

## Windows Forms and Containers in the User Interface

Windows Forms are the basis for most Microsoft Windows applications and can be configured to provide a variety of user interface (UI) options. The developer can create forms of various sizes and shapes and customize them to the user's needs. Forms are hosts for controls, which provide the main functionality of the UI. Special controls called container controls can be used to control the layout of the UI.

### Adding Forms to Your Project

Most projects will require more than one form. You can add and configure additional forms at design time, or you can create instances of predefined forms in code at run time. To add a new form to your project at design time, first, from the Project menu, select Add Windows Form. The Add New Item dialog box opens. Then, select Windows Form and type a name for the new form in the Name box. Click Add to add the form to the development environment.

You can add and configure at design time as many forms as your application needs. You can also create new instances of forms in your code. This method is most often employed when you want to display a form that has already been designed. To access a form's methods and properties from inside its code, use the keyword **this** in ( C#) code. For example:

```
this.Text = " Main Page";
```

You can also create new instances of forms at run time by declaring a variable that represents a type of form and creating an instance of that form. To add a form to your application at run time, declare and instantiate a variable that represents your form. This example assumes that you have already designed a form named Form1 in your project:

```
Form1 myForm;  
myForm = new Form1();  
// Displays the new form  
myForm.Show();
```

### Properties of Windows Forms

The visual appearance of your UI is an important part of your application. A UI that is poorly designed is difficult to learn and will increase training time and expense. You can modify the appearance of your UI by using Windows Forms properties. Windows Forms contain a variety of properties that allow you to customize the look and feel of

the form. You can view and change these properties in the Properties window of the Designer.

Table 2-1 summarizes some of the Windows Forms properties that are important in the look and feel of the application. Note that this is not an exhaustive list of all Windows Forms properties but a selected subset.

TABLE 2-1 Some Properties of the Form Class

Property	Description
Name.	<i>Sets the name of the Form class shown in the Designer. This property can be set only at design time.</i>
BackColor	<i>Indicates the background color of the form.</i>
BackgroundImage	<i>Indicates the background image of the form.</i>
BackgroundImageLayout	<i>Determines how the image indicated by the BackgroundImage property will be laid out on the form. If no background image is selected, this property has no effect.</i>
ControlBox	<i>Determines whether the form has a Control/System menu box.</i>
Cursor	<i>Indicates the cursor that appears when the cursor is moved over the form.</i>
Enabled	<i>Determines whether the form is able to receive user input. If Enabled is set to False, all controls contained by the form are likewise disabled.</i>
Font	<i>Sets the default font for the form. All controls contained by the form will also adopt this font unless their Font property is set separately.</i>
ForeColor	<i>Indicates the forecolor of the form, which is the color used to display text. All controls contained by the form will also adopt this forecolor unless their forecolor property is set separately.</i>
FormBorderStyle	<i>Indicates the appearance and behavior of the form border and title bar.</i>
HelpButton	<i>Indicates whether the form has a Help button.</i>
Icon	<i>Indicates the icon that is used to represent this form.</i>
Location	<i>When the StartPosition property is set to Manual, this property indicates the starting location of the form relative to the upper left-hand corner of the screen.</i>
MaximizeBox	<i>Indicates whether the form has a maximize box.</i>
MaximumSize	<i>Determines the maximum size for the form. If this property is set to a size of (0,0), the form has no upper size limit.</i>
MinimizeBox	<i>Indicates whether the form has a minimize box.</i>

<i>MinimumSize</i>	<i>Determines the minimum size to which the user can resize the form.</i>
<i>Opacity</i>	<i>Represents the opacity, or conversely the transparency of the form from 0% to 100%. A form with 100% opacity is completely opaque, and a form with 0% opacity is completely transparent.</i>
<i>Size</i>	<i>Gets and sets the initial size of the form.</i>
<i>StartPosition</i>	<i>Indicates the position of the form when the form is first displayed.</i>
<i>Text</i>	<i>Determines the text caption of the form.</i>
<i>TopMost</i>	<i>Indicates whether the form always appears above all other forms that do not have this property set to True</i>
<i>Visible</i>	<i>Determines whether the form is visible when running.</i>
<i>WindowState</i>	<i>Determines whether the form is minimized, maximized, or set to the size indicated by the Size property when first shown.</i>

