

Table of Contents

[Windows Forms Controls](#)

[Arranging Controls on Windows Forms](#)

[How to: Align Multiple Controls on Windows Forms](#)

[How to: Anchor Controls on Windows Forms](#)

[How to: Copy Controls Between Windows Forms](#)

[How to: Dock Controls on Windows Forms](#)

[How to: Layer Objects on Windows Forms](#)

[How to: Lock Controls to Windows Forms](#)

[How to: Position Controls on Windows Forms](#)

[How to: Resize Controls on Windows Forms](#)

[How to: Set Grid Options for All Windows Forms](#)

[How to: Set the Tab Order on Windows Forms](#)

[Walkthrough: Arranging Controls on Windows Forms Using Snaplines](#)

[How to: Arrange Controls with Snaplines and the Grid in Windows Forms](#)

[How to: Reassign Existing Controls to a Different Parent](#)

[Walkthrough: Laying Out Windows Forms Controls with Padding, Margins, and the AutoSize Property](#)

[Putting Controls on Windows Forms](#)

[How to: Add Controls to Windows Forms](#)

[How to: Add Controls Without a User Interface to Windows Forms](#)

[How to: Add to or Remove from a Collection of Controls at Run Time](#)

[Walkthrough: Automatically Populating the Toolbox with Custom Components](#)

[How to: Add ActiveX Controls to Windows Forms](#)

[Considerations When Hosting an ActiveX Control on a Windows Form](#)

[Labeling Individual Windows Forms Controls and Providing Shortcuts to Them](#)

[How to: Create Access Keys for Windows Forms Controls](#)

[How to: Create Access Keys for Windows Forms Controls Using the Designer](#)

[How to: Set the Text Displayed by a Windows Forms Control](#)

[How to: Set the Text Displayed by a Windows Forms Control Using the Designer](#)

[How to: Set the Image Displayed by a Windows Forms Control](#)

[How to: Set the Image Displayed by a Windows Forms Control Using the Designer](#)

[Providing Accessibility Information for Controls on a Windows Form](#)

[Controls to Use on Windows Forms](#)

[Windows Forms Controls by Function](#)

[Controls with Built-In Owner-Drawing Support](#)

[BackgroundWorker Component](#)

[BindingNavigator Control](#)

[BindingSource Component](#)

[Button Control](#)

[CheckBox Control](#)

[CheckedListBox Control](#)

[ColorDialog Component](#)

[ComboBox Control](#)

[ContextMenu Component](#)

[ContextMenuStrip Control](#)

[DataGrid Control](#)

[DataGridView Control](#)

[DateTimePicker Control](#)

[Dialog-Box Controls and Components](#)

[DomainUpDown Control](#)

[ErrorProvider Component](#)

[FileDialog Class](#)

[FlowLayoutPanel Control](#)

[FolderBrowserDialog Component](#)

[FontDialog Component](#)

[GroupBox Control](#)

[HelpProvider Component](#)

[HScrollBar and VScrollBar Controls](#)

[ImageList Component](#)

[Label Control](#)

[LinkLabel Control](#)

ListBox Control
ListView Control
MainMenu Component
MaskedTextBox Control
MenuStrip Control
MonthCalendar Control
NotifyIcon Component
NumericUpDown Control
OpenFileDialog Component
PageSetupDialog Component
Panel Control
PictureBox Control
PrintDialog Component
PrintDocument Component
PrintPreviewControl Control
PrintPreviewDialog Control
ProgressBar Control
RadioButton Control
RichTextBox Control
SaveFileDialog Component
SoundPlayer Class
SplitContainer Control
Splitter Control
StatusBar Control
StatusStrip Control
TabControl Control
TableLayoutPanel Control
TextBox Control
Timer Component
ToolBar Control
ToolStrip Control
ToolStripContainer Control

[ToolStripPanel Control](#)

[ToolStripProgressBar Control](#)

[ToolStripStatusLabel Control](#)

[ToolTip Component](#)

[TrackBar Control](#)

[TreeView Control](#)

[WebBrowser Control](#)

[Windows Forms Controls Used to List Options](#)

[Developing Custom Windows Forms Controls with the .NET Framework](#)

[Overview of Using Controls in Windows Forms](#)

[Varieties of Custom Controls](#)

[Windows Forms Control Development Basics](#)

[Properties in Windows Forms Controls](#)

[Events in Windows Forms Controls](#)

[Attributes in Windows Forms Controls](#)

[Custom Control Painting and Rendering](#)

[Layout in Windows Forms Controls](#)

[Multithreading in Windows Forms Controls](#)

[Windows Forms Controls in the .NET Framework by Function](#)

[Developing Windows Forms Controls at Design Time](#)

[Walkthrough: Authoring a Composite Control with Visual Basic](#)

[Walkthrough: Authoring a Composite Control with Visual C#](#)

[Walkthrough: Inheriting from a Windows Forms Control with Visual Basic](#)

[Walkthrough: Inheriting from a Windows Forms Control with Visual C#](#)

[Walkthrough: Performing Common Tasks Using Smart Tags on Windows Forms Controls](#)

[Walkthrough: Serializing Collections of Standard Types with the DesignerSerializationVisibilityAttribute](#)

[Walkthrough: Debugging Custom Windows Forms Controls at Design Time](#)

[Walkthrough: Creating a Windows Forms Control That Takes Advantage of Visual Studio Design-Time Features](#)

[How to: Author Controls for Windows Forms](#)

[How to: Author Composite Controls](#)

[How to: Inherit from the UserControl Class](#)

[How to: Inherit from Existing Windows Forms Controls](#)

[How to: Inherit from the Control Class](#)

[How to: Align a Control to the Edges of Forms at Design Time](#)

[How to: Display a Control in the Choose Toolbox Items Dialog Box](#)

[How to: Provide a Toolbox Bitmap for a Control](#)

[How to: Test the Run-Time Behavior of a UserControl](#)

[Design-Time Errors in the Windows Forms Designer](#)

[Troubleshooting Control and Component Authoring](#)

Windows Forms Controls

5/4/2018 • 1 min to read • [Edit Online](#)

As you design and modify the user interface of your Windows Forms applications, you will need to add, align, and position controls. Controls are objects that are contained within form objects. Each type of control has its own set of properties, methods, and events that make it suitable for a particular purpose. You can manipulate controls in the designer and write code to add controls dynamically at run time.

In This Section

[Putting Controls on Windows Forms](#)

Provides links related to putting controls on forms.

[Arranging Controls on Windows Forms](#)

Provides links related to arranging controls on forms.

[Labeling Individual Windows Forms Controls and Providing Shortcuts to Them](#)

Describes the uses of keyboard shortcuts, text labels on controls, and modifier keys.

[Controls to Use on Windows Forms](#)

Lists the controls that work with Windows Forms, and basic things you can accomplish with each control.

[Developing Custom Windows Forms Controls with the .NET Framework](#)

Provides background information and samples to help users develop custom Windows Forms controls.

[Developing Windows Forms Controls at Design Time](#)

Describes techniques for creating custom controls through design and inheritance.

Related Sections

[Client Applications](#)

Provides an overview of developing Windows-based applications.

[Windows Forms Walkthroughs](#)

Lists walkthrough topics about Windows Forms and controls.

Arranging Controls on Windows Forms

5/4/2018 • 2 min to read • [Edit Online](#)

By placing and manipulating controls on forms in different ways, you can create user interfaces that are both intuitive and functional for users.

In This Section

[How to: Align Multiple Controls on Windows Forms](#)

Gives directions for lining up the position of a number of controls on your Windows Form.

[How to: Anchor Controls on Windows Forms](#)

Gives directions for setting controls to resize dynamically at run time.

[How to: Copy Controls Between Windows Forms](#)

Gives directions for duplicating controls between forms.

[How to: Dock Controls on Windows Forms](#)

Gives directions for making controls "stick" to the side(s) of a form.

[How to: Layer Objects on Windows Forms](#)

Gives directions for establishing which controls are on top relative to the z-axis (z-order).

[How to: Lock Controls to Windows Forms](#)

Gives directions for fastening controls permanently to the form.

[How to: Position Controls on Windows Forms](#)

Gives directions for setting the coordinates of the controls on a form.

[How to: Resize Controls on Windows Forms](#)

Gives directions for setting the size of controls on a form.

[How to: Set Grid Options for All Windows Forms](#)

Gives directions for calibrating the size of the grid that covers a form.

[How to: Set the Tab Order on Windows Forms](#)

Gives directions for regulating the order in which controls will have focus when the user presses TAB.

[How to: Arrange Controls with Snaplines and the Grid in Windows Forms](#)

Gives directions for affixing controls to the grid on a form.

[How to: Reassign Existing Controls to a Different Parent](#)

Gives directions for assigning existing controls to a new parent container.

[Walkthrough: Laying Out Windows Forms Controls with Padding, Margins, and the AutoSize Property](#)

Describes how you can place controls on your forms by using the Margin, Padding, and AutoSize properties within the **Forms Designer**.

[Walkthrough: Arranging Controls on Windows Forms Using Snaplines](#)

Demonstrates the various layout roles fulfilled by snaplines.

Related Sections

[How to: Designate a Windows Forms Button as the Cancel Button Using the Designer](#)

Gives directions for establishing a button as the control to cancel the form.

[How to: Designate a Windows Forms Button as the Accept Button Using the Designer](#)

Gives directions for establishing a button (often an "OK" button) as the "accept input" button when ENTER is pressed regardless of where focus is at the time in the dialog box.

[How to: Group Windows Forms RadioButton Controls to Function as a Set](#)

Gives directions for establishing a set of `RadioButton` controls as being related to one another.

[Windows Forms Controls](#)

Provides general information about controls.

How to: Align Multiple Controls on Windows Forms

5/4/2018 • 1 min to read • [Edit Online](#)

To standardize the layout of the user interface (UI) of your Windows-based application, you can position groups of controls with a single command.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To align multiple controls on a form

1. Open the form containing the controls you want to position in the **Windows Forms Designer**.
2. Select the controls you want to align so that the first control you select is the primary control to which the others should be aligned.
3. On the **Format** menu, point to **Align**, and then click one of the seven choices available.

See Also

[Windows Forms Controls](#)

[How to: Add Controls to Windows Forms](#)

[Arranging Controls on Windows Forms](#)

[Controls to Use on Windows Forms](#)

[Windows Forms Controls by Function](#)

[Walkthrough: Arranging Controls on Windows Forms Using Snaplines](#)

[Walkthrough: Arranging Controls on Windows Forms Using a TableLayoutPanel](#)

[Walkthrough: Arranging Controls on Windows Forms Using a FlowLayoutPanel](#)

[How to: Reassign Existing Controls to a Different Parent](#)

How to: Anchor Controls on Windows Forms

5/4/2018 • 2 min to read • [Edit Online](#)

If you are designing a form that the user can resize at run time, the controls on your form should resize and reposition properly. To resize controls dynamically with the form, you can use the [Anchor](#) property of Windows Forms controls. The [Anchor](#) property defines an anchor position for the control. When a control is anchored to a form and the form is resized, the control maintains the distance between the control and the anchor positions. For example, if you have a [TextBox](#) control that is anchored to the left, right, and bottom edges of the form, as the form is resized, the [TextBox](#) control resizes horizontally so that it maintains the same distance from the right and left sides of the form. In addition, the control positions itself vertically so that its location is always the same distance from the bottom edge of the form. If a control is not anchored and the form is resized, the position of the control relative to the edges of the form is changed.

The [Anchor](#) property interacts with the [AutoSize](#) property. For more information, see [AutoSize Property Overview](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To anchor a control on a form

1. Select the control you want to anchor.

NOTE

You can anchor multiple controls simultaneously by pressing the CTRL key, clicking each control to select it, and then following the rest of this procedure.

2. In the **Properties** window, click the arrow to the right of the [Anchor](#) property.

An editor is displayed that shows a cross.

3. To set an anchor, click the top, left, right, or bottom section of the cross.

Controls are anchored to the top and left by default.

4. To clear a side of the control that has been anchored, click that arm of the cross.

5. To close the [Anchor](#) property editor, click the [Anchor](#) property name again.

When your form is displayed at run time, the control resizes to remain positioned at the same distance from the edge of the form. The distance from the anchored edge always remains the same as the distance defined when the control is positioned in the Windows Forms Designer.

NOTE

Certain controls, such as the [ComboBox](#) control, have a limit to their height. Anchoring the control to the bottom of its form or container cannot force the control to exceed its height limit.

Inherited controls must be **Protected** to be able to be anchored. To change the access level of a control, set its **Modifiers** property in the **Properties** window.

See Also

[Windows Forms Controls](#)

[Arranging Controls on Windows Forms](#)

[AutoSize Property Overview](#)

[How to: Dock Controls on Windows Forms](#)

[Walkthrough: Arranging Controls on Windows Forms Using a FlowLayoutPanel](#)

[Walkthrough: Arranging Controls on Windows Forms Using a TableLayoutPanel](#)

[Walkthrough: Laying Out Windows Forms Controls with Padding, Margins, and the AutoSize Property](#)

How to: Copy Controls Between Windows Forms

5/4/2018 • 1 min to read • [Edit Online](#)

A control may be copied onto the same form, onto another form within the project, or onto the Clipboard for use in other solutions.

To copy a control

1. Select the control, and from the **Edit** menu choose **Copy**.

This control can now be pasted to any form that accepts that type of control. Additionally, the control has been added to the Clipboard.

See Also

[Windows Forms Controls](#)

[How to: Add Controls to Windows Forms](#)

[How to: Add ActiveX Controls to Windows Forms](#)

[How to: Add Controls Without a User Interface to Windows Forms](#)

[Arranging Controls on Windows Forms](#)

[How to: Set the Text Displayed by a Windows Forms Control](#)

[Putting Controls on Windows Forms](#)

[Labeling Individual Windows Forms Controls and Providing Shortcuts to Them](#)

[Controls to Use on Windows Forms](#)

[Windows Forms Controls by Function](#)

How to: Dock Controls on Windows Forms

5/4/2018 • 1 min to read • [Edit Online](#)

You can dock controls to the edges of your form or have them fill the control's container (either a form or a container control). For example, Windows Explorer docks its [TreeView](#) control to the left side of the window and its [ListView](#) control to the right side of the window. Use the [Dock](#) property for all visible Windows Forms controls to define the docking mode.

NOTE

Controls are docked in reverse z-order.

The [Dock](#) property interacts with the [AutoSize](#) property. For more information, see [AutoSize Property Overview](#).

To dock a control

1. Select the control that you want to dock.
2. In the Properties window, click the arrow to the right of the [Dock](#) property.

An editor is displayed that shows a series of boxes representing the edges and the center of the form.

3. Click the button that represents the edge of the form where you want to dock the control. To fill the contents of the control's form or container control, click the center box. Click **(none)** to disable docking.

The control is automatically resized to fit the boundaries of the docked edge.

NOTE

Inherited controls must be [Protected](#) to be able to be docked. To change the access level of a control, set its **Modifier** property in the Properties window.

See Also

[Windows Forms Controls](#)

[Arranging Controls on Windows Forms](#)

[Labeling Individual Windows Forms Controls and Providing Shortcuts to Them](#)

[Controls to Use on Windows Forms](#)

[Windows Forms Controls by Function](#)

[How to: Anchor and Dock Child Controls in a FlowLayoutPanel Control](#)

[How to: Anchor and Dock Child Controls in a TableLayoutPanel Control](#)

[AutoSize Property Overview](#)

[How to: Anchor Controls on Windows Forms](#)

How to: Layer Objects on Windows Forms

5/4/2018 • 1 min to read • [Edit Online](#)

When you create a complex user interface, or work with a multiple document interface (MDI) form, you will often want to layer both controls and child forms to create more complex user interfaces (UI). To move and keep track of controls and windows within the context of a group, you manipulate their z-order. *Z-order* is the visual layering of controls on a form along the form's z-axis (depth). The window at the top of the z-order overlaps all other windows. All other windows overlap the window at the bottom of the z-order.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To layer controls at design time

1. Select a control that you want to layer.
2. On the **Format** menu, point to **Order**, and then click **Bring To Front** or **Send To Back**.

To layer controls programmatically

- Use the [BringToFront](#) and [SendToBack](#) methods to manipulate the z-order of the controls.

For example, if a [TextBox](#) control, `txtFirstName`, is underneath another control and you want to have it on top, use the following code:

```
txtFirstName.BringToFront()
```

```
txtFirstName.BringToFront();
```

```
txtFirstName->BringToFront();
```

NOTE

Windows Forms supports *control containment*. Control containment involves placing a number of controls within a containing control, such as a number of [RadioButton](#) controls within a [GroupBox](#) control. You can then layer the controls within the containing control. Moving the group box moves the controls as well, because they are contained inside it.

See Also

[Windows Forms Controls](#)

[Arranging Controls on Windows Forms](#)

[Labeling Individual Windows Forms Controls and Providing Shortcuts to Them](#)

[Controls to Use on Windows Forms](#)

[Windows Forms Controls by Function](#)

How to: Lock Controls to Windows Forms

5/4/2018 • 1 min to read • [Edit Online](#)

When you design the user interface (UI) of your Windows application, you can lock the controls once they are positioned correctly, so that you do not inadvertently move or resize them when setting other properties.

Additionally, you can lock and unlock all the controls on the form at once, which is helpful for forms with many controls, or you can unlock individual controls. Once you have placed all the controls where you want them on the form, lock them all in place to prevent erroneous movement.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To lock a control

1. In the **Properties** window, click the **Locked** property and select `true`. (Double-clicking the name toggles the property setting.)

Alternatively, right-click the control and choose **Lock Controls**.

NOTE

Locking controls prevents them from being dragged to a new size or location on the design surface. However, you can still change the size or location of controls by means of the **Properties** window or in code.

To lock all the controls on a form

1. From the **Format** menu, choose **Lock Controls**.

NOTE

This command locks the form's size as well, because a form is a control.

To unlock all locked controls on a form

1. From the **Format** menu, choose **Lock Controls**.

All previously locked controls on the form are now unlocked.

To unlock locked controls individually

1. In the **Properties** window, click the **Locked** property and select `false`. (Double-clicking the name toggles the property setting.)

See Also

[Windows Forms Controls](#)

[Arranging Controls on Windows Forms](#)

[Labeling Individual Windows Forms Controls and Providing Shortcuts to Them](#)

[Controls to Use on Windows Forms](#)

Windows Forms Controls by Function

How to: Position Controls on Windows Forms

5/4/2018 • 1 min to read • [Edit Online](#)

To position controls, use the Windows Forms Designer, or specify the [Location](#) property.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To position a control on the design surface of the Windows Forms Designer

- Drag the control to the appropriate location with the mouse.

NOTE

Select the control and move it with the ARROW keys to position it more precisely. Also, *snaplines* assist you in placing controls precisely on your form. For more information, see [Walkthrough: Arranging Controls on Windows Forms Using Snaplines](#).

To position a control using the Properties window

1. Click the control you want to position.
2. In the **Properties** window, type values for the [Location](#) property, separated by a comma, to position the control within its container.

The first number (X) is the distance from the left border of the container; the second number (Y) is the distance from the upper border of the container area, measured in pixels.

NOTE

You can expand the [Location](#) property to type the **X** and **Y** values individually.

To position a control programmatically

1. Set the [Location](#) property of the control to a [Point](#).

```
Button1.Location = New Point(100, 100)
```

```
button1.Location = new Point(100, 100);
```

```
button1->Location = Point(100, 100);
```

2. Change the X coordinate of the control's location using the [Left](#) subproperty.

```
Button1.Left = 300
```

```
button1.Left = 300;
```

```
button1->Left = 300;
```

To increment a control's location programmatically

1. Set the [Left](#) subproperty to increment the X coordinate of the control.

```
Button1.Left += 200
```

```
button1.Left += 200;
```

```
button1->Left += 200;
```

NOTE

Use the [Location](#) property to set a control's X and Y positions simultaneously. To set a position individually, use the control's [Left \(X\)](#) or [Top \(Y\)](#) subproperty. Do not try to implicitly set the X and Y coordinates of the [Point](#) structure that represents the button's location, because this structure contains a copy of the button's coordinates.

See Also

[Windows Forms Controls](#)

[Walkthrough: Arranging Controls on Windows Forms Using Snaplines](#)

[Walkthrough: Arranging Controls on Windows Forms Using a TableLayoutPanel](#)

[Walkthrough: Arranging Controls on Windows Forms Using a FlowLayoutPanel](#)

[Arranging Controls on Windows Forms](#)

[Labeling Individual Windows Forms Controls and Providing Shortcuts to Them](#)

[Controls to Use on Windows Forms](#)

[Windows Forms Controls by Function](#)

[How to: Set the Screen Location of Windows Forms](#)

How to: Resize Controls on Windows Forms

5/4/2018 • 1 min to read • [Edit Online](#)

You can resize individual controls, and you can resize multiple controls of the same or different kind, such as [Button](#) and [GroupBox](#) controls.

To resize a control

1. Click the control to be resized and drag one of the eight sizing handles.

NOTE

Select the control and press the ARROW keys while holding down the SHIFT key to resize the control one pixel at a time. Press the DOWN or RIGHT arrow keys while holding down the SHIFT and CTRL keys to resize the control in large increments.

To resize multiple controls on a form

1. Hold down the CTRL or SHIFT key and select the controls you want to resize. The size of the first control you select is used for the other controls.
2. On the **Format** menu, choose **Make Same Size**, and select one of the four options. The first three commands change the dimensions of the controls to match the first-selected control.

See Also

[Windows Forms Controls](#)

[Arranging Controls on Windows Forms](#)

[Labeling Individual Windows Forms Controls and Providing Shortcuts to Them](#)

[Controls to Use on Windows Forms](#)

[Windows Forms Controls by Function](#)

[How to: Resize Windows Forms Using the Designer](#)

How to: Set Grid Options for All Windows Forms

5/4/2018 • 1 min to read • [Edit Online](#)

As you become used to working in the Visual Studio development environment, you can set preferences for all the forms and projects you work with in the Windows Forms Designer.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To set global Windows Forms options

1. From the **Tools** menu, select **Options**.
2. In the left pane of the **Options** dialog box, click **Windows Forms Designer**.

