.

```
1. dynamicRadioButton.Image = Image.FromFile(@ "C:\Images\Dock.jpg");
2. dynamicRadioButton.ImageAlign = ContentAlignment.MiddleRight;
3. dynamicRadioButton.FlatStyle = FlatStyle.Flat;
```

RadioButton States

A typical RadioButton control has two possible states – Checked and Unchecked. Checked state is when the RadioButton has check mark on and Unchecked is when the RadioButton is not checked. Typically, we use a mouse to check or uncheck a RadioButton.

Checked property is true when a RadioButton is in checked state.

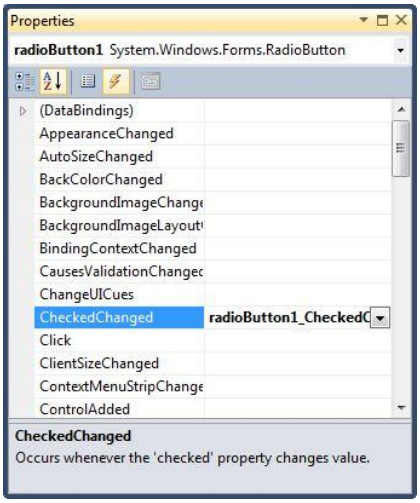
```
1. dynamicRadioButton.Checked = true;
```

Usually, we check if a RadioButton is checked or not and decide to take an action on that state something like following code snippet.

```
1. if (dynamicRadioButton.Checked) {
2.     // Do something when RadioButton is checked
3. } else {
4.     // Do something here when RadioButton is not checked
5. }
```

AutoCheck property represents whether the Checked or CheckState values and the RadioButton's appearance are automatically changed when the RadioButton is clicked. By default this property is true but if set to false.

```
1. dynamicRadioButton.AutoCheck = false;
```



RadioButton Checked Event Hander

CheckedChanged event occurs when the value of the Checked property changes. To add this event handler, you go to Events window and double click on CheckedChanged events.

The following code snippet defines and implements these events and their respective event handlers. You can write this code to implement CheckedChanged event dynamically.

```
1. dynamicRadioButton.CheckedChanged += new System.EventH
   andler(RadioButtonCheckedChanged);
2. privatevoid RadioButtonCheckedChanged(object sender, EventArgs e) {}
```

CheckBox Controls

A **CheckBox control** allows users to select single or multiple items from a list of items. In this article, I will discuss how to create a CheckBox control in Windows

Forms at design-time as well as run-time. After that, I will discuss how to use its various properties and methods.

Properties

- **Location, Height, Width, and Size** - takes a Point that specifies the starting position of the CheckBox on a Form. You may also use Left and Top properties to specify the location of a control from the left top corner of the Form. The Size property specifies the size of the control. We can also use Width and Height property instead of Size property. The following code snippet sets Location, Width, and Height properties of a CheckBox control.

```
1.  // Set CheckBox properties
2.  dynamicCheckBox.Location = new Point(20, 150);
3.  dynamicCheckBox.Height = 40;
4.  dynamicCheckBox.Width = 300;
5.  Background, Foreground, BorderStyle;
```

- **BackColor and ForeColor -** BackColor and ForeColor properties are used to set background and foreground color of a CheckBox respectively. If you click on these properties in Properties window, the Color Dialog pops up.

```
1.  // Set background and foreground
2.  dynamicCheckBox.BackColor = Color.Red;
3.  dynamicCheckBox.ForeColor = Color.Blue;
```

- **Name -** represents a unique name of a CheckBox control. It is used to access the control in the code. The following code snippet sets and gets the name and text of a CheckBox control.

```
1.  dynamicCheckBox.Name = "DynamicCheckBox";
2.  string name = dynamicCheckBox.Name;
```

- **Text and TextAlign -** represents the current text of a CheckBox control. The TextAlign property represents text alignment that can be Left, Center, or Right. The following code snippet sets the Text and TextAlign properties and gets the size of a CheckBox control.

```
1.  dynamicCheckBox.Text = "I am a Dynamic CheckBox";
2.  dynamicCheckBox.TextAlign = ContentAlignment.MiddleCenter;
```

- **Check Mark Alignment** - used to align the check mark in a CheckBox. By using CheckAlign and TextAlign properties, we can place text and check mark to any position on a CheckBox we want. The following code snippet aligns check mark to middle-center and text to top-right and creates a CheckBox.

```
1.  dynamicCheckBox.CheckAlign = ContentAlignment.MiddleCenter;
2.  dynamicCheckBox.TextAlign = ContentAlignment.TopRight;
```