## Setting the Title of the Form

You can change the title of a form by changing the Text property. To change the title of a form at design time, set the Text property of the form in the Property Grid. To change the title of a form at run time, set the Text property of the form in code, as shown in the following code:

```
// C#  
Form1.Text = "Please enter your address";
```

## Setting the Border Style of the Form

The border style of a form determines how the border of the form looks and, to a certain extent, how a form behaves at run time. Depending on the setting, the `FormBorderStyle` property can control how the border appears, whether a form is resizable by the user at run time, and whether various control boxes appear (although these are also determined by other form properties).

The `FormBorderStyle` property has seven possible values, which are explained in Table 2-2.

TABLE 2-2 Values for the FormBorderStyle Property

Property	Description
<i>None</i>	<i>The form has no border and has no minimize, maximize, help, or control boxes.</i>
<i>FixedSingle</i>	<i>The form has a single border and the user cannot resize it. It can have a minimize, maximize, help, or control box as determined by other properties.</i>
<i>Fixed3D</i>	<i>The form's border has a three-dimensional appearance and the user cannot resize it. It can have a minimize, maximize, help, or control box as determined by other properties.</i>
<i>FixedDialog</i>	<i>The form has a single border and the user cannot resize it. Additionally, it has no control box. It can have a minimize, maximize, or help box as determined by other properties.</i>
<i>Sizable</i>	<i>This is the default setting for a form. The user can resize it, and it can contain a minimize, maximize, or help box as determined by other properties.</i>
<i>FixedToolWindow</i>	<i>The form has a single border and the user cannot resize it. The window contains no boxes except the close box</i>
<i>SizableToolWindow</i>	<i>The form has a single border and the user can resize it. The window contains no boxes except the close box</i>

You can set the `FormBorderStyle` property at either design time or run time. To change the border style of a form at design time, set the `FormBorderStyle` property in the Property Grid. To change the border style of a form at run time, set the `FormBorderStyle` property in code, as shown in the following example:

```
// C#  
aForm.FormBorderStyle = FormBorderStyle.Fixed3D;
```

## Setting the Startup State of the Form

The `WindowState` property determines what state the form is in when it first opens. The `WindowState` property has three possible values: `Normal`, `Minimized`, and `Maximized`. The default setting is `Normal`. When the `WindowState` property is set to `Normal`, the form will start at the size determined by the `Size` property. When the `WindowState` property is set to `Minimized`, the form will start up minimized in the taskbar. When the `WindowState` property is set to `Maximized`, the form will start up maximized. Although you can set this property at run time, doing so will have no effect on the state of the form. Thus it is useful to set this property in the Property Grid at design time only.

## Resizing the Form

When the `WindowState` property is set to `Normal`, it will start at the size determined by the `Size` property. The `Size` property is actually an instance of the `Size` structure, which has two members, `Width` and `Height`. You can resize the form by setting the `Size` property in the Property Grid, or you can set the `Width` and `Height` separately by expanding the `Size` property and setting the values for the individual fields. You can also resize the form by grabbing and dragging the lower right-hand corner, the lower edge, or the right-hand edge of the form in the Designer. As the form is visibly resized in the Designer, the `Size` property is automatically set to the new size.

You can resize the form at run time by setting the `Size` property in code. The `Width` and `Height` fields of the `Size` property are also exposed as properties of the form itself. You can set either the individual `Width` and `Height` properties or the `Size` property to a new instance of the `Size` structure, as shown in the following example:

```
// C#  
// Set the Width and Height separately  
aForm.Width = 300;  
aForm.Height = 200;  
// Set the Size property to a new instance of the Size  
structure  
aForm.Size = new Size(300,200);
```

Note that if the form's `StartPosition` property is set to `WindowsDefaultBounds`, the size will be set to the window's default rather than to the size indicated by the `Size` property.