In the right pane, under the **Layout Settings** heading, you can set the default grid settings for all the new forms you create. These settings include the grid size, whether controls snap to it, and whether it is on by default. In addition, you can select between **SnapToGrid** and **SnapLines** layout modes. For more information on snaplines, see [Walkthrough: Arranging Controls on Windows Forms Using Snaplines](#).

See Also

[Walkthrough: Arranging Controls on Windows Forms Using Snaplines](#)

[General, Windows Forms Designer, Options Dialog Box](#)

[Windows Forms Controls](#)

[How to: Add Controls to Windows Forms](#)

[Arranging Controls on Windows Forms](#)

[How to: Set the Tab Order on Windows Forms](#)

[How to: Set the Text Displayed by a Windows Forms Control](#)

[Putting Controls on Windows Forms](#)

[Controls to Use on Windows Forms](#)

[Windows Forms Controls by Function](#)

How to: Set the Tab Order on Windows Forms

5/4/2018 • 2 min to read • [Edit Online](#)

The tab order is the order in which a user moves focus from one control to another by pressing the TAB key. Each form has its own tab order. By default, the tab order is the same as the order in which you created the controls. Tab-order numbering begins with zero.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To set the tab order of a control

1. On the **View** menu, click **Tab Order**.

This activates the tab-order selection mode on the form. A number (representing the **TabIndex** property) appears in the upper-left corner of each control.

2. Click the controls sequentially to establish the tab order you want.

NOTE

A control's place within the tab order can be set to any value greater than or equal to 0. When duplicates occur, the z-order of the two controls is evaluated and the control on top is tabbed to first. (The z-order is the visual layering of controls on a form along the form's z-axis [depth]. The z-order determines which controls are in front of other controls.) For more information on z-order, see [Layering Objects on Windows Forms](#).

3. When you have finished, click **Tab Order** on the **View** menu again to leave tab order mode.

NOTE

Controls that cannot get the focus, as well as disabled and invisible controls, do not have a **TabIndex** property and are not included in the tab order. As a user presses the TAB key, these controls are skipped.

Alternatively, tab order can be set in the Properties window using the **TabIndex** property. The **TabIndex** property of a control determines where it is positioned in the tab order. By default, the first control drawn has a **TabIndex** value of 0, the second has a **TabIndex** of 1, and so on.

Additionally, by default, a **GroupBox** control has its own **TabIndex** value, which is a whole number. A **GroupBox** control itself cannot have focus at run time. Thus, each control within a **GroupBox** has its own decimal **TabIndex** value, beginning with .0. Naturally, as the **TabIndex** of a **GroupBox** control is incremented, the controls within it will be incremented accordingly. If you changed a **TabIndex** value from 5 to 6, the **TabIndex** value of the first control in its group automatically changes to 6.0, and so on.

Finally, any control of the many on your form can be skipped in the tab order. Usually, pressing TAB successively at run time selects each control in the tab order. By turning off the **TabStop** property, you can make a control be passed over in the tab order of the form.

To remove a control from the tab order

1. Set the control's **TabStop** property to `false` in the Properties window.

A control whose **TabStop** property has been set to `false` still maintains its position in the tab order, even though the control is skipped when you cycle through the controls with the TAB key.

NOTE

A radio button group has a single tab stop at run time. The selected button (that is, the button with its **Checked** property set to `true`) has its **TabStop** property automatically set to `true`, while the other buttons have their **TabStop** property set to `false`. For more information about grouping **RadioButton** controls, see [Grouping Windows Forms RadioButton Controls to Function as a Set](#).

See Also

[Windows Forms Controls](#)

[Arranging Controls on Windows Forms](#)

[Controls to Use on Windows Forms](#)

[Windows Forms Controls by Function](#)

Walkthrough: Arranging Controls on Windows Forms Using Snaplines

5/4/2018 • 10 min to read • [Edit Online](#)

Precise placement of controls on your form is a high priority for many applications. The Windows Forms Designer gives you many layout tools to accomplish this. One of the most important is the [SnapLine](#) feature.

Snaplines show you precisely where to line up controls with other controls. They also show you the recommended distances for margins between controls, as specified by the Windows User Interface Guidelines. For details, see [User Interface Design and Development](#).

Snaplines make it easy to align your controls, for crisp, professional appearance and behavior (look and feel).

Tasks illustrated in this walkthrough include:

- Creating a Windows Forms project
- Spacing and Aligning Controls Using Snaplines
- Aligning to Form and Container Margins
- Aligning to Grouped Controls
- Using Snaplines to Place a Control by Outlining Its Size
- Using Snaplines When Dragging a Control from the Toolbox
- Resizing Controls Using Snaplines
- Aligning a Label to a Control's Text
- Using Snaplines with Keyboard Navigation
- Snaplines and Layout Panels
- Disabling Snaplines

When you are finished, you will have an understanding of the layout role played by the snaplines feature.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

Creating the Project

The first step is to create the project and set up the form.

To create the project

1. Create a Windows-based application project called "SnaplineExample". For details, see [How to: Create a Windows Application Project](#).
2. Select the form in the Forms Designer.

Spacing and Aligning Controls Using Snaplines

Snaplines give you an accurate and intuitive way to align controls on your form. They appear when you are moving a selected control or controls near a position that would align with another control or set of controls. Your selection will "snap" to the suggested position as you move it past the other controls.

To arrange controls using snaplines

1. Drag a **Button** control from the **Toolbox** onto your form.
2. Move the **Button** control to the lower-right corner of the form. Note the snaplines that appear as the **Button** control approaches the bottom and right borders of the form. These snaplines display the recommended distance between the borders of the control and the form.
3. Move the **Button** control around the borders of the form and note where the snaplines appear. When you are finished, move the **Button** control near the center of the form.
4. Drag another **Button** control from the **Toolbox** onto your form.
5. Move the second **Button** control until it is nearly level with the first. Note the snapline that appears at the text baseline of both buttons, and note that the control you are moving snaps to a position that is exactly level with the other control.
6. Move the second **Button** control until it is positioned directly above the first. Note the snaplines that appear along the left and right edges of both buttons, and note that the control you are moving snaps to a position that is exactly aligned with the other control.
7. Select one of the **Button** controls and move it close to the other, until they are almost touching. Note the snapline that appears between them. This distance is the recommended distance between the borders of the controls. Also note that the control you are moving snaps to this position.
8. Drag two **Panel** controls from the **Toolbox** onto your form.
9. Move one of the **Panel** controls until it is nearly level with the first. Note the snaplines that appear along the top and bottom edges of both controls, and note that the control you are moving snaps to a position that is exactly level with the other control.

Aligning to Form and Container Margins

Snaplines help you to align your controls to form and container margins in a consistent manner.

To align controls to form and container margins

1. Select one of the **Button** controls and move it close to the right border of the form until a snapline appears. The snapline's distance from the right border is the sum of the control's **Margin** property and the form's **Padding** property values.

NOTE

If the form's **Padding** property is set to 0,0,0,0, the Windows Forms Designer gives the form a shadowed **Padding** value of 9,9,9,9. To override this behavior, assign a value other than 0,0,0,0.

1. Change the value of the **Button** control's **Margin** property by expanding the **Margin** entry in the **Properties** window and setting the **All** property to 0. For details, see [Walkthrough: Laying Out Windows Forms Controls with Padding, Margins, and the AutoSize Property](#).
2. Move the **Button** control close to the right border of the form until a snapline appears. This distance is now given by the value of the form's **Padding** property.
3. Drag a **GroupBox** control from the **Toolbox** onto your form.

4. Change the value of the **GroupBox** control's **Padding** property by expanding the **Padding** entry in the **Properties** window and setting the **All** property to 10.
5. Drag a **Button** control from the **Toolbox** into the **GroupBox** control.
6. Move the **Button** control close to the right border of the **GroupBox** control until a snapline appears. Move the **Button** control within the **GroupBox** control and note where the snaplines appear.

Aligning to Grouped Controls

You can use snaplines to align grouped controls as well as controls within a **GroupBox** control.

To align to grouped controls

1. Select two of the controls on your form. Move the selection around and note the snaplines that appear between your selection and the other controls.
2. Drag a **GroupBox** control from the **Toolbox** onto your form.
3. Drag a **Button** control from the **Toolbox** into the **GroupBox** control.
4. Select one of the **Button** controls and move it around the **GroupBox** control. Note the snaplines that appear at the edges of the **GroupBox** control. Also note the snaplines that appear at the edges of the **Button** control that is contained by the **GroupBox** control. Controls that are children of a container control also support snaplines.

Using Snaplines to Place a Control by Outlining Its Size

Snaplines help you align controls when you first place them on a form.

To use snaplines to place a control by outlining its size

1. In the **Toolbox**, click the **Button** control icon. Do not drag it onto the form.
2. Move the mouse pointer over the form's design surface. Note that the pointer changes to a crosshair with the **Button** control icon attached. Also note the snaplines that appear to suggest aligned positions for the **Button** control.
3. Click and hold the mouse button.
4. Drag the mouse pointer around the form. Note that an outline is drawn, indicating the position and the size of the control.
5. Drag the pointer until it aligns with another control on the form. Note that a snapline appears to indicate alignment.
6. Release the mouse button. The control is created at the position and size indicated by the outline.

Using Snaplines When Dragging a Control from the Toolbox

Snaplines help you align controls when you drag them from the **Toolbox** onto your form.

To use snaplines when dragging a control from the Toolbox

1. Drag a **Button** control from the **Toolbox** onto your form, but do not release the mouse button.
2. Move the mouse pointer over the form's design surface. Note that the pointer changes to indicate the position at which the new **Button** control will be created.
3. Drag the mouse pointer around the form. Note the snaplines that appear to suggest aligned positions for the **Button** control. Find a position that is aligned with other controls.
4. Release the mouse button. The control is created at the position indicated by the snaplines.

Resizing Controls Using Snaplines

Snaplines help you align controls as you resize them.

To resize a control using snaplines

1. Drag a **Button** control from the **Toolbox** onto your form.
2. Resize the **Button** control by grabbing one of the corner sizing handles and dragging. For details, see [How to: Resize Controls on Windows Forms](#).
3. Drag the sizing handle until one of the **Button** control's borders is aligned with another control. Note that a snapline appears. Also note that the sizing handle snaps to the position indicated by the snapline.
4. Resize the **Button** control in different directions and align the sizing handle to different controls. Note how the snaplines appear in various orientations to indicate alignment.

Aligning a Label to a Control's Text

Some controls offer a snapline for aligning other controls to displayed text.

To align a label to a control's text

1. Drag a **TextBox** control from the **Toolbox** onto your form. When you drop the **TextBox** control onto the form, click the smart-tag glyph and select the **Set text to textBox1** option. For details, see [Walkthrough: Performing Common Tasks Using Smart Tags on Windows Forms Controls](#).
2. Drag a **Label** control from the **Toolbox** onto your form.
3. Change the value of the **Label** control's **AutoSize** property to `true`. Note that the control's borders are adjusted to fit the display text.
4. Move the **Label** control to the left of the **TextBox** control, so it is aligned with the bottom edge of the **TextBox** control. Note the snapline that appears along the bottom edges of the two controls.
5. Move the **Label** control slightly upward, until the **Label** text and the **TextBox** text are aligned. Note the differently styled snapline that appears, indicating when the text fields of both controls are aligned.

Using Snaplines with Keyboard Navigation

Snaplines help you align controls when you are arranging them using the keyboard's arrow keys.

To use snaplines with keyboard navigation

1. Drag a **Button** control from the **Toolbox** onto your form. Place it in the upper-left corner of the form.
2. Press **CTRL+DOWN ARROW**. Note that the control moves down the form to the first available horizontal alignment position.
3. Press **CTRL+DOWN ARROW** until the control reaches the bottom of the form. Note the positions it occupies as it moves down the form.
4. Press **CTRL+RIGHT ARROW**. Note that the control moves across the form to the first available vertical alignment position.
5. Press **CTRL+RIGHT ARROW** until the control reaches the side of the form. Note the positions it occupies as it moves across the form.
6. Move the control around the form with a combination of arrow keys. Note the positions the control occupies and the snaplines that accompany them.
7. Press **SHIFT+any arrow key** to resize the **Button** control by increments of one pixel.

8. Press CTRL+SHIFT+any arrow key to resize the **Button** control in snapline increments.

Snaplines and Layout Panels

Snaplines are disabled within layout panels.

To selectively disable snaplines

1. Drag a **TableLayoutPanel** control from the **Toolbox** onto your form.
2. Double-click the **Button** control icon in the **Toolbox**. Note that a new button control appears in the **TableLayoutPanel** control's first cell.
3. Double-click the **Button** control icon in the **Toolbox** twice more. This leaves one empty cell in the **TableLayoutPanel** control.
4. Drag a **Button** control from the **Toolbox** into the empty cell of the **TableLayoutPanel** control. Note that no snaplines appear.
5. Drag the **Button** control out of the **TableLayoutPanel** control and move it around the **TableLayoutPanel** control. Note that snaplines appear again.

Disabling Snaplines

Snaplines are turned on by default. You can disable snaplines selectively, or you can disable them in the design environment.

To selectively disable snaplines

- Press the ALT key and while moving a control around the form.

Note that no snaplines appear and the control does not snap to any potential alignment positions.

To disable snaplines in the design environment

1. From the **Tools** menu, open the **Options** dialog box. Open the Windows Forms Designer dialog box. For details, see [General, Windows Forms Designer, Options Dialog Box](#).
2. Select the **General** node. In the **Layout Mode** section, change the selection from **SnapLines** to **SnapToGrid**.
3. Click OK to apply the setting.
4. Select a control on your form and move it around the other controls. Note that snaplines do not appear.

Next Steps

Snaplines offer an intuitive means of aligning controls on your form. Suggestions for more exploration include:

- Try nesting a **GroupBox** control within another **GroupBox** control. Place a **Button** control within the child **GroupBox** control, and another within the parent **GroupBox** control. Move the **Button** controls around to see how the snaplines cross container boundaries.
- Create a column of **TextBox** controls and a corresponding column of **Label** controls. Set the value of the **Label** controls' **AutoSize** property to `true`. Use snaplines to move the **Label** controls so their displayed text is aligned with the text in the **TextBox** controls.

For information about Windows user interface design, see the book *Microsoft Windows User Experience, Official Guidelines for User Interface Developers and Designers* Redmond, WA: Microsoft Press, 1999. (ISBN: 0-7356-0566-1).

See Also

[SnapLine](#)

[Walkthrough: Arranging Controls on Windows Forms Using a FlowLayoutPanel](#)

[Walkthrough: Arranging Controls on Windows Forms Using a TableLayoutPanel](#)

[Walkthrough: Laying Out Windows Forms Controls with Padding, Margins, and the AutoSize Property](#)

[Arranging Controls on Windows Forms](#)

How to: Arrange Controls with Snaplines and the Grid in Windows Forms

5/4/2018 • 1 min to read • [Edit Online](#)

Using the layout features of Visual Studio, you can precisely direct where controls are placed on a form. Controls added to a form or moved on a form can be automatically aligned to the rows and columns of the Windows Forms Designer's grid, or you can align controls by using the snaplines feature.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To snap all controls to the grid

- Select the **SnapToGrid** layout mode in the Windows Forms Designer **Options** dialog box.

For more information, see [General, Windows Forms Designer, Options Dialog Box](#). All controls now align themselves along the points on the grid.

You can snap individual controls to the grid by locking them in place. However, while they are locked, they cannot be moved or resized. For more information about locking controls, see [How to: Lock Controls to Windows Forms](#).

To align controls using snaplines

- Select the **SnapLines** layout mode in the Windows Forms Designer **Options** dialog box.

For more information, see [Walkthrough: Arranging Controls on Windows Forms Using Snaplines](#). You can now use snaplines to align and arrange controls on your form.

See Also

[General, Windows Forms Designer, Options Dialog Box](#)

[Walkthrough: Arranging Controls on Windows Forms Using Snaplines](#)

[Windows Forms Controls](#)

[How to: Add Controls to Windows Forms](#)

[Arranging Controls on Windows Forms](#)

[Labeling Individual Windows Forms Controls and Providing Shortcuts to Them](#)

[Controls to Use on Windows Forms](#)

[Windows Forms Controls by Function](#)

How to: Reassign Existing Controls to a Different Parent

5/4/2018 • 1 min to read • [Edit Online](#)

You can assign controls that exist on your form to a new container control.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To reassign existing controls to a different parent

1. Drag three **Button** controls from the **Toolbox** onto the form.

Position them near to each other, but leave them unaligned.

2. In the **Toolbox**, click the **FlowLayoutPanel** control icon.

Do not drag the icon onto the form.

3. Move the mouse pointer close to the three **Button** controls.

The pointer changes to a crosshair with the **FlowLayoutPanel** control icon attached.

4. Click and hold the mouse button.

5. Drag the mouse pointer to draw the outline of the **FlowLayoutPanel** control.

6. Draw the outline around the three **Button** controls.

7. Release the mouse button.

The three **Button** controls are now inserted into the **FlowLayoutPanel** control.

See Also

[FlowLayoutPanel](#)

[TableLayoutPanel](#)

[Arranging Controls on Windows Forms](#)

[Walkthrough: Arranging Controls on Windows Forms Using a TableLayoutPanel](#)

[Walkthrough: Arranging Controls on Windows Forms Using Snaplines](#)

Walkthrough: Laying Out Windows Forms Controls with Padding, Margins, and the AutoSize Property

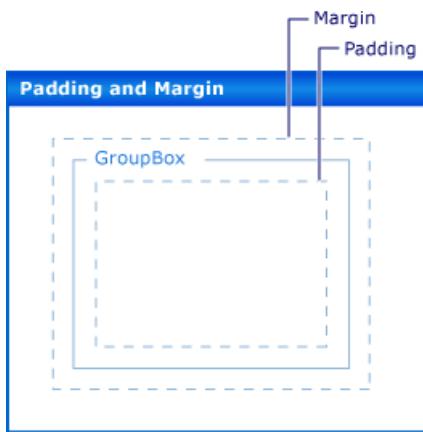
5/4/2018 • 7 min to read • [Edit Online](#)

Precise placement of controls on your form is a high priority for many applications. The **Windows Forms Designer** gives you many layout tools to accomplish this. Three of the most important are the [Margin](#), [Padding](#), and [AutoSize](#) properties, which are present on all Windows Forms controls.

The [Margin](#) property defines the space around the control that keeps other controls a specified distance from the control's borders.

The [Padding](#) property defines the space in the interior of a control that keeps the control's content (for example, the value of its [Text](#) property) a specified distance from the control's borders.

The following illustration shows the [Padding](#) and [Margin](#) properties on a control.



The [AutoSize](#) property tells a control to automatically size itself to its contents. It will not resize itself to be smaller than the value of its original [Size](#) property, and it will account for the value of its [Padding](#) property.

Tasks illustrated in this walkthrough include:

- Creating a Windows Forms project
- Setting Margins for Your Controls
- Setting Padding for Your Controls
- Automatically Sizing Your Controls

When you are finished, you will have an understanding of the role played by these important layout features.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

Prerequisites

In order to complete this walkthrough, you will need:

- Sufficient permissions to be able to create and run Windows Forms application projects on the computer where Visual Studio is installed.

Creating the Project

The first step is to create the project and set up the form.

To create the project

1. Create a **Windows Application** project called `LayoutExample`. For more information, see [How to: Create a Windows Application Project](#).
2. Select the form in the **Windows Forms Designer**.

Setting Margins for Your Controls

You can set the default distance between your controls using the **Margin** property. When you move a control close enough to another control, you will see a snapline that shows the margins for the two controls. The control you are moving will also snap to the distance defined by the margins.

To arrange controls on your form using the Margin property

1. Drag two **Button** controls from the **Toolbox** onto your form.
2. Select one of the **Button** controls and move it close to the other, until they are almost touching.

Observe the snapline that appears between them. This distance is the sum of the two controls' **Margin** values. The control you are moving snaps to this distance. For details, see [Walkthrough: Arranging Controls on Windows Forms Using Snaplines](#).

3. Change the **Margin** property of one of the controls by expanding the **Margin** entry in the **Properties** window and setting the **All** property to 20.
4. Select one of the **Button** controls and move it close to the other.

The snapline defining the sum of the margin values is longer and that the control snaps to a greater distance from the other control.

5. Change the **Margin** property of the selected control by expanding the **Margin** entry in the **Properties** window and setting the **Top** property to 5.
6. Move the selected control below the other control and observe that the snapline is shorter. Move the selected control to the left of the other control and observe that the snapline retains the value observed in step 4.
7. You can set each of the aspects of the **Margin** property, **Left**, **Top**, **Right**, **Bottom**, to different values, or you can set them all to the same value with the **All** property.

Setting Padding for Your Controls

To achieve the precise layout required for your application, your controls will often contain child controls. When you want to specify the proximity of the child control's border to the parent control's border, use the parent control's **Padding** property in conjunction with the child control's **Margin** property. The **Padding** property is also used to control the proximity of a control's content (for example, a **Button** control's **Text** property) to its borders.

To arrange controls on your form using padding

1. Drag a **Button** control from the **Toolbox** onto your form.
2. Change the value of the **Button** control's **AutoSize** property to `true`.
3. Change the **Padding** property by expanding the **Padding** entry in the **Properties** window and setting the

All property to 5.

The control expands to provide room for the new padding.

4. Drag a **GroupBox** control from the **Toolbox** onto your form. Drag a **Button** control from the **Toolbox** into the **GroupBox** control. Position the **Button** control so it is flush with the lower-right corner of the **GroupBox** control.

Observe the snaplines that appear as the **Button** control approaches the bottom and right borders of the **GroupBox** control. These snaplines correspond to the **Margin** property of the **Button**.

5. Change the **GroupBox** control's **Padding** property by expanding the **Padding** entry in the **Properties** window and setting the **All** property to 20.
6. Select the **Button** control within the **GroupBox** control and move it toward the center of the **GroupBox**.

The snaplines appear at a greater distance from the borders of the **GroupBox** control. This distance is the sum of the **Button** control's **Margin** property and the **GroupBox** control's **Padding** property.

Automatically Sizing Your Controls

In some applications, the size of a control will not be the same at run time as it was at design time. The text of a **Button** control, for example, may be taken from a database, and its length will not be known in advance.

When the **AutoSize** property is set to `true`, the control will size itself to its content. For more information, see [AutoSize Property Overview](#).