- **Font -** represents the font of text of a CheckBox control. If you click on the Font property in Properties window, you will see Font name, size, and other font options. The following code snippet sets Font property at run-time.

```
dynamicCheckBox.Font = new Font("Georgia", 16);
```

- **Read CheckBox Contents -** The simplest way of reading a CheckBox control contents is using the Text property. The following code snippet reads contents of a CheckBox in a string.

```
string CheckBoxContents = dynamicCheckBox.Text;
```

- **Appearance -** can be used to set the appearance of a CheckBox to a Button or a CheckBox. The following property makes a CheckBox look like a Button control.

```
dynamicCheckBox.Appearance = Appearance.Button;
```

- **Image in CheckBox -** used to set the background as an image. The Image property needs an Image object. The Image class has a static method called FromFile that takes an image file name with full path and creates an Image object.

```
// Assign an image to the CheckBox.
dynamicCheckBox.Image = Image.FromFile(@"C:\Images\Dock.jpg");

// Align the image and text on the CheckBox.
dynamicCheckBox.ImageAlign = ContentAlignment.MiddleRight;

// Give the CheckBox a flat appearance.
dynamicCheckBox.FlatStyle = FlatStyle.Flat;
```

CheckBox States

A typical CheckBox control has two possible states – Checked and Unchecked. The checked state is when the CheckBox has a check mark on and Unchecked is when the CheckBox is not checked. Typically, we use a mouse to check or uncheck a CheckBox.

Checked property is true when a CheckBox is in the checked state.

```
dynamicCheckBox.Checked = true;
```

CheckState property represents the state of a CheckBox. It can be checked or unchecked. Usually, we check if a CheckBox is checked or not and decide to take an action on that state something like following code snippet.

```
if (dynamicCheckBox.Checked)
{
// Do something when CheckBox is checked
}
```

```
else
{
// Do something here when CheckBox is not checked
}
```

ThreeState is a new property added to the CheckBox in latest versions of Windows Forms. When this property is true, the CheckBox has three states - Checked, Unchecked, and Indeterminate. The following code snippet sets and checks CheckState.

```
dynamicCheckBox.CheckState = CheckState.Indeterminate;

if (dynamicCheckBox.CheckState == CheckState.Checked)
{
}
else if (dynamicCheckBox.CheckState == CheckState.Indeterminate)
{
}
else
{
}
```

AutoCheck property represents whether the Checked or CheckState values and the CheckBox's appearance are automatically changed when the CheckBox is clicked. By default this property is true but if set to false.

```
dynamicCheckBox.AutoCheck = false;
```

CheckBox Checked Event Hander

CheckedChanged and CheckStateChanged are two important events for a CheckBox control. The CheckedChanged event occurs when the value of the Checked property changes. The CheckStateChanged event occurs when the value of the CheckState property changes.
To add these event handlers, you go to Events window and double click on CheckedChanged and CheckedStateChanged events.

The following code snippet defines and implements these events and their respective event handlers.

```
dynamicCheckBox.CheckedChanged += new System.EventHandler(CheckBoxCheckedChanged);
dynamicCheckBox.CheckStateChanged += new System.EventHandler(CheckBoxCheckedChanged);

private void CheckBoxCheckedChanged(object sender, EventArgs e)
```

```
{

}
private void CheckBoxCheckedChanged(object sender, EventArgs e)
{

}
```

PictureBox

PictureBox control is used to display images in Windows Forms. In this article, I will discuss how to use a PictureBox control to display images in Windows Forms applications.

Creating a PictureBox

```
PictureBox imageControl = newPictureBox();
imageControl.Width = 400;
imageControl.Height = 400;
Controls.Add(imageControl);
```

Display an Image

```
privatevoid DisplayImage()
{
    PictureBox imageControl = newPictureBox();
    imageControl.Width = 400;
    imageControl.Height = 400;
    Bitmap image = newBitmap("C:\\Images\\Creek.jpg");
    imageControl.Dock = DockStyle.Fill;
    imageControl.Image = (Image) image;
    Controls.Add(imageControl);
}
```

SizeMode

SizeMode property is used to position an image within a PictureBox. It can be Normal, StretchImage, AutoSize, CenterImage, and Zoom. The following code snippet sets SizeMode property of a PictureBox control.

imageControl.SizeMode = PictureBoxSizeMode.CenterImage;