## Opacity and Transparency in Forms

You can use the `Opacity` property to create striking visual effects in your form. The `Opacity` property sets the transparency of the form. When set in the Property Grid, the opacity value can range from 0 percent to 100 percent, indicating the degree of opacity. An opacity of 100 percent indicates a form that is completely opaque (solid and visible), and a value of 0 percent indicates a form that is completely transparent. Values between 0 percent and 100 percent result in a partially transparent form.

You can also set the `Opacity` property in code. When the `Opacity` property is set in code, it is set to a value between 0 and 1, with 0 representing complete transparency and 1 representing complete opacity. The following example demonstrates how to set a form's opacity to 50 percent:

```
// C#  
aForm.Opacity = 0.5;
```

## Setting the Startup Form

The startup object is specified in the Main method. By default, this method is located in a class called Program.cs, which Visual Studio automatically creates. The Program.cs class contains, by default, a Main method, as follows:

```
static void Main()  
{  
    Application.EnableVisualStyles();  
    Application.SetCompatibleTextRenderingDefault(false);  
    Application.Run(new Form1());  
}
```

The startup object is indicated by the line :

```
Application.Run(new Form1());
```

You can set the startup form for the project by changing this line in the Program.cs class to the form that you want to start the application. For example, if you wanted a form called myForm to be the startup form, you would change this line to read as follows:

```
Application.Run(new myForm());
```

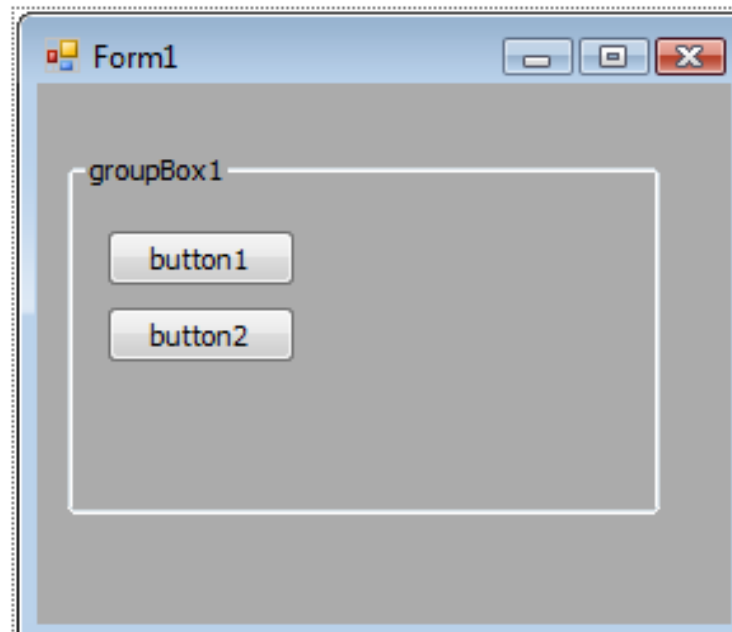
## Managing Control Layout with Container Controls

Container controls are specialized controls that serve as a customizable container for other controls. Examples of container controls include the Panel, FlowLayoutPanel, and SplitContainer controls. Container controls give the form logical and physical subdivisions that can group other controls into consistent UI subunits. For example, you might contain a set of related RadioButton controls in a GroupBox control. The use of container controls helps create a sense of style or information flow in your UI and allows you to manipulate contained controls in a consistent fashion.

When a container control holds other controls, changes to the properties of the container control can affect the contained controls. For example, if the Enabled property of a panel is set to False, all of the controls contained in the panel are disabled. Likewise, changes to properties related to the UI, such as BackColor, Visible, or Font, are also applied to the contained controls. Note that you can still manually change any property inside a contained control, but if the container is disabled, all controls inside that container will be inaccessible regardless of their individual property settings.

## The GroupBox Control

The GroupBox control is a container control that appears as a subdivision of the form surrounded by a border. It does not provide scrollbars, like the Panel control, nor does it provide any kind of specialized layout capabilities. A GroupBox can have a caption, which is set by the `Text` property, or it can appear without a caption when the `Text` property is set to an empty string.



## The Panel Control

The Panel control creates a subsection of a form that can host other controls. The Panel can be indistinguishable from the rest of the surrounding form, or it can be surrounded by a border as determined by the `BorderStyle` property. A Panel can have a `BorderStyle` property of `None`, which indicates no border; `FixedSingle`, which indicates a single edge around the Panel; or `Fixed3D`, which represents a border with a three-dimensional appearance.