To arrange controls on your form using the AutoSize property

1. Drag a **Button** control from the **Toolbox** onto your form.
2. Change the value of the **Button** control's **AutoSize** property to `true`.
3. Change the **Button** control's **Text** property to "**This button has a long string for its Text property.**"

When you commit the change, the **Button** control resizes itself to fit the new text.

4. Drag another **Button** control from the **Toolbox** onto your form.
5. Change the **Button** control's **Text** property to "**This button has a long string for its Text property.**"

When you commit the change, the **Button** control does not resize itself, and the text is clipped by the right edge of the control.

6. Change the **Padding** property by expanding the **Padding** entry in the **Properties** window and setting the **All** property to 5.

The text in the control's interior is clipped on all four sides.

7. Change the **Button** control's **AutoSize** property to `true`.

The **Button** control resizes itself to encompass the entire string. Also, padding has been added around the text, causing the **Button** control to expand in all four directions.

8. Drag a **Button** control from the **Toolbox** onto your form. Position it near the lower-right corner of the form.
9. Change the value of the **Button** control's **AutoSize** property to `true`.
10. Set the **Button** control's **Anchor** property to **Right, Bottom**.
11. Change the **Button** control's **Text** property to "**This button has a long string for its Text property.**"

When you commit the change, the [Button](#) control resizes itself toward the left. In general, automatic sizing will increase the size of a control in the direction opposite its [Anchor](#) property setting.

AutoSize and AutoSizeMode Properties

Some controls support the [AutoSizeMode](#) property, which gives you more fine-grained control over the automatic sizing behavior of a control.

To use the AutoSizeMode property

1. Drag a [Panel](#) control from the [Toolbox](#) onto your form.
2. Set the value of the [Panel](#) control's [AutoSize](#) property to `true`.
3. Drag a [Button](#) control from the [Toolbox](#) into the [Panel](#) control.
4. Place the [Button](#) control near the lower-right corner of the [Panel](#) control.
5. Select the [Panel](#) control and grab the lower-right sizing handle. Resize the [Panel](#) control to be larger and smaller.

NOTE

You can freely resize the [Panel](#) control, but you cannot size it smaller than the position of the [Button](#) control's lower-right corner. This behavior is specified by the default value of the [AutoSizeMode](#) property, which is [GrowOnly](#).

6. Set the value of the [Panel](#) control's [AutoSizeMode](#) property to [GrowAndShrink](#).

The [Panel](#) control sizes itself to surround the [Button](#) control. You cannot resize the [Panel](#) control.

7. Drag the [Button](#) control toward the upper-left corner of the [Panel](#) control.

The [Panel](#) control resizes to the [Button](#) control's new position.

Next Steps

There are many other layout features for arranging controls in your Windows Forms applications. Here are some combinations you might try:

- Build a form using a [TableLayoutPanel](#) control. For details, see [Walkthrough: Arranging Controls on Windows Forms Using a TableLayoutPanel](#). Try changing the values of the [TableLayoutPanel](#) control's [Padding](#) property, as well as the [Margin](#) property on its child controls.
- Try the same experiment using a [FlowLayoutPanel](#) control. For details, see [Walkthrough: Arranging Controls on Windows Forms Using a FlowLayoutPanel](#).
- Experiment with docking child controls in a [Panel](#) control. The [Padding](#) property is a more general realization of the [DockPadding](#) property, and you can satisfy yourself that this is the case by putting a child control in a [Panel](#) control and setting the child control's [Dock](#) property to [Fill](#). Set the [Panel](#) control's [Padding](#) property to various values and note the effect.

See Also

[AutoSize](#)

[DockPadding](#)

[Margin](#)

[Padding](#)

[AutoSize Property Overview](#)

[Walkthrough: Arranging Controls on Windows Forms Using a TableLayoutPanel](#)

[Walkthrough: Arranging Controls on Windows Forms Using a FlowLayoutPanel](#)

[Walkthrough: Arranging Controls on Windows Forms Using Snaplines](#)

Putting Controls on Windows Forms

5/4/2018 • 1 min to read • [Edit Online](#)

There are a wide variety of controls that you can put on Windows Forms, depending on the needs of your application.

In This Section

[How to: Add Controls to Windows Forms](#)

Gives directions for attaching controls to your form.

[How to: Add Controls Without a User Interface to Windows Forms](#)

Gives directions for appending controls with no user interface to your application.

[How to: Add to or Remove from a Collection of Controls at Run Time](#)

Explains how to add and remove controls on a panel at run time.

[Walkthrough: Automatically Populating the Toolbox with Custom Components](#)

Demonstrates how you can make a custom component automatically appear in the **Toolbox** once the component is created.

[How to: Add ActiveX Controls to Windows Forms](#)

Gives directions for working with legacy ActiveX controls.

[Considerations When Hosting an ActiveX Control on a Windows Form](#)

Describes things to keep in mind when planning an application that uses ActiveX controls.

Related Sections

[Windows Forms Controls](#)

Provides links to introductory topics on controls and what you can do with them.

[User Input Validation in Windows Forms](#)

Explains the basics of and theory behind validating the contents of Windows Forms controls.

How to: Add Controls to Windows Forms

5/4/2018 • 2 min to read • [Edit Online](#)

Most forms are designed by adding controls to the surface of the form to define a user interface (UI). A *control* is a component on a form used to display information or accept user input. For more information about controls, see [Windows Forms Controls](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To draw a control on a form

1. Open the form. For more information, see [How to: Display Windows Forms in the Designer](#).
2. In the **Toolbox**, click the control you want to add to your form.
3. On the form, click where you want the upper-left corner of the control to be located, and drag to where you want the lower-right corner of the control to be located.

The control is added to the form with the specified location and size.

NOTE

Each control has a default size defined. You can add a control to your form in the control's default size by dragging it from the **Toolbox** to the form.

To drag a control to a form

1. Open the form. For more information, see [How to: Display Windows Forms in the Designer](#).
2. In the **Toolbox**, click the control you want and drag it to your form.

The control is added to the form at the specified location in its default size.

NOTE

You can double-click a control in the **Toolbox** to add it to the upper-left corner of the form in its default size.

You can also add controls dynamically to a form at run time. In the following code example, a **TextBox** control will be added to the form when a **Button** control is clicked.

NOTE

The following procedure requires the existence of a form with a **Button** control, `Button1`, already placed on it.

To add a control to a form programmatically

1. In the method that handles the button's `Click` event within your form's class, insert code similar to the following to add a reference to your control variable, set the control's `Location`, and add the

control.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim MyText As New TextBox()
    MyText.Location = New Point(25, 25)
    Me.Controls.Add(MyText)
End Sub
```

```
private void button1_Click(object sender, System.EventArgs e)
{
    TextBox myText = new TextBox();
    myText.Location = new Point(25,25);
    this.Controls.Add (myText);
}
```

```
private:
System::Void button1_Click(System::Object ^  sender,
                           System::EventArgs ^  e)
{
    TextBox ^ myText = gcnew TextBox();
    myText->Location = Point(25,25);
    this->Controls->Add(myText);
}
```

NOTE

You can also add code to initialize other properties of the control.

IMPORTANT

You might expose your local computer to a security risk through the network by referencing a malicious `UserControl`. This would only be a concern in the case of a malicious person creating a damaging custom control, followed by you mistakenly adding it to your project.

See Also

[Windows Forms Controls](#)

[Arranging Controls on Windows Forms](#)

[How to: Resize Controls on Windows Forms](#)

[How to: Set the Text Displayed by a Windows Forms Control](#)

[Controls to Use on Windows Forms](#)

How to: Add Controls Without a User Interface to Windows Forms

5/4/2018 • 2 min to read • [Edit Online](#)

A nonvisual control (or component) provides functionality to your application. Unlike other controls, components do not provide a user interface to the user and thus do not need to be displayed on the Windows Forms Designer surface. When a component is added to a form, the Windows Forms Designer displays a resizable tray at the bottom of the form where all components are displayed. Once a control has been added to the component tray, you can select the component and set its properties as you would any other control on the form.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To add a component to a Windows Form

1. Open the form. For details, see [How to: Display Windows Forms in the Designer](#).
2. In the **Toolbox**, click a component and drag it to your form.

Your component appears in the component tray.

Furthermore, components can be added to a form at run time. This is a common scenario, especially because components do not have a visual expression, unlike controls that have a user interface. In the example below, a **Timer** component is added at run time. (Note that Visual Studio contains a number of different timers; in this case, use a Windows Forms **Timer** component. For more information about the different timers in Visual Studio, see [Introduction to Server-Based Timers](#).)

Caution

Components often have control-specific properties that must be set for the component to function effectively. In the case of the **Timer** component below, you set the **Interval** property. Be sure, when adding components to your project, that you set the properties necessary for that component.

To add a component to a Windows Form programmatically

1. Create an instance of the **Timer** class in code.
2. Set the **Interval** property to determine the time between ticks of the timer.
3. Configure any other necessary properties for your component.

The following code shows the creation of a **Timer** with its **Interval** property set.

```
Public Sub CreateTimer()
    Dim timerKeepTrack As New System.Windows.Forms.Timer
    timerKeepTrack.Interval = 1000
End Sub
```

```
public void createTimer()
{
    System.Windows.Forms.Timer timerKeepTrack = new
        System.Windows.Forms.Timer();
    timerKeepTrack.Interval = 1000;
}
```

```
public:
void createTimer()
{
    System::Windows::Forms::Timer^ timerKeepTrack = gcnew
        System::Windows::Forms::Timer();
    timerKeepTrack->Interval = 1000;
}
```

IMPORTANT

You might expose your local computer to a security risk through the network by referencing a malicious UserControl. This would only be a concern in the case of a malicious person creating a damaging custom control, followed by you mistakenly adding it to your project.

See Also

[Windows Forms Controls](#)

[How to: Add Controls to Windows Forms](#)

[How to: Add ActiveX Controls to Windows Forms](#)

[How to: Copy Controls Between Windows Forms](#)

[Putting Controls on Windows Forms](#)

[Labeling Individual Windows Forms Controls and Providing Shortcuts to Them](#)

[Controls to Use on Windows Forms](#)

[Windows Forms Controls by Function](#)

How to: Add to or Remove from a Collection of Controls at Run Time

5/4/2018 • 2 min to read • [Edit Online](#)

Common tasks in application development are adding controls to and removing controls from any container control on your forms (such as the [Panel](#) or [GroupBox](#) control, or even the form itself). At design time, controls can be dragged directly onto a panel or group box. At run time, these controls maintain a `Controls` collection, which keeps track of what controls are placed on them.

NOTE

The following code example applies to any control that maintains a collection of controls within it.

To add a control to a collection programmatically

1. Create an instance of the control to be added.
2. Set properties of the new control.
3. Add the control to the `Controls` collection of the parent control.

The following code example shows how to create an instance of the [Button](#) control. It requires a form with a [Panel](#) control and that the event-handling method for the button being created, `NewPanelButton_Click`, already exists.

```
Public NewPanelButton As New Button()

Public Sub AddNewControl()
    ' The Add method will accept as a parameter any object that derives
    ' from the Control class. In this case, it is a Button control.
    Panel1.Controls.Add(NewPanelButton)
    ' The event handler indicated for the Click event in the code
    ' below is used as an example. Substitute the appropriate event
    ' handler for your application.
    AddHandler NewPanelButton.Click, AddressOf NewPanelButton_Click
End Sub
```

```
public Button newPanelButton = new Button();

public void addNewControl()
{
    // The Add method will accept as a parameter any object that derives
    // from the Control class. In this case, it is a Button control.
    panel1.Controls.Add(newPanelButton);
    // The event handler indicated for the Click event in the code
    // below is used as an example. Substitute the appropriate event
    // handler for your application.
    this.newPanelButton.Click += new System.EventHandler(this. NewPanelButton_Click);
}
```

To remove controls from a collection programmatically

1. Remove the event handler from the event. In Visual Basic, use the [RemoveHandler Statement](#) keyword; in Visual C#, use the [-= Operator \(C# Reference\)](#).

2. Use the `Remove` method to delete the desired control from the panel's `Controls` collection.

3. Call the `Dispose` method to release all the resources used by the control.

```
Public Sub RemoveControl()
    ' NOTE: The code below uses the instance of
    ' the button (NewPanelButton) from the previous example.
    If Panel1.Controls.Contains(NewPanelButton) Then
        RemoveHandler NewPanelButton.Click, AddressOf _
            NewPanelButton_Click
        Panel1.Controls.Remove(NewPanelButton)
        NewPanelButton.Dispose()
    End If
End Sub
```

```
private void removeControl(object sender, System.EventArgs e)
{
    // NOTE: The code below uses the instance of
    // the button (newPanelButton) from the previous example.
    if(panel1.Controls.Contains(newPanelButton))
    {
        this.newPanelButton.Click -= new System.EventHandler(this.
            NewPanelButton_Click);
        panel1.Controls.Remove(newPanelButton);
        newPanelButton.Dispose();
    }
}
```

See Also

[Panel](#)

[Panel Control](#)

Walkthrough: Automatically Populating the Toolbox with Custom Components

5/4/2018 • 3 min to read • [Edit Online](#)

If your components are defined by a project in the currently open solution, they will automatically appear in the **Toolbox**, with no action required by you. You can also manually populate the **Toolbox** with your custom components by using the [Choose Toolbox Items Dialog Box \(Visual Studio\)](#), but the **Toolbox** takes account of items in your solution's build outputs with all the following characteristics:

- Implements [IComponent](#);
- Does not have [ToolboxItemAttribute](#) set to `false`;
- Does not have [DesignTimeVisibleAttribute](#) set to `false`.

NOTE

The **Toolbox** does not follow reference chains, so it will not display items that are not built by a project in your solution.

This walkthrough demonstrates how a custom component automatically appears in the **Toolbox** once the component is built. Tasks illustrated in this walkthrough include:

- Creating a Windows Forms project.
- Creating a custom component.
- Creating an instance of a custom component.
- Unloading and reloading a custom component.

When you are finished, you will see that the **Toolbox** is populated with a component that you have created.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

Creating the Project

The first step is to create the project and to set up the form.

To create the project

1. Create a Windows-based application project called `ToolboxExample`.

For more information, see [How to: Create a Windows Application Project](#).

2. Add a new component to the project. Call it `DemoComponent`.

For more information, see [NIB:How to: Add New Project Items](#).

3. Build the project.

4. From the **Tools** menu, click the **Options** item. Click **General** under the **Windows Forms Designer** item and ensure that the **AutoToolboxPopulate** option is set to **True**.

Creating an Instance of a Custom Component

The next step is to create an instance of the custom component on the form. Because the **Toolbox** automatically accounts for the new component, this is as easy as creating any other component or control.

To create an instance of a custom component

1. Open the project's form in the **Forms Designer**.
2. In the **Toolbox**, click the new tab called **ToolboxExample Components**.

Once you click the tab, you will see **DemoComponent**.

NOTE

For performance reasons, components in the auto-populated area of the **Toolbox** do not display custom bitmaps, and the **ToolboxBitmapAttribute** is not supported. To display an icon for a custom component in the **Toolbox**, use the **Choose Toolbox Items** dialog box to load your component.

3. Drag your component onto your form.

An instance of the component is created and added to the **Component Tray**.

Unloading and Reloading a Custom Component

The **Toolbox** takes account of the components in each loaded project, and when a project is unloaded, it removes references to the project's components.

To experiment with the effect on the Toolbox of unloading and reloading components

1. Unload the project from the solution.

For more information about unloading projects, see [NIB:How to: Unload and Reload Projects](#). If you are prompted to save, choose **Yes**.

2. Add a new **Windows Application** project to the solution. Open the form in the **Designer**.

The **ToolboxExample Components** tab from the previous project is now gone.

3. Reload the **ToolboxExample** project.

The **ToolboxExample Components** tab now reappears.

Next Steps

This walkthrough demonstrates that the **Toolbox** takes account of a project's components, but the **Toolbox** is also takes account of controls. Experiment with your own custom controls by adding and removing control projects from your solution.

See Also

[General, Windows Forms Designer, Options Dialog Box](#)

[How to: Manipulate Toolbox Tabs](#)

[Choose Toolbox Items Dialog Box \(Visual Studio\)](#)

[Putting Controls on Windows Forms](#)

How to: Add ActiveX Controls to Windows Forms

5/4/2018 • 1 min to read • [Edit Online](#)

While the Windows Forms Designer is optimized to host Windows Forms controls, you can also put ActiveX controls on Windows Forms.

Caution

There are performance limitations for Windows Forms when ActiveX controls are added to them.

Before you add ActiveX controls to your form, you must add them to the Toolbox. For more information, see [COM Components, Customize Toolbox Dialog Box](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, click **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To add an ActiveX control to your Windows Form

- Double-click the control on the Toolbox.

Visual Studio adds all references to the control in your project. For more information about things to keep in mind when using ActiveX controls on Windows Forms, see [Considerations When Hosting an ActiveX Control on a Windows Form](#).

NOTE

The Windows Forms ActiveX Control Importer (AxImp.exe) creates event arguments of a different type than expected upon importation of ActiveX dynamic link libraries. The arguments created by AxImp.exe are similar to the following: `Invoke(object sender, DWebBrowserEvents2_ProgressChangeEvent e)`, when `Invoke(object sender, DWebBrowserEvents2_ProgressChangeEventArgs e)` is expected. Be aware that this irregularity does not prevent code from functioning normally. For details, see [Windows Forms ActiveX Control Importer \(Aximp.exe\)](#).

See Also

[Windows Forms Controls](#)

[Controls and Programmable Objects Compared in Various Languages and Libraries](#)

[How to: Add Controls to Windows Forms](#)

[Arranging Controls on Windows Forms](#)

[Labeling Individual Windows Forms Controls and Providing Shortcuts to Them](#)

[Controls to Use on Windows Forms](#)

[Windows Forms Controls by Function](#)

Considerations When Hosting an ActiveX Control on a Windows Form

5/4/2018 • 1 min to read • [Edit Online](#)

Although Windows Forms have been optimized to host Windows Forms controls, you can still use ActiveX controls. Keep the following considerations in mind when planning an application that uses ActiveX controls:

- **Security** The common language runtime has been enhanced with regard to code access security. Applications featuring Windows Forms can run in a fully trusted environment without issue and in a semi-trusted environment with most of the functionality accessible. Windows Forms controls can be hosted in a browser with no complications. However, ActiveX controls on Windows Forms cannot take advantage of these security enhancements. Running an ActiveX control requires unmanaged code permission, which is set with the [SecurityPermissionAttribute.UnmanagedCode](#) property. For more information about security and unmanaged code permission, see [SecurityPermissionAttribute](#).
- **Total Cost of Ownership** ActiveX controls added to a Windows Form are deployed with that Windows Form in their entirety, which can add significantly to the size of the file(s) created. Additionally, using ActiveX controls on Windows Forms requires writing to the registry. This is more invasive to a user's computer than Windows Forms controls, which do not require this.

NOTE

Working with an ActiveX control requires the use of a COM interop wrapper. For more information, see [COM Interoperability in Visual Basic and Visual C#](#).

NOTE

If the name of a member of the ActiveX control matches a name defined in the .NET Framework, then the ActiveX Control Importer will prefix the member name with **Ctl** when it creates the [AxHost](#) derived class. For example, if your ActiveX control has a member named **Layout**, it is renamed **CtlLayout** in the AxHost-derived class because the **Layout** event is defined within the .NET Framework.

See Also

[How to: Add ActiveX Controls to Windows Forms](#)

[Code Access Security](#)

[Controls and Programmable Objects Compared in Various Languages and Libraries](#)

[Putting Controls on Windows Forms](#)

[Windows Forms Controls](#)

Labeling Individual Windows Forms Controls and Providing Shortcuts to Them

5/4/2018 • 1 min to read • [Edit Online](#)

Controls added to Windows Forms have properties and methods that are used to further specialize the user experience. Customizing your user interface to suit the needs of the user is extremely important for well-designed Windows applications.

In This Section

[How to: Set the Text Displayed by a Windows Forms Control](#)

Describes how to assign a text label to a control.

[How to: Set the Image Displayed by a Windows Forms Control](#)

Explains how to configure a control to display images.

[How to: Create Access Keys for Windows Forms Controls](#)

Gives information about creating predefined keyboard shortcuts.

[Providing Accessibility Information for Controls on a Windows Form](#)

Gives information about enabling your controls to work with accessibility aids.

Related Sections

[Windows Forms Controls](#)

Links to other basic things you can do with controls.

Also see [How to: Create Access Keys for Windows Forms Controls Using the Designer](#), [How to: Set the Text Displayed by a Windows Forms Control Using the Designer](#), [How to: Set the Image Displayed by a Windows Forms Control Using the Designer](#).

How to: Create Access Keys for Windows Forms Controls

5/4/2018 • 1 min to read • [Edit Online](#)

An *access key* is an underlined character in the text of a menu, menu item, or the label of a control such as a button. With an access key, the user can "click" a button by pressing the ALT key in combination with the predefined access key. For example, if a button runs a procedure to print a form, and therefore its `Text` property is set to "Print," adding an ampersand before the letter "P" causes the letter "P" to be underlined in the button text at run time. The user can run the command associated with the button by pressing ALT+P. You cannot have an access key for a control that cannot receive focus.

To create an access key for a control

1. Set the `Text` property to a string that includes an ampersand (&) before the letter that will be the shortcut.

```
' Set the letter "P" as an access key.  
Button1.Text = "&Print"
```

```
// Set the letter "P" as an access key.  
button1.Text = "&Print";
```

```
// Set the letter "P" as an access key.  
button1->Text = "&Print";
```

NOTE

To include an ampersand in a caption without creating an access key, include two ampersands (&&). A single ampersand is displayed in the caption and no characters are underlined.

See Also

[Button](#)

[How to: Respond to Windows Forms Button Clicks](#)

[How to: Set the Text Displayed by a Windows Forms Control](#)

[Labeling Individual Windows Forms Controls and Providing Shortcuts to Them](#)

How to: Create Access Keys for Windows Forms Controls Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

An *access key* is an underlined character in the text of a menu, menu item, or the label of a control such as a button. It enables the user to "click" a button by pressing the ALT key in combination with the predefined access key. For example, if a button runs a procedure to print a form, and therefore its `Text` property is set to "Print," adding an ampersand (&) before the letter "P" causes the letter "P" to be underlined in the button text at run time. The user can run the command associated with the button by pressing ALT+P. You cannot have an access key for a control that cannot receive focus.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To create an access key for a control

1. In the **Properties** window, set the `Text` property to a string that includes an ampersand (&) before the letter that will be the access key. For example, to set the letter "P" as the access key, type **&Print** into the grid.

See Also

[Button](#)

[How to: Respond to Windows Forms Button Clicks](#)

[How to: Set the Text Displayed by a Windows Forms Control](#)

[Labeling Individual Windows Forms Controls and Providing Shortcuts to Them](#)

How to: Set the Text Displayed by a Windows Forms Control

5/4/2018 • 1 min to read • [Edit Online](#)

Windows Forms controls usually display some text that is related to the primary function of the control. For example, a [Button](#) control usually displays a caption indicating what action will be performed when the button is clicked. For all controls, you can set or return the text by using the [Text](#) property. You can change the font by using the [Font](#) property. You can also set the text using the designer. Also see [How to: Create Access Keys for Windows Forms Controls Using the Designer](#), [How to: Set the Text Displayed by a Windows Forms Control Using the Designer](#), [How to: Set the Image Displayed by a Windows Forms Control Using the Designer](#).

To set the text displayed by a control programmatically

1. Set the [Text](#) property to a string.

To create an underlined access key, includes an ampersand (&) before the letter that will be the access key.

2. Set the [Font](#) property to an object of type [Font](#).

```
Button1.Text = "Click here to save changes"  
Button1.Font = New Font("Arial", 10, FontStyle.Bold, GraphicsUnit.Point)
```

```
button1.Text = "Click here to save changes";  
button1.Font = new Font("Arial", 10, FontStyle.Bold,  
    GraphicsUnit.Point);
```

```
button1->Text = "Click here to save changes";  
button1->Font = new System::Drawing::Font("Arial",  
    10, FontStyle::Bold, GraphicsUnit::Point);
```

NOTE

You can use an escape character to display a special character in user-interface elements that would normally interpret them differently, such as menu items. For example, the following line of code sets the menu item's text to read "& Now For Something Completely Different":

```
MPMenuItem.Text = "&& Now For Something Completely Different"
```

```
mpMenuItem.Text = "&& Now For Something Completely Different";
```

```
mpMenuItem->Text = "&& Now For Something Completely Different";
```

See Also

[Control.Text](#)

[How to: Create Access Keys for Windows Forms Controls](#)

How to: Respond to Windows Forms Button Clicks

How to: Set the Text Displayed by a Windows Forms Control Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

Windows Forms controls typically display some text that is related to the primary function of the control. For example, a [Button](#) control typically displays a caption that indicates what action will be performed when the button is clicked. For all controls, you can set or return the text by using the [Text](#) property. You can change the font by using the [Font](#) property.

To set the text and font with the designer

1. In the Properties window, set the [Text](#) property of the control to an appropriate string.

To create an underlined shortcut key, includes an ampersand (&) before the letter that will be the shortcut key.

2. In the Properties window, click the ellipsis button (...) next to the [Font](#) property.

In the standard font dialog box, select the font, font style, size, effects (such as strikeout or underline), and script that you want.

See Also

[How to: Set the Text Displayed by a Windows Forms Control](#)

[Using Fonts and Text](#)

[Labeling Individual Windows Forms Controls and Providing Shortcuts to Them](#)

How to: Set the Image Displayed by a Windows Forms Control

5/4/2018 • 1 min to read • [Edit Online](#)

Several Windows Forms controls can display images. These images can be icons that clarify the purpose of the control, such as a diskette icon on a button denoting the **Save** command. Alternatively, the icons can be background images to give the control the appearance and behavior you want.

To set the image displayed by a control

1. Set the control's `Image` or `BackgroundImage` property to an object of type `Image`. Generally, you will be loading the image from a file by using the `FromFile` method.

In the following code example, the path set for the location of the image is the **My Pictures** folder. Most computers running the Windows operating system will include this directory. This also enables users with minimal system access levels to run the application safely. The following code example requires that you already have a form with a `PictureBox` control added.

```
' Replace the image named below
' with an icon of your own choosing.
PictureBox1.Image = Image.FromFile _
    (System.Environment.GetFolderPath _
    (System.Environment.SpecialFolder.MyPictures) _
    & "\Image.gif")
```

```
// Replace the image named below
// with an icon of your own choosing.
// Note the escape character used (@) when specifying the path.
pictureBox1.Image = Image.FromFile
    (System.Environment.GetFolderPath
    (System.Environment.SpecialFolder.MyPictures)
    + @"\Image.gif");
```

```
// Replace the image named below
// with an icon of your own choosing.
pictureBox1->Image = Image::FromFile(String::Concat
    (System::Environment::GetFolderPath
    (System::Environment::SpecialFolder::MyPictures),
    "\\Image.gif"));
```

See Also

[FromFile](#)

[Image](#)

[BackgroundImage](#)

How to: Set the Image Displayed by a Windows Forms Control Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

Several Windows Forms controls can display images. The image can be an icon that clarifies the purpose of the control, such as a disk icon on a button denoting the **Save** command. Alternatively, the icon can be a background image to give the control the appearance you want.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To set the image displayed by a control

1. In the **Properties** window, select the **Image** or **BackgroundImage** property of the control, then click the ellipsis button () to display the **Select Resource** dialog box.
2. Select the image you want to display.

See Also

[FromFile](#)

[Image](#)

[BackgroundImage](#)

[Labeling Individual Windows Forms Controls and Providing Shortcuts to Them](#)

Providing Accessibility Information for Controls on a Windows Form

5/4/2018 • 1 min to read • [Edit Online](#)

Accessibility aids are specialized programs and devices that help people with disabilities use computers more effectively. Examples include screen readers for people who are blind and voice input utilities for people who provide verbal commands instead of using the mouse or keyboard. These accessibility aids interact with the accessibility properties exposed by Windows Forms controls. These properties are:

- **AccessibilityObject**
- **AccessibleDefaultActionDescription**
- **AccessibleDescription**
- **AccessibleName**
- **AccessibleRole**

AccessibilityObject Property

This read-only property contains an [AccessibleObject](#) instance. The **AccessibleObject** implements the [IAccessible](#) interface, which provides information about the control's description, screen location, navigational abilities, and value. The designer sets this value when the control is added to the form.

AccessibleDefaultActionDescription Property

This string describes the action of the control. It does not appear in the Properties window and may only be set in code. The following example sets this property for a button control:

```
' Visual Basic
Button1.AccessibleDefaultActionDescription = _
    "Closes the application."

// C#
Button1.AccessibleDefaultActionDescription =
    "Closes the application.';

// C++
button1->AccessibleDefaultActionDescription =
    "Closes the application.;"
```

AccessibleDescription Property

This string describes the control. It may be set in the Properties window, or in code as follows:

```
' Visual Basic
Button1.AccessibleDescription = "A button with text 'Exit'."

// C#
Button1.AccessibleDescription = "A button with text 'Exit'";

// C++
button1->AccessibleDescription = "A button with text 'Exit'";
```

AccessibleName Property

This is the name of a control reported to accessibility aids. It may be set in the Properties window, or in code as follows:

```
' Visual Basic
Button1.AccessibleName = "Order"

// C#
Button1.AccessibleName = "Order";

// C++
button1->AccessibleName = "Order";
```

AccessibleRole Property

This property, which contains an [AccessibleRole](#) enumeration, describes the user interface role of the control. A new control has the value set to `Default`. This would mean that by default, a **Button** control acts as a **Button**. You may want to reset this property if a control has another role. For example, you may be using a **PictureBox** control as a **Chart**, and you may want accessibility aids to report the role as a **Chart**, not as a **PictureBox**. You may also want to specify this property for custom controls you have developed. This property may be set in the Properties window, or in code as follows:

```
' Visual Basic
PictureBox1.AccessibleRole = AccessibleRole.Chart

// C#
PictureBox1.AccessibleRole = AccessibleRole.Chart;

// C++
pictureBox1->AccessibleRole = AccessibleRole::Chart;
```

See Also

[AccessibleObject](#)
[Control.AccessibleObject](#)
[Control.AccessibleDefaultActionDescription](#)
[Control.AccessibleDescription](#)
[Control.AccessibleName](#)
[Control.AccessibleRole](#)
[AccessibleRole](#)

Controls to Use on Windows Forms

5/4/2018 • 6 min to read • [Edit Online](#)

The following is an alphabetic list of controls and components that can be used on Windows Forms. In addition to the Windows Forms controls covered in this section, you can add ActiveX and custom controls to Windows Forms. If you do not find the control you need listed here, you can also create your own. For details, see [Developing Windows Forms Controls at Design Time](#). For more information about choosing the control you need, see [Windows Forms Controls by Function](#).

NOTE

Visual Basic controls are based on classes provided by the .NET Framework.

In This Section

[Windows Forms Controls by Function](#)

Lists and describes Windows Forms controls based on the .NET Framework.

[Controls with Built-In Owner-Drawing Support](#)

Describes how to alter aspects of a control's appearance that are not available through properties.

[BackgroundWorker Component](#)

Enables a form or control to run an operation asynchronously.

[BindingNavigator Control](#)

Provides the navigation and manipulation user interface (UI) for controls that are bound to data.

[BindingSource Component](#)

Encapsulates a data source for binding to controls.

[Button Control](#)

Presents a standard button that the user can click to perform actions.

[CheckBox Control](#)

Indicates whether a condition is on or off.

[CheckedListBox Control](#)

Displays a list of items with a check box next to each item.

[ColorDialog Component](#)

Allows the user to select a color from a palette in a pre-configured dialog box and to add custom colors to that palette.

[ComboBox Control](#)

Displays data in a drop-down combo box.

[ContextMenu Component](#)

Provides users with an easily accessible menu of frequently used commands that are associated with the selected object. Although [ContextMenuStrip](#) replaces and adds functionality to the [ContextMenu](#) control of previous versions, [ContextMenu](#) is retained for both backward compatibility and future use if so desired.

[ContextMenuStrip Control](#)

Represents a shortcut menu. Although [ContextMenuStrip](#) replaces and adds functionality to the [ContextMenu](#) control of previous versions, [ContextMenu](#) is retained for both backward compatibility and future use if so desired.

[DataGrid Control](#)

Displays tabular data from a dataset and allows for updates to the data source.

[DataGridView Control](#)

Provides a flexible, extensible system for displaying and editing tabular data.

[DateTimePicker Control](#)

Allows the user to select a single item from a list of dates or times.

[Dialog-Box Controls and Components](#)

Describes a set of controls that allow users to perform standard interactions with the application or system.

[DomainUpDown Control](#)

Displays text strings that a user can browse through and select from.

[ErrorProvider Component](#)

Displays error information to the user in a non-intrusive way.

[FileDialog Class](#) Provides base-class functionality for file dialog boxes.

[FlowLayoutPanel Control](#)

Represents a panel that dynamically lays out its contents horizontally or vertically.

[FolderBrowserDialog Component](#)

Displays an interface with which users can browse and select a directory or create a new one.

[FontDialog Component](#)

Exposes the fonts that are currently installed on the system.

[GroupBox Control](#)

Provides an identifiable grouping for other controls.

[HelpProvider Component](#)

Associates an HTML Help file with a Windows-based application.

[HScrollBar and VScrollBar Controls](#)

Provide navigation through a list of items or a large amount of information by scrolling either horizontally or vertically within an application or control.

[ImageList Component](#)

Displays images on other controls.

[Label Control](#)

Displays text that cannot be edited by the user.

[LinkLabel Control](#)

Allows you to add Web-style links to Windows Forms applications.

[ListBox Control](#)

Allows the user to select one or more items from a predefined list.

[ListView Control](#)

Displays a list of items with icons, in the manner of Windows Explorer.

[MainMenu Component](#)

Displays a menu at run time. Although [MenuStrip](#) replaces and adds functionality to the [MainMenu](#) control of previous versions, [MainMenu](#) is retained for both backward compatibility and future use if you choose.

[MaskedTextBox Control](#)

Constrains the format of user input in a form.

[MenuStrip Control](#)

Provides a menu system for a form. Although [MenuStrip](#) replaces and adds functionality to the [MainMenu](#) control of previous versions, [MainMenu](#) is retained for both backward compatibility and future use if you choose.

[MonthCalendar Control](#)

Presents an intuitive graphical interface for users to view and set date information.

[NotifyIcon Component](#)

Displays icons for processes that run in the background and would not otherwise have user interfaces.

[NumericUpDown Control](#)

Displays numerals that a user can browse through and select from.

[OpenFileDialog Component](#)

Allows users to open files by using a pre-configured dialog box.

[PageSetupDialog Component](#)

Sets page details for printing through a pre-configured dialog box.

[Panel Control](#)

Provide an identifiable grouping for other controls, and allows for scrolling.

[PictureBox Control](#)

Displays graphics in bitmap, GIF, JPEG, metafile, or icon format.

[PrintDialog Component](#)

Selects a printer, chooses the pages to print, and determines other print-related settings.

[PrintDocument Component](#)

Sets the properties that describe what to print, and prints the document in Windows-based applications.

[PrintPreviewControl Control](#)

Allows you to create your own [PrintPreview](#) component or dialog box instead of using the pre-configured version.

[PrintPreviewDialog Control](#)

Displays a document as it will appear when it is printed.

[ProgressBar Control](#)

Graphically indicates the progress of an action towards completion.

[RadioButton Control](#)

Presents a set of two or more mutually exclusive options to the user.

[RichTextBox Control](#)

Allows users to enter, display, and manipulate text with formatting.

[SaveFileDialog Component](#)

Selects files to save and where to save them.

[SoundPlayer Class](#) Enables you to easily include sounds in your applications.

[SplitContainer Control](#)

Allows the user to resize a docked control.

[Splitter Control](#)

Allows the user to resize a docked control (.NET Framework version 1.x).

[StatusBar Control](#)

Displays status information related to the control that has focus. Although [StatusStrip](#) replaces and extends the [StatusBar](#) control of previous versions, [StatusBar](#) is retained for both backward compatibility and future use if you choose.

[StatusStrip Control](#)

Represents a Windows status bar control. Although [StatusStrip](#) replaces and extends the [StatusBar](#) control of previous versions, [StatusBar](#) is retained for both backward compatibility and future use if you choose.

[TabControl Control](#)

Displays multiple tabs that can contain pictures or other controls.

[TableLayoutPanel Control](#)

Represents a panel that dynamically lays out its contents in a grid composed of rows and columns.

[TextBox Control](#)

Allows editable, multiline input from the user.

[Timer Component](#)

Raises an event at regular intervals.

[ToolBar Control](#)

Displays menus and bitmapped buttons that activate commands. You can extend the functionality of the control and modify its appearance and behavior. Although [ToolStrip](#) replaces and adds functionality to the [ToolBar](#) control of previous versions, [ToolBar](#) is retained for both backward compatibility and future use if you choose.

[ToolStrip Control](#)

Creates custom toolbars and menus in your Windows Forms applications. Although [ToolStrip](#) replaces and adds functionality to the [ToolBar](#) control of previous versions, [ToolBar](#) is retained for both backward compatibility and future use if you choose.

[ToolStripContainer Control](#)

Provides panels on each side of a form for docking, rafting, and arranging [ToolStrip](#) controls, and a central [ToolStripContentPanel](#) for traditional controls.

[ToolStripPanel Control](#)

Provides one panel for docking, rafting and arranging [ToolStrip](#) controls.

[ToolStripProgressBar Control Overview](#)

Graphically indicates the progress of an action towards completion. The [ToolStripProgressBar](#) is typically contained in a [StatusStrip](#).

[ToolStripStatusLabel Control](#)

Represents a panel in a [StatusStrip](#) control.

[ToolTip Component](#)

Displays text when the user points at other controls.

[TrackBar Control](#)

Allows navigation through a large amount of information or visually adjusting a numeric setting.

[TreeView Control](#)

Displays a hierarchy of nodes that can be expanded or collapsed.

[WebBrowser Control](#)

Hosts Web pages and provides Internet Web browsing capabilities to your application.

[Windows Forms Controls Used to List Options](#)

Describes a set of controls used to provide users with a list of options to choose from.

Related Sections

[Windows Forms Controls](#)

Explains the use of Windows Forms controls, and describes important concepts for working with them.

[Developing Windows Forms Controls at Design Time](#)

Provides links to step-by-step topics, recommendations for which kind of control to create, and other information about creating your own control.

[Controls and Programmable Objects Compared in Various Languages and Libraries](#)

Provides a table that maps controls in Visual Basic 6.0 to the corresponding control in Visual Basic 2005. Note that controls are now classes in the .NET Framework.

[How to: Add ActiveX Controls to Windows Forms](#)

Describes how to use ActiveX controls on Windows Forms.

Windows Forms Controls by Function

5/4/2018 • 4 min to read • [Edit Online](#)

Windows Forms offers controls and components that perform a number of functions. The following table lists the Windows Forms controls and components according to general function. In addition, where multiple controls exist that serve the same function, the recommended control is listed with a note regarding the control it superseded. In a separate subsequent table, the superseded controls are listed with their recommended replacements.

NOTE

The following tables do not list every control or component you can use in Windows Forms; for a more comprehensive list, see [Controls to Use on Windows Forms](#)

Recommended Controls and Components by Function

FUNCTION	CONTROL	DESCRIPTION
Data display	DataGridView control	The DataGridView control provides a customizable table for displaying data. The DataGridView class enables customization of cells, rows, columns, and borders. Note: The DataGridView control provides numerous basic and advanced features that are missing in the DataGrid control. For more information, see Differences Between the Windows Forms DataGridView and DataGrid Controls
Data binding and navigation	BindingSource component	Simplifies binding controls on a form to data by providing currency management, change notification, and other services.
	BindingNavigator control	Provides a toolbar-type interface to navigate and manipulate data on a form.
Text editing	TextBox control	Displays text entered at design time that can be edited by users at run time, or changed programmatically.
	RichTextBox control	Enables text to be displayed with formatting in plain text or rich-text format (RTF).
	MaskedTextBox control	Constrains the format of user input
Information display (read-only)	Label control	Displays text that users cannot directly edit.

FUNCTION	CONTROL	DESCRIPTION
	LinkLabel control	Displays text as a Web-style link and triggers an event when the user clicks the special text. Usually the text is a link to another window or a Web site.
	StatusStrip control	Displays information about the application's current state using a framed area, usually at the bottom of a parent form.
	ProgressBar control	Displays the current progress of an operation to the user.
Web page display	WebBrowser control	Enables the user to navigate Web pages inside your form.
Selection from a list	CheckedListBox control	Displays a scrollable list of items, each accompanied by a check box.
	ComboBox control	Displays a drop-down list of items.
	DomainUpDown control	Displays a list of text items that users can scroll through with up and down buttons.
	ListBox control	Displays a list of text and graphical items (icons).
	ListView control	Displays items in one of four different views. Views include text only, text with small icons, text with large icons, and a details view.
	NumericUpDown control	Displays a list of numerals that users can scroll through with up and down buttons.
	TreeView control	Displays a hierarchical collection of node objects that can consist of text with optional check boxes or icons.
Graphics display	PictureBox control	Displays graphical files, such as bitmaps and icons, in a frame.
Graphics storage	ImageList control	Serves as a repository for images. ImageList controls and the images they contain can be reused from one application to the next.
Value setting	CheckBox control	Displays a check box and a label for text. Generally used to set options.
	CheckedListBox control	Displays a scrollable list of items, each accompanied by a check box.

FUNCTION	CONTROL	DESCRIPTION
	RadioButton control	Displays a button that can be turned on or off.
	TrackBar control	Allows users to set values on a scale by moving a "thumb" along a scale.
Date setting	DateTimePicker control	Displays a graphical calendar to allow users to select a date or a time.
	MonthCalendar control	Displays a graphical calendar to allow users to select a range of dates.
Dialog boxes	ColorDialog control	Displays the color picker dialog box that allows users to set the color of an interface element.
	FontDialog control	Displays a dialog box that allows users to set a font and its attributes.
	OpenFileDialog control	Displays a dialog box that allows users to navigate to and select a file.
	PrintDialog control	Displays a dialog box that allows users to select a printer and set its attributes.
	PrintPreviewDialog control	Displays a dialog box that displays how a control PrintDocument component will appear when printed.
	FolderBrowserDialog control	Displays a dialog that allows users to browse, create, and eventually select a folder
	SaveFileDialog control	Displays a dialog box that allows users to save a file.
Menu controls	MenuStrip control	Creates custom menus. Note: The MenuStrip is designed to replace the MainMenu control.
	ContextMenuStrip control	Creates custom context menus. Note: The ContextMenuStrip is designed to replace the ContextMenu control.
Commands	Button control	Starts, stops, or interrupts a process.
	LinkLabel control	Displays text as a Web-style link and triggers an event when the user clicks the special text. Usually the text is a link to another window or a Web site.
	NotifyIcon control	Displays an icon in the status notification area of the taskbar that represents an application running in the background.

FUNCTION	CONTROL	DESCRIPTION
	ToolStrip control	Creates toolbars that can have a Microsoft Windows XP, Microsoft Office, Microsoft Internet Explorer, or custom look and feel, with or without themes, and with support for overflow and runtime item reordering. Note: The ToolStrip control is designed to replace the ToolBar control.
User Help	HelpProvider component	Provides pop-up or online Help for controls.
	ToolTip component	Provides a pop-up window that displays a brief description of a control's purpose when the user rests the pointer on the control.
Grouping other controls	Panel control	Groups a set of controls on an unlabeled, scrollable frame.
	GroupBox control	Groups a set of controls (such as radio buttons) on a labeled, nonscrollable frame.
	TabControl control	Provides a tabbed page for organizing and accessing grouped objects efficiently.
	SplitContainer control	Provides two panels separated by a movable bar. Note: The SplitContainer control is designed to replace the Splitter control.
	TableLayoutPanel control	Represents a panel that dynamically lays out its contents in a grid composed of rows and columns.
	FlowLayoutPanel control	Represents a panel that dynamically lays out its contents horizontally or vertically.
Audio	SoundPlayer control	Plays sound files in the .wav format. Sounds can be loaded or played asynchronously.

Superseded Controls and Components by Function

FUNCTION	SUPERSEDED CONTROL	RECOMMENDED REPLACEMENT
Data display	DataGrid	DataGridView

FUNCTION	SUPERSEDED CONTROL	RECOMMENDED REPLACEMENT
Information Display (Read-only controls)	StatusBar	StatusStrip
Menu controls	ContextMenu	ContextMenuStrip
	MainMenu	MenuStrip
Commands	ToolBar	ToolStrip
	StatusBar	StatusStrip
Form layout	Splitter	SplitContainer

See Also

[Controls to Use on Windows Forms](#)

[Developing Custom Windows Forms Controls with the .NET Framework](#)

Controls with Built-In Owner-Drawing Support

5/4/2018 • 6 min to read • [Edit Online](#)

Owner drawing in Windows Forms, which is also known as custom drawing, is a technique for changing the visual appearance of certain controls.

NOTE

The word "control" in this topic is used to mean classes that derive from either [Control](#) or [Component](#).

Typically, Windows handles painting automatically by using property settings such as [BackColor](#) to determine the appearance of a control. With owner drawing, you take over the painting process, changing elements of appearance that are not available by using properties. For example, many controls let you set the color of the text that is displayed, but you are limited to a single color. Owner drawing enables you to do things like display part of the text in black and part in red.

In practice, owner drawing is similar to drawing graphics on a form. For example, you could use graphics methods in a handler for the form's [Paint](#) event to emulate a [ListBox](#) control, but you would have to write your own code to handle all user interaction. With owner drawing, the control uses your code to draw its contents but otherwise retains all its intrinsic capabilities. You can use graphics methods to draw each item in the control or to customize some aspects of each item while you use the default appearance for other aspects of each item.

Owner Drawing in Windows Forms Controls

To perform owner drawing in controls that support it, you will typically set one property and handle one or more events.

Most controls that support owner drawing have an [OwnerDraw](#) or [DrawMode](#) property that indicates whether the control will raise its drawing-related event or events when it paints itself.

Controls that do not have an [OwnerDraw](#) or [DrawMode](#) property include the [DataGridView](#) control, which provides drawing events that occur automatically, and the [ToolStrip](#) control, which is drawn using an external rendering class that has its own drawing-related events.

There are many different kinds of drawing events, but a typical drawing event occurs in order to draw a single item within a control. The event handler receives an [EventArgs](#) object that contains information about the item being drawn and tools you can use to draw it. For example, this object typically contains the item's index number within its parent collection, a [Rectangle](#) that indicates the item's display boundaries, and a [Graphics](#) object for calling paint methods. For some events, the [EventArgs](#) object provides additional information about the item and methods that you can call to paint some aspects of the item by default, such as the background or a focus rectangle.

To create a reusable control that contains your owner-drawn customizations, create a new class that derives from a control class that supports owner drawing. Rather than handling drawing events, include your owner-drawing code in overrides for the appropriate [on EventName](#) method or methods in the new class. Make sure that you call the base class [on EventName](#) method or methods in this case so that users of your control can handle owner-drawing events and provide additional customization.

The following Windows Forms controls support owner drawing in all versions of the .NET Framework:

- [ListBox](#)
- [ComboBox](#)

- [MenuItem](#) (used by [MainMenu](#) and [ContextMenu](#))
- [TabControl](#)

The following controls support owner drawing only in .NET Framework 2.0:

- [ToolTip](#)
- [ListView](#)
- [TreeView](#)

The following controls support owner drawing and are new in .NET Framework 2.0:

- [DataGridView](#)
- [ToolStrip](#)

The following sections provide additional details for each of these controls.

ListBox and ComboBox Controls

The [ListBox](#) and [ComboBox](#) controls enable you to draw individual items in the control either all in one size, or in varying sizes.

NOTE

Although the [CheckedListBox](#) control is derived from the [ListBox](#) control, it does not support owner drawing.

To draw each item the same size, set the `DrawMode` property to [OwnerDrawFixed](#) and handle the `DrawItem` event.

To draw each item using a different size, set the `DrawMode` property to [OwnerDrawVariable](#) and handle both the `MeasureItem` and `DrawItem` events. The `MeasureItem` event lets you indicate the size of an item before the `DrawItem` event occurs for that item.

For more information, including code examples, see the following topics:

- [ListBox.DrawMode](#)
- [ListBox.MeasureItem](#)
- [ListBox.DrawItem](#)
- [ComboBox.DrawMode](#)
- [ComboBox.MeasureItem](#)
- [ComboBox.DrawItem](#)
- [How to: Create Variable Sized Text in a ComboBox Control](#)

MenuItem Component

The [MenuItem](#) component represents a single menu item in a [MainMenu](#) or [ContextMenu](#) component.

To draw a [MenuItem](#), set its `OwnerDraw` property to `true` and handle its `DrawItem` event. To customize the size of the menu item before the `DrawItem` event occurs, handle the item's `MeasureItem` event.

For more information, including code examples, see the following reference topics:

- [MenuItem.OwnerDraw](#)
- [MenuItem.DrawItem](#)

- [MenuItem.MeasureItem](#)

TabControl Control

The [TabControl](#) control enables you to draw individual tabs in the control. Owner drawing affects only the tabs; the [TabPage](#) contents are not affected.

To draw each tab in a [TabControl](#), set the [DrawMode](#) property to [OwnerDrawFixed](#) and handle the [DrawItem](#) event. This event occurs once for each tab only when the tab is visible in the control.

For more information, including code examples, see the following reference topics:

- [TabControl.DrawMode](#)
- [TabControl.DrawItem](#)

ToolTip Component

The [ToolTip](#) component enables you to draw the entire ToolTip when it is displayed.

To draw a [ToolTip](#), set its [OwnerDraw](#) property to [true](#) and handle its [Draw](#) event. To customize the size of the [ToolTip](#) before the [Draw](#) event occurs, handle the [Popup](#) event and set the [ToolTipSize](#) property in the event handler.

For more information, including code examples, see the following reference topics:

- [ToolTip.OwnerDraw](#)
- [ToolTip.Draw](#)
- [ToolTip.Popup](#)

ListView Control

The [ListView](#) control enables you to draw individual items, subitems, and column headers in the control.

To enable owner drawing in the control, set the [OwnerDraw](#) property to [true](#).

To draw each item in the control, handle the [DrawItem](#) event.

To draw each subitem or column header in the control when the [View](#) property is set to [Details](#), handle the [DrawSubItem](#) and [DrawColumnHeader](#) events.

For more information, including code examples, see the following reference topics:

- [ListView.OwnerDraw](#)
- [ListView.DrawItem](#)
- [ListView.DrawSubItem](#)
- [ListView.DrawColumnHeader](#)

TreeView Control

The [TreeView](#) control enables you to draw individual nodes in the control.

To draw only the text displayed in each node, set the [DrawMode](#) property to [OwnerDrawText](#) and handle the [DrawNode](#) event to draw the text.

To draw all elements of each node, set the [DrawMode](#) property to [OwnerDrawAll](#) and handle the [DrawNode](#) event to draw whichever elements you need, such as text, icons, check boxes, plus and minus signs, and lines connecting the nodes.

For more information, including code examples, see the following reference topics:

- [TreeView.DrawMode](#)
- [TreeView.DrawNode](#)

DataGridView Control

The [DataGridView](#) control enables you to draw individual cells and rows in the control.

To draw individual cells, handle the [CellPainting](#) event.

To draw individual rows or elements of rows, handle one or both of the [RowPrePaint](#) and [RowPostPaint](#) events.

The [RowPrePaint](#) event occurs before the cells in a row are painted, and the [RowPostPaint](#) event occurs after the cells are painted. You can handle both events and the [CellPainting](#) event to paint row background, individual cells, and row foreground separately, or you can provide specific customizations where you need them and use the default display for other elements of the row.

For more information, including code examples, see the following topics:

- [CellPainting](#)
- [RowPrePaint](#)
- [RowPostPaint](#)
- [How to: Customize the Appearance of Cells in the Windows Forms DataGridView Control](#)
- [How to: Customize the Appearance of Rows in the Windows Forms DataGridView Control](#)

ToolStrip Control

[ToolStrip](#) and derived controls enable you to customize any aspect of their appearance.

To provide custom rendering for [ToolStrip](#) controls, set the [Renderer](#) property of a [ToolStrip](#), [ToolStripManager](#), [ToolStripPanel](#), or [ToolStripContentPanel](#) to a [ToolStripRenderer](#) object and handle one or more of the many drawing events provided by the [ToolStripRenderer](#) class. Alternatively, set a [Renderer](#) property to an instance of your own class derived from [ToolStripRenderer](#), [ToolStripProfessionalRenderer](#), or [ToolStripSystemRenderer](#) that implements or overrides specific `on EventName` methods.

For more information, including code examples, see the following topics:

- [ToolStripRenderer](#)
- [How to: Create and Set a Custom Renderer for the ToolStrip Control in Windows Forms](#)
- [How to: Custom Draw a ToolStrip Control](#)

See Also

[Controls to Use on Windows Forms](#)

BackgroundWorker Component

5/4/2018 • 1 min to read • [Edit Online](#)

The `BackgroundWorker` component enables your form or control to run an operation asynchronously.

In This Section

[BackgroundWorker Component Overview](#)

Describes the `BackgroundWorker` component, which gives you the ability to execute time-consuming operations asynchronously ("in the background"), on a thread different from your application's main UI thread.

[Walkthrough: Running an Operation in the Background](#)

Demonstrates how to use the `BackgroundWorker` component in the designer to run a time-consuming operation on a separate thread.

[How to: Run an Operation in the Background](#)

Demonstrates how to use the `BackgroundWorker` component to run a time-consuming operation on a separate thread.

[Walkthrough: Implementing a Form That Uses a Background Operation](#)

Creates an application using the designer that does mathematical computations asynchronously.

[How to: Implement a Form That Uses a Background Operation](#)

Creates an application that does mathematical computations asynchronously.

[How to: Download a File in the Background](#)

Demonstrates how to use the `BackgroundWorker` component to download a file on a separate thread.

Reference

[BackgroundWorker](#)

Describes this class and has links to all its members.

[RunWorkerCompletedEventArgs](#)

Describes the type that holds data for the [RunWorkerCompleted](#) event.

[ProgressChangedEventArgs](#)

Describes the type that holds data for the [ProgressChanged](#) event.

Related Sections

[Event-based Asynchronous Pattern Overview](#)

Describes how the asynchronous pattern makes available the advantages of multithreaded applications while hiding many of the complex issues inherent in multithreaded design.

BackgroundWorker Component Overview

5/4/2018 • 2 min to read • [Edit Online](#)

There are many commonly performed operations that can take a long time to execute. For example:

- Image downloads
- Web service invocations
- File downloads and uploads (including for peer-to-peer applications)
- Complex local computations
- Database transactions
- Local disk access, given its slow speed relative to memory access

Operations like these can cause your user interface to hang while they are running. When you want a responsive UI and you are faced with long delays associated with such operations, the [BackgroundWorker](#) component provides a convenient solution.

The [BackgroundWorker](#) component gives you the ability to execute time-consuming operations asynchronously ("in the background"), on a thread different from your application's main UI thread. To use a [BackgroundWorker](#), you simply tell it what time-consuming worker method to execute in the background, and then you call the [RunWorkerAsync](#) method. Your calling thread continues to run normally while the worker method runs asynchronously. When the method is finished, the [BackgroundWorker](#) alerts the calling thread by firing the [RunWorkerCompleted](#) event, which optionally contains the results of the operation.

The [BackgroundWorker](#) component is available from the [Toolbox](#), in the [Components](#) tab. To add a [BackgroundWorker](#) to your form, drag the [BackgroundWorker](#) component onto your form. It appears in the component tray, and its properties appear in the [Properties](#) window.

To start your asynchronous operation, use the [RunWorkerAsync](#) method. [RunWorkerAsync](#) takes an optional [object](#) parameter, which can be used to pass arguments to your worker method. The [BackgroundWorker](#) class exposes the [DoWork](#) event, to which your worker thread is attached through a [DoWork](#) event handler.

The [DoWork](#) event handler takes a [DoWorkEventArgs](#) parameter, which has an [Argument](#) property. This property receives the parameter from [RunWorkerAsync](#) and can be passed to your worker method, which will be called in the [DoWork](#) event handler. The following example shows how to assign a result from a worker method called [ComputeFibonacci](#). It is part of a larger example, which you can find at [How to: Implement a Form That Uses a Background Operation](#).