The Panel control is a scrollable control, which means that it supports horizontal and vertical scroll bars. Controls can be hosted in the Panel outside of its visible bounds. When the `AutoScroll` property is set to `True`, scroll bars will automatically be available if any controls are placed outside of the visible bounds of the control. If the `AutoScroll` property is set to `False`, controls outside the visible bounds of the Panel are inaccessible. Important properties of the Panel control are shown in Table 2-3.

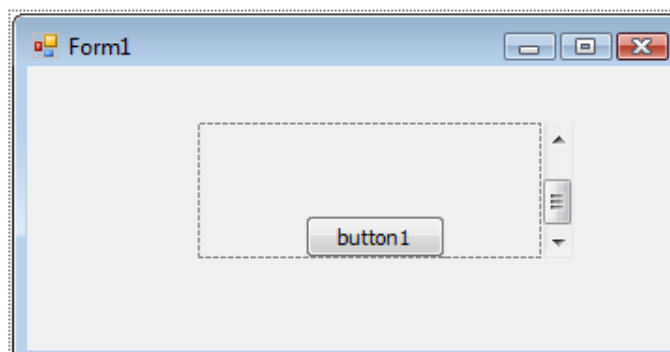
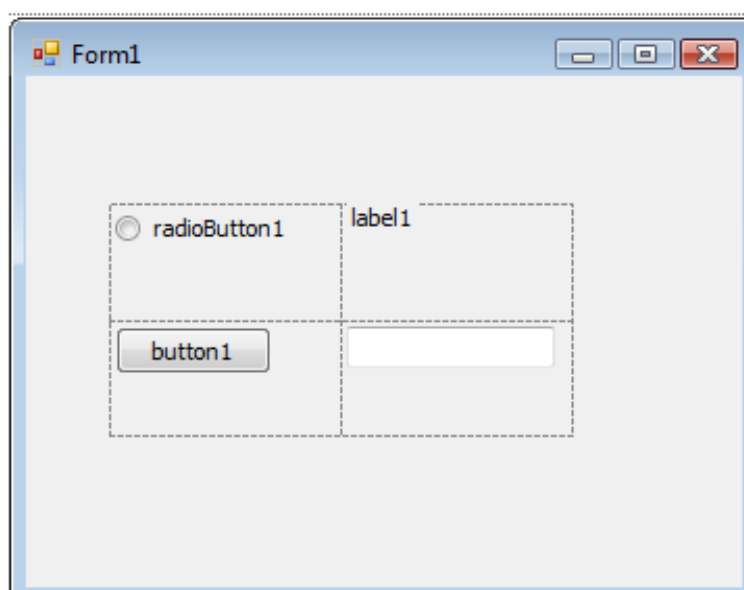


TABLE 2-3 Important Properties of the Panel Control

Property	Description
<i>AutoScroll</i>	<i>Determines if the Panel will display scroll bars when controls are hosted outside the visible bounds of the Panel. Scroll bars are displayed when this property is set to True and are not displayed when it is set to False.</i>
<i>BorderStyle</i>	<i>Represents the visual appearance of the Panel border. This property can be set to None, which indicates no border; FixedSingle, which creates a single-line border; or Fixed3D, which creates a border with a three-dimensional appearance.</i>