```
// This event handler is where the actual,
// potentially time-consuming work is done.
void backgroundWorker1_DoWork( Object^ sender, DoWorkEventArgs^ e )
{
    // Get the BackgroundWorker that raised this event.
    BackgroundWorker^ worker = dynamic_cast<BackgroundWorker^>(sender);

    // Assign the result of the computation
    // to the Result property of the DoWorkEventArgs
    // object. This is will be available to the
    // RunWorkerCompleted eventhandler.
    e->Result = ComputeFibonacci( safe_cast<Int32>(e->Argument), worker, e );
}
```

```

// This event handler is where the actual,
// potentially time-consuming work is done.
private void backgroundWorker1_DoWork(object sender,
    DoWorkEventArgs e)
{
    // Get the BackgroundWorker that raised this event.
    BackgroundWorker worker = sender as BackgroundWorker;

    // Assign the result of the computation
    // to the Result property of the DoWorkEventArgs
    // object. This is will be available to the
    // RunWorkerCompleted eventhandler.
    e.Result = ComputeFibonacci((int)e.Argument, worker, e);
}

```

```

' This event handler is where the actual work is done.
Private Sub backgroundWorker1_DoWork( _
    ByVal sender As Object, _
    ByVal e As DoWorkEventArgs) _
Handles backgroundWorker1.DoWork

    ' Get the BackgroundWorker object that raised this event.
    Dim worker As BackgroundWorker = _
        CType(sender, BackgroundWorker)

    ' Assign the result of the computation
    ' to the Result property of the DoWorkEventArgs
    ' object. This is will be available to the
    ' RunWorkerCompleted eventhandler.
    e.Result = ComputeFibonacci(e.Argument, worker, e)
End Sub 'backgroundWorker1_DoWork

```

For more information on using event handlers, see [Events](#).

Caution

When using multithreading of any sort, you potentially expose yourself to very serious and complex bugs. Consult the [Managed Threading Best Practices](#) before implementing any solution that uses multithreading.

For more information on using the [BackgroundWorker](#) class, see [How to: Run an Operation in the Background](#).

See Also

[NOT IN BUILD: Multithreading in Visual Basic](#)

[How to: Implement a Form That Uses a Background Operation](#)

How to: Run an Operation in the Background

5/4/2018 • 6 min to read • [Edit Online](#)

If you have an operation that will take a long time to complete, and you do not want to cause delays in your user interface, you can use the [BackgroundWorker](#) class to run the operation on another thread.

The following code example shows how to run a time-consuming operation in the background. The form has **Start** and **Cancel** buttons. Click the **Start** button to run an asynchronous operation. Click the **Cancel** button to stop a running asynchronous operation. The outcome of each operation is displayed in a [MessageBox](#).

There is extensive support for this task in Visual Studio.

Also see [Walkthrough: Running an Operation in the Background](#).

Example

```
using System;
using System.ComponentModel;
using System.Drawing;
using System.Threading;
using System.Windows.Forms;

namespace BackgroundWorkerExample
{
    public class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
        {
            // Do not access the form's BackgroundWorker reference directly.
            // Instead, use the reference provided by the sender parameter.
            BackgroundWorker bw = sender as BackgroundWorker;

            // Extract the argument.
            int arg = (int)e.Argument;

            // Start the time-consuming operation.
            e.Result = TimeConsumingOperation(bw, arg);

            // If the operation was canceled by the user,
            // set the DoWorkEventArgs.Cancel property to true.
            if (bw.CancellationPending)
            {
                e.Cancel = true;
            }
        }

        // This event handler demonstrates how to interpret
        // the outcome of the asynchronous operation implemented
        // in the DoWork event handler.
        private void backgroundWorker1_RunWorkerCompleted(
            object sender,
            RunWorkerCompletedEventArgs e)
        {
            if (e.Cancelled)
            {

```

```

        // The user canceled the operation.
        MessageBox.Show("Operation was canceled");
    }
    else if (e.Error != null)
    {
        // There was an error during the operation.
        string msg = String.Format("An error occurred: {0}", e.Error.Message);
        MessageBox.Show(msg);
    }
    else
    {
        // The operation completed normally.
        string msg = String.Format("Result = {0}", e.Result);
        MessageBox.Show(msg);
    }
}

// This method models an operation that may take a long time
// to run. It can be cancelled, it can raise an exception,
// or it can exit normally and return a result. These outcomes
// are chosen randomly.
private int TimeConsumingOperation(
    BackgroundWorker bw,
    int sleepPeriod )
{
    int result = 0;

    Random rand = new Random();

    while (!bw.CancellationPending)
    {
        bool exit = false;

        switch (rand.Next(3))
        {
            // Raise an exception.
            case 0:
            {
                throw new Exception("An error condition occurred.");
                break;
            }

            // Sleep for the number of milliseconds
            // specified by the sleepPeriod parameter.
            case 1:
            {
                Thread.Sleep(sleepPeriod);
                break;
            }

            // Exit and return normally.
            case 2:
            {
                result = 23;
                exit = true;
                break;
            }

            default:
            {
                break;
            }
        }

        if( exit )
        {
            break;
        }
    }
}

```

```

        return result;
    }

private void startBtn_Click(object sender, EventArgs e)
{
    this.backgroundWorker1.RunWorkerAsync(2000);
}

private void cancelBtn_Click(object sender, EventArgs e)
{
    this.backgroundWorker1.CancelAsync();
}

/// <summary>
/// Required designer variable.
/// </summary>
private System.ComponentModel.IContainer components = null;

/// <summary>
/// Clean up any resources being used.
/// </summary>
/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
protected override void Dispose(bool disposing)
{
    if (disposing && (components != null))
    {
        components.Dispose();
    }
    base.Dispose(disposing);
}

#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.backgroundWorker1 = new System.ComponentModel.BackgroundWorker();
    this.startBtn = new System.Windows.Forms.Button();
    this.cancelBtn = new System.Windows.Forms.Button();
    this.SuspendLayout();
    //
    // backgroundWorker1
    //
    this.backgroundWorker1.WorkerSupportsCancellation = true;
    this.backgroundWorker1.DoWork += new
System.ComponentModel.DoWorkEventHandler(this.backgroundWorker1_DoWork);
    this.backgroundWorker1.RunWorkerCompleted += new
System.ComponentModel.RunWorkerCompletedEventHandler(this.backgroundWorker1_RunWorkerCompleted);
    //
    // startBtn
    //
    this.startBtn.Location = new System.Drawing.Point(12, 12);
    this.startBtn.Name = "startBtn";
    this.startBtn.Size = new System.Drawing.Size(75, 23);
    this.startBtn.TabIndex = 0;
    this.startBtn.Text = "Start";
    this.startBtn.Click += new System.EventHandler(this.startBtn_Click);
    //
    // cancelBtn
    //
    this.cancelBtn.Location = new System.Drawing.Point(94, 11);
    this.cancelBtn.Name = "cancelBtn";
    this.cancelBtn.Size = new System.Drawing.Size(75, 23);
    this.cancelBtn.TabIndex = 1;
    this.cancelBtn.Text = "Cancel";
}

```

```

        this.cancelBtn.Click += new System.EventHandler(this.cancelBtn_Click);
        //
        // Form1
        //
        this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.ClientSize = new System.Drawing.Size(183, 49);
        this.Controls.Add(this.cancelBtn);
        this.Controls.Add(this.startBtn);
        this.Name = "Form1";
        this.Text = "Form1";
        this.ResumeLayout(false);

    }

    #endregion

    private System.ComponentModel.BackgroundWorker backgroundWorker1;
    private System.Windows.Forms.Button startBtn;
    private System.Windows.Forms.Button cancelBtn;
}

public class Program
{
    private Program()
    {
    }

    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.Run(new Form1());
    }
}
}

```

```

Imports System
Imports System.ComponentModel
Imports System.Drawing
Imports System.Threading
Imports System.Windows.Forms

Public Class Form1
    Inherits Form

    Public Sub New()
        InitializeComponent()
    End Sub

    Private Sub backgroundWorker1_DoWork( _
        sender As Object, e As DoWorkEventArgs) _
        Handles backgroundWorker1.DoWork

        ' Do not access the form's BackgroundWorker reference directly.
        ' Instead, use the reference provided by the sender parameter.
        Dim bw As BackgroundWorker = CType(sender, BackgroundWorker)

        ' Extract the argument.
        Dim arg As Integer = Fix(e.Argument)

        ' Start the time-consuming operation.
        e.Result = TimeConsumingOperation(bw, arg)
    End Sub

```

```

' If the operation was canceled by the user,
' set the DoWorkEventArgs.Cancel property to true.
If bw.CancellationPending Then
    e.Cancel = True
End If

End Sub

' This event handler demonstrates how to interpret
' the outcome of the asynchronous operation implemented
' in the DoWork event handler.
Private Sub backgroundWorker1_RunWorkerCompleted( _
    sender As Object, e As RunWorkerCompletedEventArgs) _
Handles backgroundWorker1.RunWorkerCompleted

    If e.Cancelled Then
        ' The user canceled the operation.
        MessageBox.Show("Operation was canceled")
    ElseIf (e.Error IsNot Nothing) Then
        ' There was an error during the operation.
        Dim msg As String = String.Format("An error occurred: {0}", e.Error.Message)
        MessageBox.Show(msg)
    Else
        ' The operation completed normally.
        Dim msg As String = String.Format("Result = {0}", e.Result)
        MessageBox.Show(msg)
    End If
End Sub

' This method models an operation that may take a long time
' to run. It can be cancelled, it can raise an exception,
' or it can exit normally and return a result. These outcomes
' are chosen randomly.
Private Function TimeConsumingOperation( _
    bw As BackgroundWorker, _
    sleepPeriod As Integer) As Integer

    Dim result As Integer = 0

    Dim rand As New Random()

    While Not bw.CancellationPending
        Dim [exit] As Boolean = False

        Select Case rand.Next(3)
            ' Raise an exception.
            Case 0
                Throw New Exception("An error condition occurred.")
                Exit While

            ' Sleep for the number of milliseconds
            ' specified by the sleepPeriod parameter.
            Case 1
                Thread.Sleep(sleepPeriod)
                Exit While

            ' Exit and return normally.
            Case 2
                result = 23
                [exit] = True
                Exit While

            Case Else
                Exit While
        End Select

        If [exit] Then
            Exit While
        End If
    End While
End Function

```

```

        End If
    End While

    Return result
End Function

Private Sub startButton_Click(ByVal sender As Object, ByVal e As EventArgs) Handles startBtn.Click
    Me.backgroundWorker1.RunWorkerAsync(2000)
End Sub

Private Sub cancelButton_Click(ByVal sender As Object, ByVal e As EventArgs) Handles cancelBtn.Click
    Me.backgroundWorker1.CancelAsync()
End Sub

' Required designer variable.
Private components As System.ComponentModel.IContainer = Nothing

' Clean up any resources being used.
' <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
Protected Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing AndAlso (components IsNot Nothing) Then
        components.Dispose()
    End If
    MyBase.Dispose(disposing)
End Sub 'Dispose

#Region "Windows Form Designer generated code"

' Required method for Designer support - do not modify
' the contents of this method with the code editor.
Private Sub InitializeComponent()
    Me.backgroundWorker1 = New System.ComponentModel.BackgroundWorker()
    Me.startBtn = New System.Windows.Forms.Button()
    Me.cancelBtn = New System.Windows.Forms.Button()
    Me.SuspendLayout()

    ' backgroundWorker1

    Me.backgroundWorker1.WorkerSupportsCancellation = True
    '

    ' startButton

    Me.startBtn.Location = New System.Drawing.Point(12, 12)
    Me.startBtn.Name = "startButton"
    Me.startBtn.Size = New System.Drawing.Size(75, 23)
    Me.startBtn.TabIndex = 0
    Me.startBtn.Text = "Start"
    '

    ' cancelButton

    Me.cancelBtn.Location = New System.Drawing.Point(94, 11)
    Me.cancelBtn.Name = "cancelButton"
    Me.cancelBtn.Size = New System.Drawing.Size(75, 23)
    Me.cancelBtn.TabIndex = 1
    Me.cancelBtn.Text = "Cancel"
    '

    ' Form1

    Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0F, 13.0F)
    Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
    Me.ClientSize = New System.Drawing.Size(183, 49)
    Me.Controls.Add(cancelBtn)
    Me.Controls.Add(startBtn)
    Me.Name = "Form1"
    Me.Text = "Form1"
    Me.ResumeLayout(False)
End Sub

```

```

End Sub

#End Region

Private WithEvents backgroundWorker1 As System.ComponentModel.BackgroundWorker
Private WithEvents startBtn As System.Windows.Forms.Button
Private WithEvents cancelBtn As System.Windows.Forms.Button
End Class

Public Class Program

    Private Sub New()

        End Sub

        ' The main entry point for the application.
        <STAThread()> _
        Shared Sub Main()
            Application.EnableVisualStyles()
            Application.Run(New Form1())
        End Sub

    End Class

```

Compiling the Code

This example requires:

- References to the System, System.Drawing and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[BackgroundWorker](#)
[DoWorkEventArgs](#)
[How to: Implement a Form That Uses a Background Operation](#)
[BackgroundWorker Component](#)

How to: Download a File in the Background

5/4/2018 • 9 min to read • [Edit Online](#)

Downloading a file is a common task, and it is often useful to run this potentially time-consuming operation on a separate thread. Use the [BackgroundWorker](#) component to accomplish this task with very little code.

Example

The following code example demonstrates how to use a [BackgroundWorker](#) component to load an XML file from a URL. When the user clicks the **Download** button, the [Click](#) event handler calls the [RunWorkerAsync](#) method of a [BackgroundWorker](#) component to start the download operation. The button is disabled for the duration of the download, and then enabled when the download is complete. A [MessageBox](#) displays the contents of the file.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Threading;
using System.Windows.Forms;
using System.Xml;

public class Form1 : Form
{
    private BackgroundWorker backgroundWorker1;
    private Button downloadButton;
    private ProgressBar progressBar1;
    private XmlDocument document = null;

    public Form1()
    {
        InitializeComponent();

        // Instantiate BackgroundWorker and attach handlers to its
        // DoWork and RunWorkerCompleted events.
        backgroundWorker1 = new System.ComponentModel.BackgroundWorker();
        backgroundWorker1.DoWork += new
System.ComponentModel.DoWorkEventHandler(this.backgroundWorker1_DoWork);
        backgroundWorker1.RunWorkerCompleted += new
System.ComponentModel.RunWorkerCompletedEventHandler(this.backgroundWorker1_RunWorkerCompleted);
    }

    private void downloadButton_Click(object sender, EventArgs e)
    {
        // Start the download operation in the background.
        this.backgroundWorker1.RunWorkerAsync();

        // Disable the button for the duration of the download.
        this.downloadButton.Enabled = false;

        // Once you have started the background thread you
        // can exit the handler and the application will
        // wait until the RunWorkerCompleted event is raised.

        // Or if you want to do something else in the main thread,
        // such as update a progress bar, you can do so in a loop
        // while checking IsBusy to see if the background task is
        // still running.

        while (this.backgroundWorker1.IsBusy)
    }
```

```

        progressBar1.Increment(1);
        // Keep UI messages moving, so the form remains
        // responsive during the asynchronous operation.
        Application.DoEvents();
    }

}

private void backgroundWorker1_DoWork(
    object sender,
    DoWorkEventArgs e)
{
    document = new XmlDocument();

    // Uncomment the following line to
    // simulate a noticeable latency.
    //Thread.Sleep(5000);

    // Replace this file name with a valid file name.
    document.Load(@"http://www.tailspintoys.com/sample.xml");
}

private void backgroundWorker1_RunWorkerCompleted(
    object sender,
    RunWorkerCompletedEventArgs e)
{
    // Set progress bar to 100% in case it's not already there.
    progressBar1.Value = 100;

    if (e.Error == null)
    {
        MessageBox.Show(document.InnerXml, "Download Complete");
    }
    else
    {
        MessageBox.Show(
            "Failed to download file",
            "Download failed",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }

    // Enable the download button and reset the progress bar.
    this.downloadButton.Enabled = true;
    progressBar1.Value = 0;
}

#region Windows Form Designer generated code

/// <summary>
/// Required designer variable.
/// </summary>
private System.ComponentModel.IContainer components = null;

/// <summary>
/// Clean up any resources being used.
/// </summary>
/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
protected override void Dispose(bool disposing)
{
    if (disposing && (components != null))
    {
        components.Dispose();
    }
    base.Dispose(disposing);
}

/// <summary>
/// Required method for Designer support
/// </summary>

```

```

    ...
    private void InitializeComponent()
    {
        this.downloadButton = new System.Windows.Forms.Button();
        this.progressBar1 = new System.Windows.Forms.ProgressBar();
        this.SuspendLayout();
        //
        // downloadButton
        //
        this.downloadButton.Location = new System.Drawing.Point(12, 12);
        this.downloadButton.Name = "downloadButton";
        this.downloadButton.Size = new System.Drawing.Size(100, 23);
        this.downloadButton.TabIndex = 0;
        this.downloadButton.Text = "Download file";
        this.downloadButton.UseVisualStyleBackColor = true;
        this.downloadButton.Click += new System.EventHandler(this.downloadButton_Click);
        //
        // progressBar1
        //
        this.progressBar1.Location = new System.Drawing.Point(12, 50);
        this.progressBar1.Name = "progressBar1";
        this.progressBar1.Size = new System.Drawing.Size(100, 26);
        this.progressBar1.TabIndex = 1;
        //
        // Form1
        //
        this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.ClientSize = new System.Drawing.Size(133, 104);
        this.Controls.Add(this.progressBar1);
        this.Controls.Add(this.downloadButton);
        this.Name = "Form1";
        this.Text = "Form1";
        this.ResumeLayout(false);

    }

    #endregion
}

static class Program
{
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.Run(new Form1());
    }
}

```

```

Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Drawing
Imports System.Threading
Imports System.Windows.Forms
Imports System.Xml

Public Class Form1
    Inherits Form

    Private WithEvents downloadButton As Button
    Private WithEvents progressBar1 As ProgressBar
    Private WithEvents backgroundWorker1 As BackgroundWorker
    Private document As XmlDocument = Nothing

```

```

Public Sub New()
    InitializeComponent()
    Me.backgroundWorker1 = New System.ComponentModel.BackgroundWorker()
End Sub

Private Sub downloadButton_Click( _
    ByVal sender As Object, _
    ByVal e As EventArgs) _
    Handles downloadButton.Click

    ' Start the download operation in the background.
    Me.backgroundWorker1.RunWorkerAsync()

    ' Disable the button for the duration of the download.
    Me.downloadButton.Enabled = False

    ' Once you have started the background thread you
    ' can exit the handler and the application will
    ' wait until the RunWorkerCompleted event is raised.

    ' If you want to do something else in the main thread,
    ' such as update a progress bar, you can do so in a loop
    ' while checking IsBusy to see if the background task is
    ' still running.

    While Me.backgroundWorker1.IsBusy
        progressBar1.Increment(1)
        ' Keep UI messages moving, so the form remains
        ' responsive during the asynchronous operation.
        Application.DoEvents()
    End While
End Sub

Private Sub backgroundWorker1_DoWork( _
    ByVal sender As Object, _
    ByVal e As DoWorkEventArgs) _
    Handles backgroundWorker1.DoWork

    document = New XmlDocument()

    ' Replace this file name with a valid file name.
    document.Load("http://www.tailspintoyos.com/sample.xml")

    ' Uncomment the following line to
    ' simulate a noticeable latency.
    'Thread.Sleep(5000);
End Sub

Private Sub backgroundWorker1_RunWorkerCompleted( _
    ByVal sender As Object, _
    ByVal e As RunWorkerCompletedEventArgs) _
    Handles backgroundWorker1.RunWorkerCompleted

    ' Set progress bar to 100% in case it isn't already there.
    progressBar1.Value = 100

    If e.Error Is Nothing Then
        MessageBox.Show(document.InnerXml, "Download Complete")
    Else
        MessageBox.Show("Failed to download file", "Download failed", MessageBoxButtons.OK,
        MessageBoxIcon.Error)
    End If

    ' Enable the download button and reset the progress bar.
    Me.downloadButton.Enabled = True
    progressBar1.Value = 0
End Sub

#Region "Windows Form Designer generated code"

```

```

' Required designer variable.
Private components As System.ComponentModel.IContainer = Nothing

' Clean up any resources being used.
<param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
Protected Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing AndAlso (components IsNot Nothing) Then
        components.Dispose()
    End If
    MyBase.Dispose(disposing)
End Sub

' Required method for Designer support - do not modify
' the contents of this method with the code editor.
Private Sub InitializeComponent()
    Me.downloadButton = New System.Windows.Forms.Button
    Me.progressBar1 = New System.Windows.Forms.ProgressBar
    Me.SuspendLayout()

    'downloadButton

    Me.downloadButton.Location = New System.Drawing.Point(12, 12)
    Me.downloadButton.Name = "downloadButton"
    Me.downloadButton.Size = New System.Drawing.Size(100, 23)
    Me.downloadButton.TabIndex = 0
    Me.downloadButton.Text = "Download file"
    Me.downloadButton.UseVisualStyleBackColor = True
    '

    'progressBar1

    Me.progressBar1.Location = New System.Drawing.Point(12, 50)
    Me.progressBar1.Name = "progressBar1"
    Me.progressBar1.Size = New System.Drawing.Size(100, 26)
    Me.progressBar1.TabIndex = 1
    '

    'Form1

    Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)
    Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
    Me.ClientSize = New System.Drawing.Size(136, 104)
    Me.Controls.Add(Me.downloadButton)
    Me.Controls.Add(Me.progressBar1)
    Me.Name = "Form1"
    Me.Text = "Form1"
    Me.ResumeLayout(False)

End Sub

#End Region
End Class

Public Class Program

    ' The main entry point for the application.
    <STAThread()> _
    Shared Sub Main()
        Application.EnableVisualStyles()
        Application.Run(New Form1())
    End Sub
End Class

```

Downloading the file

The file is downloaded on the [BackgroundWorker](#) component's worker thread, which runs the [DoWork](#) event handler. This thread starts when your code calls the [RunWorkerAsync](#) method.

```
private void backgroundWorker1_DoWork(
    object sender,
    DoWorkEventArgs e)
{
    document = new XmlDocument();

    // Uncomment the following line to
    // simulate a noticeable latency.
    //Thread.Sleep(5000);

    // Replace this file name with a valid file name.
    document.Load(@"http://www.tailspintoys.com/sample.xml");
}
```

```
Private Sub backgroundWorker1_DoWork( _
    ByVal sender As Object, _
    ByVal e As DoWorkEventArgs) _
Handles backgroundWorker1.DoWork

    document = New XmlDocument()

    ' Replace this file name with a valid file name.
    document.Load("http://www.tailspintoys.com/sample.xml")

    ' Uncomment the following line to
    ' simulate a noticeable latency.
    'Thread.Sleep(5000);
End Sub
```

Waiting for a BackgroundWorker to finish

The `downloadButton_Click` event handler demonstrates how to wait for a [BackgroundWorker](#) component to finish its asynchronous task.

If you only want the application to respond to events and do not want to do any work in the main thread while you wait for the background thread to complete, just exit the handler.

If you want to continue doing work in the main thread, use the [IsBusy](#) property to determine whether the [BackgroundWorker](#) thread is still running. In the example, a progress bar is updated while the download is processing. Be sure to call the [Application.DoEvents](#) method to keep the UI responsive.

```

private void downloadButton_Click(object sender, EventArgs e)
{
    // Start the download operation in the background.
    this.backgroundWorker1.RunWorkerAsync();

    // Disable the button for the duration of the download.
    this.downloadButton.Enabled = false;

    // Once you have started the background thread you
    // can exit the handler and the application will
    // wait until the RunWorkerCompleted event is raised.

    // Or if you want to do something else in the main thread,
    // such as update a progress bar, you can do so in a loop
    // while checking IsBusy to see if the background task is
    // still running.

    while (this.backgroundWorker1.IsBusy)
    {
        progressBar1.Increment(1);
        // Keep UI messages moving, so the form remains
        // responsive during the asynchronous operation.
        Application.DoEvents();
    }
}

```

```

Private Sub downloadButton_Click( _
    ByVal sender As Object, _
    ByVal e As EventArgs) _
Handles downloadButton.Click

    ' Start the download operation in the background.
    Me.backgroundWorker1.RunWorkerAsync()

    ' Disable the button for the duration of the download.
    Me.downloadButton.Enabled = False

    ' Once you have started the background thread you
    ' can exit the handler and the application will
    ' wait until the RunWorkerCompleted event is raised.

    ' If you want to do something else in the main thread,
    ' such as update a progress bar, you can do so in a loop
    ' while checking IsBusy to see if the background task is
    ' still running.
    While Me.backgroundWorker1.IsBusy
        progressBar1.Increment(1)
        ' Keep UI messages moving, so the form remains
        ' responsive during the asynchronous operation.
        Application.DoEvents()
    End While
End Sub

```

Displaying the result

The `backgroundWorker1_RunWorkerCompleted` method handles the `RunWorkerCompleted` event and is called when the background operation is completed. This method first checks the `EventArgs.Error` property. If `EventArgs.Error` is `null`, then this method displays the contents of the file. It then enables the download button, which was disabled when the download began, and it resets the progress bar.

```

private void backgroundWorker1_RunWorkerCompleted(
    object sender,
    RunWorkerCompletedEventArgs e)
{
    // Set progress bar to 100% in case it's not already there.
    progressBar1.Value = 100;

    if (e.Error == null)
    {
        MessageBox.Show(document.InnerXml, "Download Complete");
    }
    else
    {
        MessageBox.Show(
            "Failed to download file",
            "Download failed",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }

    // Enable the download button and reset the progress bar.
    this.downloadButton.Enabled = true;
    progressBar1.Value = 0;
}

```

```

Private Sub backgroundWorker1_RunWorkerCompleted( _
    ByVal sender As Object, _
    ByVal e As RunWorkerCompletedEventArgs) _
Handles backgroundWorker1.RunWorkerCompleted

    ' Set progress bar to 100% in case it isn't already there.
    progressBar1.Value = 100

    If e.Error Is Nothing Then
        MessageBox.Show(document.InnerXml, "Download Complete")
    Else
        MessageBox.Show("Failed to download file", "Download failed", MessageBoxButtons.OK,
        MessageBoxIcon.Error)
    End If

    ' Enable the download button and reset the progress bar.
    Me.downloadButton.Enabled = True
    progressBar1.Value = 0
End Sub

```

Compiling the Code

This example requires:

- References to the System.Drawing, System.Windows.Forms, and System.Xml assemblies.

For information about building this example from the command line for visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

Robust Programming

Always check the [AsyncCompletedEventArgs.Error](#) property in your [RunWorkerCompleted](#) event handler before attempting to access the [RunWorkerCompletedEventArgs.Result](#) property or any other object that may have been affected by the [DoWork](#) event handler.

See Also

[BackgroundWorker](#)

[How to: Run an Operation in the Background](#)

[How to: Implement a Form That Uses a Background Operation](#)

How to: Implement a Form That Uses a Background Operation

5/4/2018 • 14 min to read • [Edit Online](#)

The following example program creates a form that calculates Fibonacci numbers. The calculation runs on a thread that is separate from the user interface thread, so the user interface continues to run without delays as the calculation proceeds.

There is extensive support for this task in Visual Studio.

Also see [Walkthrough: Implementing a Form That Uses a Background Operation](#).

Example

```
#using <System.Drawing.dll>
#using <System.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Collections;
using namespace System::ComponentModel;
using namespace System::Drawing;
using namespace System::Threading;
using namespace System::Windows::Forms;

public ref class FibonacciForm: public System::Windows::Forms::Form
{
private:

    int numberToCompute;
    int highestPercentageReached;

    System::Windows::Forms::NumericUpDown^ numericUpDown1;
    System::Windows::Forms::Button^ startAsyncButton;
    System::Windows::Forms::Button^ cancelAsyncButton;
    System::Windows::Forms::ProgressBar^ progressBar1;
    System::Windows::Forms::Label ^ resultLabel;
    System::ComponentModel::BackgroundWorker^ backgroundWorker1;

public:
    FibonacciForm()
    {
        InitializeComponent();
        numberToCompute = highestPercentageReached = 0;
        InitializeBackgroundWorker();
    }

private:

    // Set up the BackgroundWorker object by
    // attaching event handlers.
    void InitializeBackgroundWorker()
    {
        backgroundWorker1->DoWork += gcnew DoWorkEventHandler( this, &FibonacciForm::backgroundWorker1_DoWork );
        backgroundWorker1->RunWorkerCompleted += gcnew RunWorkerCompletedEventHandler( this,
&FibonacciForm::backgroundWorker1_RunWorkerCompleted );
        backgroundWorker1->ProgressChanged += gcnew ProgressChangedEventHandler( this,
```

```

&FibonacciForm::backgroundWorker1_ProgressChanged );
}

void startAsyncButton_Click( System::Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    // Reset the text in the result label.
    resultLabel->Text = String::Empty;

    // Disable the UpDown control until
    // the asynchronous operation is done.
    this->numericUpDown1->Enabled = false;

    // Disable the Start button until
    // the asynchronous operation is done.
    this->startAsyncButton->Enabled = false;

    // Enable the Cancel button while
    // the asynchronous operation runs.
    this->cancelAsyncButton->Enabled = true;

    // Get the value from the UpDown control.
    numberToCompute = (int)numericUpDown1->Value;

    // Reset the variable for percentage tracking.
    highestPercentageReached = 0;

    // Start the asynchronous operation.
    backgroundWorker1->RunWorkerAsync( numberToCompute );
}

void cancelAsyncButton_Click( System::Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    // Cancel the asynchronous operation.
    this->backgroundWorker1->CancelAsync();

    // Disable the Cancel button.
    cancelAsyncButton->Enabled = false;
}

// This event handler is where the actual,
// potentially time-consuming work is done.
void backgroundWorker1_DoWork( Object^ sender, DoWorkEventArgs^ e )
{
    // Get the BackgroundWorker that raised this event.
    BackgroundWorker^ worker = dynamic_cast<BackgroundWorker^>(sender);

    // Assign the result of the computation
    // to the Result property of the DoWorkEventArgs
    // object. This is will be available to the
    // RunWorkerCompleted eventhandler.
    e->Result = ComputeFibonacci( safe_cast<Int32>(e->Argument), worker, e );
}

// This event handler deals with the results of the
// background operation.
void backgroundWorker1_RunWorkerCompleted( Object^ /*sender*/, RunWorkerCompletedEventArgs^ e )
{
    // First, handle the case where an exception was thrown.
    if ( e->Error != nullptr )
    {
        MessageBox::Show( e->Error->Message );
    }
    else
    if ( e->Cancelled )
    {
        // Next, handle the case where the user cancelled
        // the operation.
        // Note that due to a race condition in

```

```

// the DoWork event handler, the Cancelled
// flag may not have been set, even though
// CancelAsync was called.
resultLabel->Text = "Cancelled";
}
else
{
    // Finally, handle the case where the operation
    // succeeded.
    resultLabel->Text = e->Result->ToString();
}

// Enable the UpDown control.
this->numericUpDown1->Enabled = true;

// Enable the Start button.
startAsyncButton->Enabled = true;

// Disable the Cancel button.
cancelAsyncButton->Enabled = false;
}

// This event handler updates the progress bar.
void backgroundWorker1_ProgressChanged( Object^ /*sender*/, ProgressChangedEventArgs^ e )
{
    this->progressBar1->Value = e->ProgressPercentage;
}

// This is the method that does the actual work. For this
// example, it computes a Fibonacci number and
// reports progress as it does its work.
long ComputeFibonacci( int n, BackgroundWorker^ worker, DoWorkEventArgs ^ e )
{
    // The parameter n must be >= 0 and <= 91.
    // Fib(n), with n > 91, overflows a long.
    if ( (n < 0) || (n > 91) )
    {
        throw gcnew ArgumentException( "value must be >= 0 and <= 91","n" );
    }

    long result = 0;

    // Abort the operation if the user has cancelled.
    // Note that a call to CancelAsync may have set
    // CancellationPending to true just after the
    // last invocation of this method exits, so this
    // code will not have the opportunity to set the
    // DoWorkEventArgs.Cancel flag to true. This means
    // that RunWorkerCompletedEventArgs.Cancelled will
    // not be set to true in your RunWorkerCompleted
    // event handler. This is a race condition.
    if ( worker->CancellationPending )
    {
        e->Cancel = true;
    }
    else
    {
        if ( n < 2 )
        {
            result = 1;
        }
        else
        {
            result = ComputeFibonacci( n - 1, worker, e ) + ComputeFibonacci( n - 2, worker, e );
        }

        // Report progress as a percentage of the total task.
        int percentComplete = (int)((float)n / (float)numberToCompute * 100);
        if ( percentComplete > highestPercentageReached )
    }
}