## The TableLayoutPanel Control

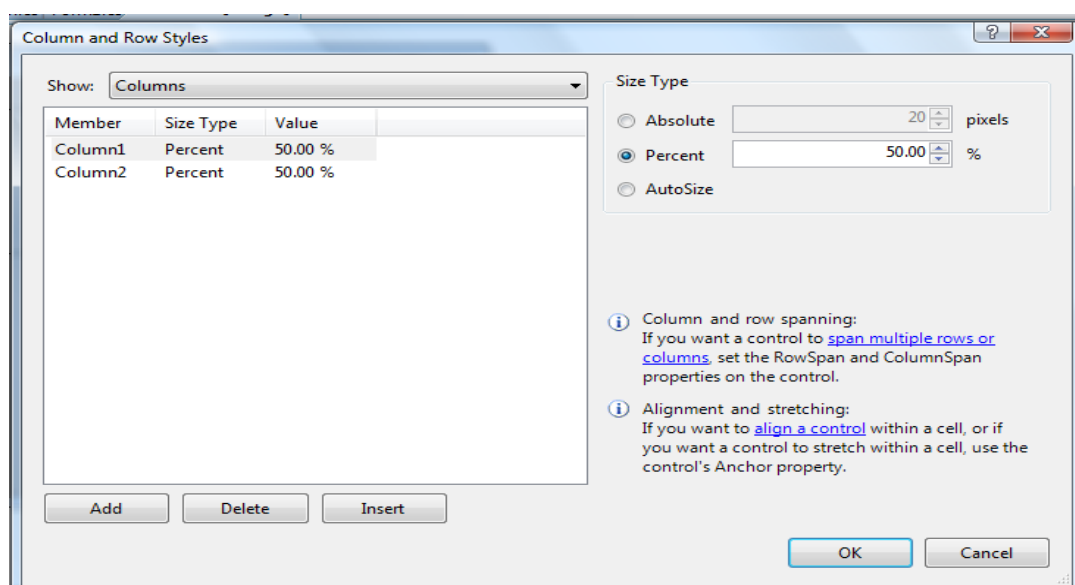
The TableLayoutPanel control is a specialized panel that aids in the design and layout of the UI. The TableLayoutPanel is essentially a table that provides cells for the individual hosting of controls. Like other panels, it is a scrollable container that provides scroll bars when the AutoScroll property is set to True.





At design time the `TableLayoutPanel` appears on the form as a table of individual cells. You can drag controls from the Toolbox into each of the cells. Generally, only one control can be hosted in a single cell. However, for complicated UI designs, you can nest other container controls inside `TableLayoutPanel` cells, each of which can host multiple controls.

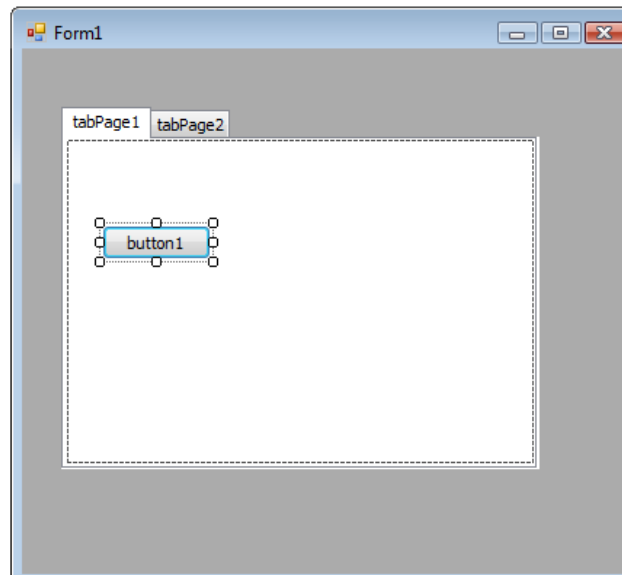
The `ColumnStyle` and `RowStyle` collections manage the columns and rows of the `TableLayoutPanel` control. At design time you can set the styles of the rows and columns by choosing the `ColumnStyles` or `RowStyles` collection in the Property Grid and launching the Columns And Rows Styles editor.