```

```

    }

    highestPercentageReached = percentComplete;
    worker->ReportProgress( percentComplete );
}
}

return result;
}

void InitializeComponent()
{
    this->nemonicUpDown1 = gcnew System::Windows::Forms::NumericUpDown;
    this->startAsyncButton = gcnew System::Windows::Forms::Button;
    this->cancelAsyncButton = gcnew System::Windows::Forms::Button;
    this->resultLabel = gcnew System::Windows::Forms::Label;
    this->progressBar1 = gcnew System::Windows::Forms::ProgressBar;
    this->backgroundWorker1 = gcnew System::ComponentModel::BackgroundWorker;
    (dynamic_cast<System::ComponentModel::ISupportInitialize^>(this->nemonicUpDown1))->BeginInit();
    this->SuspendLayout();

    //

    // numericUpDown1
    //

    this->nemonicUpDown1->Location = System::Drawing::Point( 16, 16 );
    array<Int32>^temp0 = {91,0,0,0};
    this->nemonicUpDown1->Maximum = System::Decimal( temp0 );
    array<Int32>^temp1 = {1,0,0,0};
    this->nemonicUpDown1->Minimum = System::Decimal( temp1 );
    this->nemonicUpDown1->Name = "nemonicUpDown1";
    this->nemonicUpDown1->Size = System::Drawing::Size( 80, 20 );
    this->nemonicUpDown1->TabIndex = 0;
    array<Int32>^temp2 = {1,0,0,0};
    this->nemonicUpDown1->Value = System::Decimal( temp2 );

    //

    // startAsyncButton
    //

    this->startAsyncButton->Location = System::Drawing::Point( 16, 72 );
    this->startAsyncButton->Name = "startAsyncButton";
    this->startAsyncButton->Size = System::Drawing::Size( 120, 23 );
    this->startAsyncButton->TabIndex = 1;
    this->startAsyncButton->Text = "Start Async";
    this->startAsyncButton->Click += gcnew System::EventHandler( this,
&FibonacciForm::startAsyncButton_Click );

    //

    // cancelAsyncButton
    //

    this->cancelAsyncButton->Enabled = false;
    this->cancelAsyncButton->Location = System::Drawing::Point( 153, 72 );
    this->cancelAsyncButton->Name = "cancelAsyncButton";
    this->cancelAsyncButton->Size = System::Drawing::Size( 119, 23 );
    this->cancelAsyncButton->TabIndex = 2;
    this->cancelAsyncButton->Text = "Cancel Async";
    this->cancelAsyncButton->Click += gcnew System::EventHandler( this,
&FibonacciForm::cancelAsyncButton_Click );

    //

    // resultLabel
    //

    this->resultLabel->BorderStyle = System::Windows::Forms::BorderStyle::Fixed3D;
    this->resultLabel->Location = System::Drawing::Point( 112, 16 );
    this->resultLabel->Name = "resultLabel";
    this->resultLabel->Size = System::Drawing::Size( 160, 23 );
    this->resultLabel->TabIndex = 3;
    this->resultLabel->Text = "(no result)";
    this->resultLabel->ContentAlignment = System::Drawing::ContentAlignment::MiddleCenter;

    //

```

```

    //
    // progressBar1
    //
    this->progressBar1->Location = System::Drawing::Point( 18, 48 );
    this->progressBar1->Name = "progressBar1";
    this->progressBar1->Size = System::Drawing::Size( 256, 8 );
    this->progressBar1->Step = 2;
    this->progressBar1->TabIndex = 4;

    //
    // backgroundWorker1
    //
    this->backgroundWorker1->WorkerReportsProgress = true;
    this->backgroundWorker1->WorkerSupportsCancellation = true;

    //
    // FibonacciForm
    //
    this->ClientSize = System::Drawing::Size( 292, 118 );
    this->Controls->Add( this->progressBar1 );
    this->Controls->Add( this->resultLabel );
    this->Controls->Add( this->cancelAsyncButton );
    this->Controls->Add( this->startAsyncButton );
    this->Controls->Add( this->numericUpDown1 );
    this->Name = "FibonacciForm";
    this->Text = "Fibonacci Calculator";
    (dynamic_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown1))->EndInit();
    this->ResumeLayout( false );
}

};

[STAThread]
int main()
{
    Application::Run( gcnew FibonacciForm );
}

```

```

using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Threading;
using System.Windows.Forms;

namespace BackgroundWorkerExample
{
    public class FibonacciForm : System.Windows.Forms.Form
    {
        private int numberToCompute = 0;
        private int highestPercentageReached = 0;

        private System.Windows.Forms.NumericUpDown numericUpDown1;
        private System.Windows.Forms.Button startAsyncButton;
        private System.Windows.Forms.Button cancelAsyncButton;
        private System.Windows.Forms.ProgressBar progressBar1;
        private System.Windows.Forms.Label resultLabel;
        private System.ComponentModel.BackgroundWorker backgroundWorker1;

        public FibonacciForm()
        {
            InitializeComponent();

            InitializeBackgroundWorker();
        }

        // Set up the BackgroundWorker object by
        // attaching event handlers.
        private void InitializeBackgroundWorker()

```

```

        }

        backgroundWorker1.DoWork +=
            new DoWorkEventHandler(backgroundWorker1_DoWork);
        backgroundWorker1.RunWorkerCompleted +=
            new RunWorkerCompletedEventHandler(
                backgroundWorker1_RunWorkerCompleted);
        backgroundWorker1.ProgressChanged +=
            new ProgressChangedEventHandler(
                backgroundWorker1_ProgressChanged);
    }

    private void startAsyncButton_Click(System.Object sender,
        System.EventArgs e)
    {
        // Reset the text in the result label.
        resultLabel.Text = String.Empty;

        // Disable the UpDown control until
        // the asynchronous operation is done.
        this.numericUpDown1.Enabled = false;

        // Disable the Start button until
        // the asynchronous operation is done.
        this.startAsyncButton.Enabled = false;

        // Enable the Cancel button while
        // the asynchronous operation runs.
        this.cancelAsyncButton.Enabled = true;

        // Get the value from the UpDown control.
        numberToCompute = (int)numericUpDown1.Value;

        // Reset the variable for percentage tracking.
        highestPercentageReached = 0;

        // Start the asynchronous operation.
        backgroundWorker1.RunWorkerAsync(numberToCompute);
    }

    private void cancelAsyncButton_Click(System.Object sender,
        System.EventArgs e)
    {
        // Cancel the asynchronous operation.
        this.backgroundWorker1.CancelAsync();

        // Disable the Cancel button.
        cancelAsyncButton.Enabled = false;
    }

    // This event handler is where the actual,
    // potentially time-consuming work is done.
    private void backgroundWorker1_DoWork(object sender,
        DoWorkEventArgs e)
    {
        // Get the BackgroundWorker that raised this event.
        BackgroundWorker worker = sender as BackgroundWorker;

        // Assign the result of the computation
        // to the Result property of the DoWorkEventArgs
        // object. This is will be available to the
        // RunWorkerCompleted eventhandler.
        e.Result = ComputeFibonacci((int)e.Argument, worker, e);
    }

    // This event handler deals with the results of the
    // background operation.
    private void backgroundWorker1_RunWorkerCompleted(
        object sender, RunWorkerCompletedEventArgs e)
    {

```

```

        // First, handle the case where an exception was thrown.
        if (e.Error != null)
        {
            MessageBox.Show(e.Error.Message);
        }
        else if (e.Cancelled)
        {
            // Next, handle the case where the user canceled
            // the operation.
            // Note that due to a race condition in
            // the DoWork event handler, the Cancelled
            // flag may not have been set, even though
            // CancelAsync was called.
            resultLabel.Text = "Canceled";
        }
        else
        {
            // Finally, handle the case where the operation
            // succeeded.
            resultLabel.Text = e.Result.ToString();
        }

        // Enable the UpDown control.
        this.numericUpDown1.Enabled = true;

        // Enable the Start button.
        startAsyncButton.Enabled = true;

        // Disable the Cancel button.
        cancelAsyncButton.Enabled = false;
    }

    // This event handler updates the progress bar.
    private void backgroundWorker1_ProgressChanged(object sender,
        ProgressChangedEventArgs e)
    {
        this.progressBar1.Value = e.ProgressPercentage;
    }

    // This is the method that does the actual work. For this
    // example, it computes a Fibonacci number and
    // reports progress as it does its work.
    long ComputeFibonacci(int n, BackgroundWorker worker, DoWorkEventArgs e)
    {
        // The parameter n must be >= 0 and <= 91.
        // Fib(n), with n > 91, overflows a long.
        if ((n < 0) || (n > 91))
        {
            throw new ArgumentException(
                "value must be >= 0 and <= 91", "n");
        }

        long result = 0;

        // Abort the operation if the user has canceled.
        // Note that a call to CancelAsync may have set
        // CancellationPending to true just after the
        // last invocation of this method exits, so this
        // code will not have the opportunity to set the
        // DoWorkEventArgs.Cancel flag to true. This means
        // that RunWorkerCompletedEventArgs.Cancelled will
        // not be set to true in your RunWorkerCompleted
        // event handler. This is a race condition.

        if (worker.CancellationPending)
        {
            e.Cancel = true;
        }
    }
}

```

```

        else
    {
        if (n < 2)
        {
            result = 1;
        }
        else
        {
            result = ComputeFibonacci(n - 1, worker, e) +
                ComputeFibonacci(n - 2, worker, e);
        }

        // Report progress as a percentage of the total task.
        int percentComplete =
            (int)((float)n / (float)numberToCompute * 100);
        if (percentComplete > highestPercentageReached)
        {
            highestPercentageReached = percentComplete;
            worker.ReportProgress(percentComplete);
        }
    }

    return result;
}

#region Windows Form Designer generated code

private void InitializeComponent()
{
    this.numericUpDown1 = new System.Windows.Forms.NumericUpDown();
    this.startAsyncButton = new System.Windows.Forms.Button();
    this.cancelAsyncButton = new System.Windows.Forms.Button();
    this.resultLabel = new System.Windows.Forms.Label();
    this.progressBar1 = new System.Windows.Forms.ProgressBar();
    this.backgroundWorker1 = new System.ComponentModel.BackgroundWorker();
    ((System.ComponentModel.ISupportInitialize)(this.numericUpDown1)).BeginInit();
    this.SuspendLayout();
    //
    // numericUpDown1
    //
    this.numericUpDown1.Location = new System.Drawing.Point(16, 16);
    this.numericUpDown1.Maximum = new System.Decimal(new int[] {
        91,
        0,
        0,
        0});
    this.numericUpDown1.Minimum = new System.Decimal(new int[] {
        1,
        0,
        0,
        0});
    this.numericUpDown1.Name = "numericUpDown1";
    this.numericUpDown1.Size = new System.Drawing.Size(80, 20);
    this.numericUpDown1.TabIndex = 0;
    this.numericUpDown1.Value = new System.Decimal(new int[] {
        1,
        0,
        0,
        0});
    //
    // startAsyncButton
    //
    this.startAsyncButton.Location = new System.Drawing.Point(16, 72);
    this.startAsyncButton.Name = "startAsyncButton";
    this.startAsyncButton.Size = new System.Drawing.Size(120, 23);
    this.startAsyncButton.TabIndex = 1;
    this.startAsyncButton.Text = "Start Async";
    this.startAsyncButton.Click += new System.EventHandler(this.startAsyncButton_Click);
}

```

```

// cancelAsyncButton
//
this.cancelAsyncButton.Enabled = false;
this.cancelAsyncButton.Location = new System.Drawing.Point(153, 72);
this.cancelAsyncButton.Name = "cancelAsyncButton";
this.cancelAsyncButton.Size = new System.Drawing.Size(119, 23);
this.cancelAsyncButton.TabIndex = 2;
this.cancelAsyncButton.Text = "Cancel Async";
this.cancelAsyncButton.Click += new System.EventHandler(this.cancelAsyncButton_Click);
//
// resultLabel
//
this.resultLabel.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D;
this.resultLabel.Location = new System.Drawing.Point(112, 16);
this.resultLabel.Name = "resultLabel";
this.resultLabel.Size = new System.Drawing.Size(160, 23);
this.resultLabel.TabIndex = 3;
this.resultLabel.Text = "(no result)";
this.resultLabel.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;
//
// progressBar1
//
this.progressBar1.Location = new System.Drawing.Point(18, 48);
this.progressBar1.Name = "progressBar1";
this.progressBar1.Size = new System.Drawing.Size(256, 8);
this.progressBar1.Step = 2;
this.progressBar1.TabIndex = 4;
//
// backgroundWorker1
//
this.backgroundWorker1.WorkerReportsProgress = true;
this.backgroundWorker1.WorkerSupportsCancellation = true;
//
// FibonacciForm
//
this.ClientSize = new System.Drawing.Size(292, 118);
this.Controls.Add(this.progressBar1);
this.Controls.Add(this.resultLabel);
this.Controls.Add(this.cancelAsyncButton);
this.Controls.Add(this.startAsyncButton);
this.Controls.Add(this.numericUpDown1);
this.Name = "FibonacciForm";
this.Text = "Fibonacci Calculator";
((System.ComponentModel.ISupportInitialize)(this.numericUpDown1)).EndInit();
this.ResumeLayout(false);

}
#endif

[STAThread]
static void Main()
{
    Application.Run(new FibonacciForm());
}
}
}

```

```

Imports System
Imports System.Collections
Imports System.ComponentModel
Imports System.Drawing
Imports System.Threading
Imports System.Windows.Forms

Public Class FibonacciForm
    Inherits System.Windows.Forms.Form

```

```

Private numberToCompute As Integer = 0
Private highestPercentageReached As Integer = 0

Private numericUpDown1 As System.Windows.Forms.NumericUpDown
Private WithEvents startAsyncButton As System.Windows.Forms.Button
Private WithEvents cancelAsyncButton As System.Windows.Forms.Button
Private progressBar1 As System.Windows.Forms.ProgressBar
Private resultLabel As System.Windows.Forms.Label
Private WithEvents backgroundWorker1 As System.ComponentModel.BackgroundWorker


Public Sub New()
    InitializeComponent()
End Sub 'New

Private Sub startAsyncButton_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) _
Handles startAsyncButton.Click

    ' Reset the text in the result label.
    resultLabel.Text = [String].Empty

    ' Disable the UpDown control until
    ' the asynchronous operation is done.
    Me.numericUpDown1.Enabled = False

    ' Disable the Start button until
    ' the asynchronous operation is done.
    Me.startAsyncButton.Enabled = False

    ' Enable the Cancel button while
    ' the asynchronous operation runs.
    Me.cancelAsyncButton.Enabled = True

    ' Get the value from the UpDown control.
    numberToCompute = CInt(numericUpDown1.Value)

    ' Reset the variable for percentage tracking.
    highestPercentageReached = 0

    ' Start the asynchronous operation.
    backgroundWorker1.RunWorkerAsync(numberToCompute)
End Sub

Private Sub cancelAsyncButton_Click( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) _
 Handles cancelAsyncButton.Click

    ' Cancel the asynchronous operation.
    Me.backgroundWorker1.CancelAsync()

    ' Disable the Cancel button.
    cancelAsyncButton.Enabled = False

End Sub 'cancelAsyncButton_Click

' This event handler is where the actual work is done.
Private Sub backgroundWorker1_DoWork( _
 ByVal sender As Object, _
 ByVal e As DoWorkEventArgs) _
 Handles backgroundWorker1.DoWork

    ' Get the BackgroundWorker object that raised this event.
    Dim worker As BackgroundWorker = _
        CType(sender, BackgroundWorker)

```

```

' Assign the result of the computation
' to the Result property of the DoWorkEventArgs
' object. This is will be available to the
' RunWorkerCompleted eventhandler.
e.Result = ComputeFibonacci(e.Argument, worker, e)
End Sub 'backgroundWorker1_DoWork

' This event handler deals with the results of the
' background operation.
Private Sub backgroundWorker1_RunWorkerCompleted( _
ByVal sender As Object, ByVal e As RunWorkerCompletedEventArgs) _
Handles backgroundWorker1.RunWorkerCompleted

    ' First, handle the case where an exception was thrown.
    If (e.Error IsNot Nothing) Then
        MessageBox.Show(e.Error.Message)
    ElseIf e.Cancelled Then
        ' Next, handle the case where the user canceled the
        ' operation.
        ' Note that due to a race condition in
        ' the DoWork event handler, the Cancelled
        ' flag may not have been set, even though
        ' CancelAsync was called.
        resultLabel.Text = "Canceled"
    Else
        ' Finally, handle the case where the operation succeeded.
        resultLabel.Text = e.Result.ToString()
    End If

    ' Enable the UpDown control.
    Me.numericUpDown1.Enabled = True

    ' Enable the Start button.
    startAsyncButton.Enabled = True

    ' Disable the Cancel button.
    cancelAsyncButton.Enabled = False
End Sub 'backgroundWorker1_RunWorkerCompleted

' This event handler updates the progress bar.
Private Sub backgroundWorker1_ProgressChanged( _
ByVal sender As Object, ByVal e As ProgressChangedEventArgs) _
Handles backgroundWorker1.ProgressChanged

    Me.progressBar1.Value = e.ProgressPercentage

End Sub

' This is the method that does the actual work. For this
' example, it computes a Fibonacci number and
' reports progress as it does its work.
Function ComputeFibonacci( _
    ByVal n As Integer, _
    ByVal worker As BackgroundWorker, _
    ByVal e As DoWorkEventArgs) As Long

    ' The parameter n must be >= 0 and <= 91.
    ' Fib(n), with n > 91, overflows a long.
    If n < 0 OrElse n > 91 Then
        Throw New ArgumentException( _
            "value must be >= 0 and <= 91", "n")
    End If

    Dim result As Long = 0

    ' Abort the operation if the user has canceled.
    ' Note that a call to CancelAsync may have set
    ' CancellationPending to true just after the
    ' last invocation of this method exits, so this

```

```

' code will not have the opportunity to set the
' DoworkEventArgs.Cancel flag to true. This means
' that RunWorkerCompletedEventArgs.Cancelled will
' not be set to true in your RunWorkerCompleted
' event handler. This is a race condition.
If worker.CancellationPending Then
    e.Cancel = True
Else
    If n < 2 Then
        result = 1
    Else
        result = ComputeFibonacci(n - 1, worker, e) + _
                  ComputeFibonacci(n - 2, worker, e)
    End If

    ' Report progress as a percentage of the total task.
    Dim percentComplete As Integer = _
        CSng(n) / CSng(numberToCompute) * 100
    If percentComplete > highestPercentageReached Then
        highestPercentageReached = percentComplete
        worker.ReportProgress(percentComplete)
    End If

End If

Return result

End Function

Private Sub InitializeComponent()
    Me.numericUpDown1 = New System.Windows.Forms.NumericUpDown
    Me.startAsyncButton = New System.Windows.Forms.Button
    Me.cancelAsyncButton = New System.Windows.Forms.Button
    Me.resultLabel = New System.Windows.Forms.Label
    Me.progressBar1 = New System.Windows.Forms.ProgressBar
    Me.backgroundWorker1 = New System.ComponentModel.BackgroundWorker
    CType(Me.numericUpDown1, System.ComponentModel.ISupportInitialize).BeginInit()
    Me.SuspendLayout()

    'numericUpDown1
    '

    Me.numericUpDown1.Location = New System.Drawing.Point(16, 16)
    Me.numericUpDown1.Maximum = New Decimal(New Integer() {91, 0, 0, 0})
    Me.numericUpDown1.Minimum = New Decimal(New Integer() {1, 0, 0, 0})
    Me.numericUpDown1.Name = "numericUpDown1"
    Me.numericUpDown1.Size = New System.Drawing.Size(80, 20)
    Me.numericUpDown1.TabIndex = 0
    Me.numericUpDown1.Value = New Decimal(New Integer() {1, 0, 0, 0})

    'startAsyncButton
    '

    Me.startAsyncButton.Location = New System.Drawing.Point(16, 72)
    Me.startAsyncButton.Name = "startAsyncButton"
    Me.startAsyncButton.Size = New System.Drawing.Size(120, 23)
    Me.startAsyncButton.TabIndex = 1
    Me.startAsyncButton.Text = "Start Async"

    'cancelAsyncButton
    '

    Me.cancelAsyncButton.Enabled = False
    Me.cancelAsyncButton.Location = New System.Drawing.Point(153, 72)
    Me.cancelAsyncButton.Name = "cancelAsyncButton"
    Me.cancelAsyncButton.Size = New System.Drawing.Size(119, 23)
    Me.cancelAsyncButton.TabIndex = 2
    Me.cancelAsyncButton.Text = "Cancel Async"

    'resultLabel
    '

```

```

Me.resultLabel.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle
Me.resultLabel.Location = New System.Drawing.Point(112, 16)
Me.resultLabel.Name = "resultLabel"
Me.resultLabel.Size = New System.Drawing.Size(160, 23)
Me.resultLabel.TabIndex = 3
Me.resultLabel.Text = "(no result)"
Me.resultLabel.TextAlign = System.Drawing.ContentAlignment.MiddleCenter
'

'progressBar1
'

Me.progressBar1.Location = New System.Drawing.Point(18, 48)
Me.progressBar1.Name = "progressBar1"
Me.progressBar1.Size = New System.Drawing.Size(256, 8)
Me.progressBar1.TabIndex = 4
'

'backgroundWorker1
'

Me.backgroundWorker1.WorkerReportsProgress = True
Me.backgroundWorker1.WorkerSupportsCancellation = True
'

'FibonacciForm
'

Me.ClientSize = New System.Drawing.Size(292, 118)
Me.Controls.Add(Me.progressBar1)
Me.Controls.Add(Me.resultLabel)
Me.Controls.Add(Me.cancelAsyncButton)
Me.Controls.Add(Me.startAsyncButton)
Me.Controls.Add(Me.numericUpDown1)
Me.Name = "FibonacciForm"
Me.Text = "Fibonacci Calculator"
 CType(Me.numericUpDown1, System.ComponentModel.ISupportInitialize).EndInit()
Me.ResumeLayout(False)
Me.PerformLayout(False)

End Sub 'InitializeComponent

<STAThread()> _
Shared Sub Main()
    Application.Run(New FibonacciForm)
End Sub 'Main
End Class 'FibonacciForm

```

Compiling the Code

This example requires:

- References to the System, System.Drawing, and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

Robust Programming

Caution

When using multithreading of any sort, you potentially expose yourself to very serious and complex bugs. Consult the [Managed Threading Best Practices](#) before implementing any solution that uses multithreading.