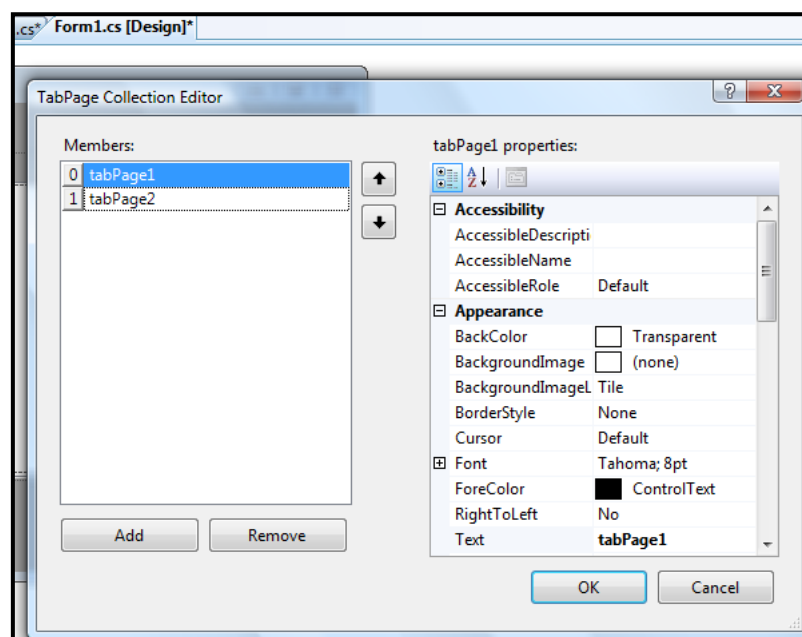
You can alter column and row size styles with this editor. Column and row size styles can be set to `Absolute`, which indicates a fixed size in pixels, or they can be set to `Relative`, which indicates a percentage of the size of all columns or rows whose style is set to `Relative`. Columns and rows can also be set to `AutoSize`. When set to this value, the columns and rows will automatically adjust to the correct size.

## The TabControl Control

The `TabControl` control enables you to group sets of controls in tabs, rather like files in a filing cabinet or dividers in a notebook. For example, you might create property pages for an application in which each page represents the properties of a specific component. The `TabControl` serves as a host for one or more `TabPage` controls, which themselves contain controls. The user can switch between tab pages (and the controls contained therein) by clicking the tabs on the `TabControl`.



The most important property of the TabControl is the TabPages property. TabPage controls are specialized container controls that are hosted only inside TabControl controls. Each TabPage has its own set of properties, and you can access these properties by editing the TabPages property at design time. This launches the TabPage Collection Editor, as shown in the following Figure :



Individual TabPage controls are a lot like Panel controls. They are scrollable controls and will generate scroll bars as needed if the AutoScroll property is set to True. Individual TabPage controls also have a Text property, which represents the text shown in the tab that represents this page in the TabControl. Also, like Panel controls,

TabPage has a `BorderStyle` property that can be set to `None`, `FixedSingle`, or `Fixed3D`, with results similar to those in the `Panel` control.

The `TabControl` has several properties that you can use to customize the look and feel of the control. The `Appearance` property controls how the tabs look. This property can be set to `Normal`, `Buttons`, or `FlatButtons`, each of which generates a different visual style. The `Alignment` property determines whether the tabs appear on the `Top`, `Bottom`, `Left`, or `Right` of the `TabControl`. The `TabControl` also has a property called `Multiline`, which indicates if more than one row of tabs is allowed. When it is set to `True`, multiple rows of tabs are supported. When it is set to `False`, only a single row of tabs is allowed. Important properties of the `TabControl` control and `TabPage` control are shown in Table 2-4 and Table 2-5, respectively.

TABLE 2-4 Important Properties of the `TabControl` Control

Property	Description
<i>Appearance</i>	<i>Determines the visual style of the TabControl</i>
<i>Alignment</i>	<i>Determines whether the tabs appear on the Top, Bottom, Left, or Right of the TabControl</i>
<i>Multiline</i>	<i>Determines whether more than one row of tabs is allowed on the TabControl</i>
<i>TabPages</i>	<i>Represents the collection of TabPage controls hosted by the TabControl</i>

TABLE 2-5 Important Properties of the `TabPage` Control

Property	Description
<i>AutoScroll</i>	<i>Determines if the TabPage will display scroll bars when controls are hosted outside the visible bounds of the Panel. Scroll bars are displayed when set to True and are not displayed when set to False.</i>
<i>BorderStyle</i>	<i>Represents the visual appearance of the TabPage border. It can be set to None, which indicates no border; FixedSingle, which creates a singleline border; or Fixed3D, which creates a border with a three-dimensional appearance.</i>
<i>Text</i>	<i>Represents the text displayed on the tab in the TabControl that represents this TabPage.</i>