See Also

[BackgroundWorker](#)
[DoWorkEventArgs](#)

Event-based Asynchronous Pattern Overview

Managed Threading Best Practices

Walkthrough: Running an Operation in the Background

5/4/2018 • 5 min to read • [Edit Online](#)

If you have an operation that will take a long time to complete, and you do not want to cause delays in your user interface, you can use the [BackgroundWorker](#) class to run the operation on another thread.

For a complete listing of the code used in this example, see [How to: Run an Operation in the Background](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To run an operation in the background

1. With your form active in the Windows Forms Designer, drag two **Button** controls from the **Toolbox** to the form, and then set the **Name** and **Text** properties of the buttons according to the following table.

BUTTON	NAME	TEXT
button1	startBtn	Start
button2	cancelBtn	Cancel

2. Open the **Toolbox**, click the **Components** tab, and then drag the [BackgroundWorker](#) component onto your form.

The `backgroundWorker1` component appears in the **Component Tray**.

3. In the **Properties** window, set the **WorkerSupportsCancellation** property to `true`.
4. In the **Properties** window, click on the **Events** button, and then double-click the **DoWork** and **RunWorkerCompleted** events to create event handlers.
5. Insert your time-consuming code into the **DoWork** event handler.
6. Extract any parameters required by the operation from the **Argument** property of the **DoWorkEventArgs** parameter.
7. Assign the result of the computation to the **Result** property of the **DoWorkEventArgs**.

This is will be available to the **RunWorkerCompleted** event handler.

```

private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
{
    // Do not access the form's BackgroundWorker reference directly.
    // Instead, use the reference provided by the sender parameter.
    BackgroundWorker bw = sender as BackgroundWorker;

    // Extract the argument.
    int arg = (int)e.Argument;

    // Start the time-consuming operation.
    e.Result = TimeConsumingOperation(bw, arg);

    // If the operation was canceled by the user,
    // set the DoWorkEventArgs.Cancel property to true.
    if (bw.CancellationPending)
    {
        e.Cancel = true;
    }
}

```

```

Private Sub backgroundWorker1_DoWork( _
    sender As Object, e As DoWorkEventArgs) _
Handles backgroundWorker1.DoWork

    ' Do not access the form's BackgroundWorker reference directly.
    ' Instead, use the reference provided by the sender parameter.
    Dim bw As BackgroundWorker = CType( sender, BackgroundWorker )

    ' Extract the argument.
    Dim arg As Integer = Fix(e.Argument)

    ' Start the time-consuming operation.
    e.Result = TimeConsumingOperation(bw, arg)

    ' If the operation was canceled by the user,
    ' set the DoWorkEventArgs.Cancel property to true.
    If bw.CancellationPending Then
        e.Cancel = True
    End If

End Sub

```

8. Insert code for retrieving the result of your operation in the [RunWorkerCompleted](#) event handler.

```

// This event handler demonstrates how to interpret
// the outcome of the asynchronous operation implemented
// in the DoWork event handler.
private void backgroundWorker1_RunWorkerCompleted(
    object sender,
    RunWorkerCompletedEventArgs e)
{
    if (e.Cancelled)
    {
        // The user canceled the operation.
        MessageBox.Show("Operation was canceled");
    }
    else if (e.Error != null)
    {
        // There was an error during the operation.
        string msg = String.Format("An error occurred: {0}", e.Error.Message);
        MessageBox.Show(msg);
    }
    else
    {
        // The operation completed normally.
        string msg = String.Format("Result = {0}", e.Result);
        MessageBox.Show(msg);
    }
}

```

```

' This event handler demonstrates how to interpret
' the outcome of the asynchronous operation implemented
' in the DoWork event handler.
Private Sub backgroundWorker1_RunWorkerCompleted( _
    sender As Object, e As RunWorkerCompletedEventArgs) _
    Handles backgroundWorker1.RunWorkerCompleted

    If e.Cancelled Then
        ' The user canceled the operation.
        MessageBox.Show("Operation was canceled")
    ElseIf (e.Error IsNot Nothing) Then
        ' There was an error during the operation.
        Dim msg As String = String.Format("An error occurred: {0}", e.Error.Message)
        MessageBox.Show(msg)
    Else
        ' The operation completed normally.
        Dim msg As String = String.Format("Result = {0}", e.Result)
        MessageBox.Show(msg)
    End If
End Sub

```

9. Implement the `TimeConsumingOperation` method.

```
// This method models an operation that may take a long time
// to run. It can be cancelled, it can raise an exception,
// or it can exit normally and return a result. These outcomes
// are chosen randomly.
private int TimeConsumingOperation(
    BackgroundWorker bw,
    int sleepPeriod )
{
    int result = 0;

    Random rand = new Random();

    while (!bw.CancellationPending)
    {
        bool exit = false;

        switch (rand.Next(3))
        {
            // Raise an exception.
            case 0:
            {
                throw new Exception("An error condition occurred.");
                break;
            }

            // Sleep for the number of milliseconds
            // specified by the sleepPeriod parameter.
            case 1:
            {
                Thread.Sleep(sleepPeriod);
                break;
            }

            // Exit and return normally.
            case 2:
            {
                result = 23;
                exit = true;
                break;
            }

            default:
            {
                break;
            }
        }

        if( exit )
        {
            break;
        }
    }

    return result;
}
```

```

' This method models an operation that may take a long time
' to run. It can be cancelled, it can raise an exception,
' or it can exit normally and return a result. These outcomes
' are chosen randomly.
Private Function TimeConsumingOperation( _
bw As BackgroundWorker, _
sleepPeriod As Integer) As Integer

    Dim result As Integer = 0

    Dim rand As New Random()

    While Not bw.CancellationPending
        Dim [exit] As Boolean = False

        Select Case rand.Next(3)
            ' Raise an exception.
            Case 0
                Throw New Exception("An error condition occurred.")
                Exit While

            ' Sleep for the number of milliseconds
            ' specified by the sleepPeriod parameter.
            Case 1
                Thread.Sleep(sleepPeriod)
                Exit While

            ' Exit and return normally.
            Case 2
                result = 23
                [exit] = True
                Exit While

            Case Else
                Exit While
        End Select

        If [exit] Then
            Exit While
        End If
    End While

    Return result
End Function

```

10. In the Windows Forms Designer, double-click `startButton` to create the `Click` event handler.

11. Call the `RunWorkerAsync` method in the `Click` event handler for `startButton`.

```

private void startBtn_Click(object sender, EventArgs e)
{
    this.backgroundWorker1.RunWorkerAsync(2000);
}

```

```

Private Sub startButton_Click(ByVal sender As Object, ByVal e As EventArgs) Handles startBtn.Click
    Me.backgroundWorker1.RunWorkerAsync(2000)
End Sub

```

12. In the Windows Forms Designer, double-click `cancelButton` to create the `Click` event handler.

13. Call the `CancelAsync` method in the `Click` event handler for `cancelButton`.

```
private void cancelBtn_Click(object sender, EventArgs e)
{
    this.backgroundWorker1.CancelAsync();
}
```

```
Private Sub cancelButton_Click(ByVal sender As Object, ByVal e As EventArgs) Handles cancelButton.Click
    Me.backgroundWorker1.CancelAsync()
End Sub
```

14. At the top of the file, import the System.ComponentModel and System.Threading namespaces.

```
using System;
using System.ComponentModel;
using System.Drawing;
using System.Threading;
using System.Windows.Forms;
```

```
Imports System
Imports System.ComponentModel
Imports System.Drawing
Imports System.Threading
Imports System.Windows.Forms
```

15. Press F6 to build the solution, and then press CTRL-F5 to run the application outside the debugger.

NOTE

If you press F5 to run the application under the debugger, the exception raised in the `TimeConsumingOperation` method is caught and displayed by the debugger. When you run the application outside the debugger, the `BackgroundWorker` handles the exception and caches it in the `Error` property of the `RunWorkerCompletedEventArgs`.

1. Click the **Start** button to run an asynchronous operation, and then click the **Cancel** button to stop a running asynchronous operation.

The outcome of each operation is displayed in a `MessageBox`.

Next Steps

- Implement a form that reports progress as an asynchronous operation proceeds. For more information, see [How to: Implement a Form That Uses a Background Operation](#).
- Implement a class that supports the Asynchronous Pattern for Components. For more information, see [Implementing the Event-based Asynchronous Pattern](#).

See Also

[BackgroundWorker](#)

[DoWorkEventArgs](#)

[How to: Implement a Form That Uses a Background Operation](#)

[How to: Run an Operation in the Background](#)

[BackgroundWorker Component](#)

Walkthrough: Implementing a Form That Uses a Background Operation

5/4/2018 • 14 min to read • [Edit Online](#)

If you have an operation that will take a long time to complete, and you do not want your user interface (UI) to stop responding or "hang," you can use the [BackgroundWorker](#) class to execute the operation on another thread.

This walkthrough illustrates how to use the [BackgroundWorker](#) class to perform time-consuming computations "in the background," while the user interface remains responsive. When you are through, you will have an application that computes Fibonacci numbers asynchronously. Even though computing a large Fibonacci number can take a noticeable amount of time, the main UI thread will not be interrupted by this delay, and the form will be responsive during the calculation.

Tasks illustrated in this walkthrough include:

- Creating a Windows-based Application
- Creating a [BackgroundWorker](#) in Your Form
- Adding Asynchronous Event Handlers
- Adding Progress Reporting and Support for Cancellation

For a complete listing of the code used in this example, see [How to: Implement a Form That Uses a Background Operation](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

Creating the Project

The first step is to create the project and to set up the form.

To create a form that uses a background operation

1. Create a Windows-based application project called `BackgroundWorkerExample`. For details, see [How to: Create a Windows Application Project](#).
2. In **Solution Explorer**, right-click **Form1** and select **Rename** from the shortcut menu. Change the file name to `FibonacciCalculator`. Click the **Yes** button when you are asked if you want to rename all references to the code element '`Form1`'.
3. Drag a [NumericUpDown](#) control from the **Toolbox** onto the form. Set the **Minimum** property to `1` and the **Maximum** property to `91`.
4. Add two [Button](#) controls to the form.
5. Rename the first [Button](#) control `startAsyncButton` and set the **Text** property to `Start Async`. Rename the second [Button](#) control `cancelAsyncButton`, and set the **Text** property to `Cancel Async`. Set its **Enabled** property to `false`.

6. Create an event handler for both of the [Button](#) controls' [Click](#) events. For details, see [How to: Create Event Handlers Using the Designer](#).
7. Drag a [Label](#) control from the **Toolbox** onto the form and rename it `resultLabel`.
8. Drag a [ProgressBar](#) control from the **Toolbox** onto the form.

Creating a BackgroundWorker in Your Form

You can create the [BackgroundWorker](#) for your asynchronous operation using the **Windows Forms Designer**.

To create a BackgroundWorker with the Designer

- From the **Components** tab of the **Toolbox**, drag a [BackgroundWorker](#) onto the form.

Adding Asynchronous Event Handlers

You are now ready to add event handlers for the [BackgroundWorker](#) component's asynchronous events. The time-consuming operation that will run in the background, which computes Fibonacci numbers, is called by one of these event handlers.

To implement asynchronous event handlers

1. In the **Properties** window, with the [BackgroundWorker](#) component still selected, click the **Events** button. Double-click the [DoWork](#) and [RunWorkerCompleted](#) events to create event handlers. For more information about how to use event handlers, see [How to: Create Event Handlers Using the Designer](#).
2. Create a new method, called `ComputeFibonacci`, in your form. This method does the actual work, and it will run in the background. This code demonstrates the recursive implementation of the Fibonacci algorithm, which is notably inefficient, taking exponentially longer time to complete for larger numbers. It is used here for illustrative purposes, to show an operation that can introduce long delays in your application.

```

// This is the method that does the actual work. For this
// example, it computes a Fibonacci number and
// reports progress as it does its work.
long ComputeFibonacci( int n, BackgroundWorker^ worker, DoWorkEventArgs ^ e )
{
    // The parameter n must be >= 0 and <= 91.
    // Fib(n), with n > 91, overflows a long.
    if ( (n < 0) || (n > 91) )
    {
        throw gcnew ArgumentException( "value must be >= 0 and <= 91","n" );
    }

    long result = 0;

    // Abort the operation if the user has cancelled.
    // Note that a call to CancelAsync may have set
    // CancellationPending to true just after the
    // last invocation of this method exits, so this
    // code will not have the opportunity to set the
    // DoWorkEventArgs.Cancel flag to true. This means
    // that RunWorkerCompletedEventArgs.Cancelled will
    // not be set to true in your RunWorkerCompleted
    // event handler. This is a race condition.
    if ( worker->CancellationPending )
    {
        e->Cancel = true;
    }
    else
    {
        if ( n < 2 )
        {
            result = 1;
        }
        else
        {
            result = ComputeFibonacci( n - 1, worker, e ) + ComputeFibonacci( n - 2, worker, e );
        }
    }

    // Report progress as a percentage of the total task.
    int percentComplete = (int)((float)n / (float)numberToCompute * 100);
    if ( percentComplete > highestPercentageReached )
    {
        highestPercentageReached = percentComplete;
        worker->ReportProgress( percentComplete );
    }
}

return result;
}

```

```

// This is the method that does the actual work. For this
// example, it computes a Fibonacci number and
// reports progress as it does its work.
long ComputeFibonacci(int n, BackgroundWorker worker, DoWorkEventArgs e)
{
    // The parameter n must be >= 0 and <= 91.
    // Fib(n), with n > 91, overflows a long.
    if ((n < 0) || (n > 91))
    {
        throw new ArgumentException(
            "value must be >= 0 and <= 91", "n");
    }

    long result = 0;

    // Abort the operation if the user has canceled.
    // Note that a call to CancelAsync may have set
    // CancellationPending to true just after the
    // last invocation of this method exits, so this
    // code will not have the opportunity to set the
    // DoWorkEventArgs.Cancel flag to true. This means
    // that RunWorkerCompletedEventArgs.Cancelled will
    // not be set to true in your RunWorkerCompleted
    // event handler. This is a race condition.

    if (worker.CancellationPending)
    {
        e.Cancel = true;
    }
    else
    {
        if (n < 2)
        {
            result = 1;
        }
        else
        {
            result = ComputeFibonacci(n - 1, worker, e) +
                ComputeFibonacci(n - 2, worker, e);
        }

        // Report progress as a percentage of the total task.
        int percentComplete =
            (int)((float)n / (float)numberToCompute * 100);
        if (percentComplete > highestPercentageReached)
        {
            highestPercentageReached = percentComplete;
            worker.ReportProgress(percentComplete);
        }
    }
}

return result;
}

```

```

' This is the method that does the actual work. For this
' example, it computes a Fibonacci number and
' reports progress as it does its work.
Function ComputeFibonacci( _
    ByVal n As Integer, _
    ByVal worker As BackgroundWorker, _
    ByVal e As DoWorkEventArgs) As Long

    ' The parameter n must be >= 0 and <= 91.
    ' Fib(n), with n > 91, overflows a long.
    If n < 0 OrElse n > 91 Then
        Throw New ArgumentException( _
            "value must be >= 0 and <= 91", "n")
    End If

    Dim result As Long = 0

    ' Abort the operation if the user has canceled.
    ' Note that a call to CancelAsync may have set
    ' CancellationPending to true just after the
    ' last invocation of this method exits, so this
    ' code will not have the opportunity to set the
    ' DoWorkEventArgs.Cancel flag to true. This means
    ' that RunWorkerCompletedEventArgs.Cancelled will
    ' not be set to true in your RunWorkerCompleted
    ' event handler. This is a race condition.
    If worker.CancellationPending Then
        e.Cancel = True
    Else
        If n < 2 Then
            result = 1
        Else
            result = ComputeFibonacci(n - 1, worker, e) + _
                ComputeFibonacci(n - 2, worker, e)
        End If

        ' Report progress as a percentage of the total task.
        Dim percentComplete As Integer = _
            CSng(n) / CSng(numberToCompute) * 100
        If percentComplete > highestPercentageReached Then
            highestPercentageReached = percentComplete
            worker.ReportProgress(percentComplete)
        End If

    End If

    Return result
End Function

```

3. In the [DoWork](#) event handler, add a call to the `ComputeFibonacci` method. Take the first parameter for `ComputeFibonacci` from the [Argument](#) property of the [DoWorkEventArgs](#). The [BackgroundWorker](#) and [DoWorkEventArgs](#) parameters will be used later for progress reporting and cancellation support. Assign the return value from `ComputeFibonacci` to the [Result](#) property of the [DoWorkEventArgs](#). This result will be available to the [RunWorkerCompleted](#) event handler.

NOTE

The `DoWork` event handler does not reference the `backgroundWorker1` instance variable directly, as this would couple this event handler to a specific instance of `BackgroundWorker`. Instead, a reference to the `BackgroundWorker` that raised this event is recovered from the `sender` parameter. This is important when the form hosts more than one `BackgroundWorker`. It is also important not to manipulate any user-interface objects in your `DoWork` event handler. Instead, communicate to the user interface through the `BackgroundWorker` events.

```
// This event handler is where the actual,
// potentially time-consuming work is done.
void backgroundWorker1_DoWork( Object^ sender, DoWorkEventArgs^ e )
{
    // Get the BackgroundWorker that raised this event.
    BackgroundWorker^ worker = dynamic_cast<BackgroundWorker^>(sender);

    // Assign the result of the computation
    // to the Result property of the DoWorkEventArgs
    // object. This is will be available to the
    // RunWorkerCompleted eventhandler.
    e->Result = ComputeFibonacci( safe_cast<Int32>(e->Argument), worker, e );
}
```

```
// This event handler is where the actual,
// potentially time-consuming work is done.
private void backgroundWorker1_DoWork(object sender,
    DoWorkEventArgs e)
{
    // Get the BackgroundWorker that raised this event.
    BackgroundWorker worker = sender as BackgroundWorker;

    // Assign the result of the computation
    // to the Result property of the DoWorkEventArgs
    // object. This is will be available to the
    // RunWorkerCompleted eventhandler.
    e.Result = ComputeFibonacci((int)e.Argument, worker, e);
}
```

```
' This event handler is where the actual work is done.
Private Sub backgroundWorker1_DoWork( _
    ByVal sender As Object, _
    ByVal e As DoWorkEventArgs) _
Handles backgroundWorker1.DoWork

    ' Get the BackgroundWorker object that raised this event.
    Dim worker As BackgroundWorker = _
        CType(sender, BackgroundWorker)

    ' Assign the result of the computation
    ' to the Result property of the DoWorkEventArgs
    ' object. This is will be available to the
    ' RunWorkerCompleted eventhandler.
    e.Result = ComputeFibonacci(e.Argument, worker, e)
End Sub 'backgroundWorker1_DoWork
```

4. In the `startAsyncButton` control's `Click` event handler, add the code that starts the asynchronous operation.

```

void startAsyncButton_Click( System::Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    // Reset the text in the result label.
    resultLabel->Text = String::Empty;

    // Disable the UpDown control until
    // the asynchronous operation is done.
    this->numericUpDown1->Enabled = false;

    // Disable the Start button until
    // the asynchronous operation is done.
    this->startAsyncButton->Enabled = false;

    // Enable the Cancel button while
    // the asynchronous operation runs.
    this->cancelAsyncButton->Enabled = true;

    // Get the value from the UpDown control.
    numberToCompute = (int)numericUpDown1->Value;

    // Reset the variable for percentage tracking.
    highestPercentageReached = 0;

    // Start the asynchronous operation.
    backgroundWorker1->RunWorkerAsync( numberToCompute );
}

```

```

private void startAsyncButton_Click(System.Object sender,
    System.EventArgs e)
{
    // Reset the text in the result label.
    resultLabel.Text = String.Empty;

    // Disable the UpDown control until
    // the asynchronous operation is done.
    this.numericUpDown1.Enabled = false;

    // Disable the Start button until
    // the asynchronous operation is done.
    this.startAsyncButton.Enabled = false;

    // Enable the Cancel button while
    // the asynchronous operation runs.
    this.cancelAsyncButton.Enabled = true;

    // Get the value from the UpDown control.
    numberToCompute = (int)numericUpDown1.Value;

    // Reset the variable for percentage tracking.
    highestPercentageReached = 0;

    // Start the asynchronous operation.
    backgroundWorker1.RunWorkerAsync(numberToCompute);
}

```

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

    ' Reset the text in the result label.
    resultLabel.Text = [String].Empty

    ' Disable the UpDown control until
    ' the asynchronous operation is done.
    Me.numericUpDown1.Enabled = False

    ' Disable the Start button until
    ' the asynchronous operation is done.
    Me.startAsyncButton.Enabled = False

    ' Enable the Cancel button while
    ' the asynchronous operation runs.
    Me.cancelAsyncButton.Enabled = True

    ' Get the value from the UpDown control.
    numberToCompute = CInt(numericUpDown1.Value)

    ' Reset the variable for percentage tracking.
    highestPercentageReached = 0

    ' Start the asynchronous operation.
    backgroundWorker1.RunWorkerAsync(numberToCompute)
End Sub
```

5. In the [RunWorkerCompleted](#) event handler, assign the result of the calculation to the `resultLabel` control.

```
// This event handler deals with the results of the
// background operation.
void backgroundWorker1_RunWorkerCompleted( Object^ /*sender*/, RunWorkerCompletedEventArgs^ e )
{
    // First, handle the case where an exception was thrown.
    if ( e->Error != nullptr )
    {
        MessageBox::Show( e->Error->Message );
    }
    else
    if ( e->Cancelled )
    {
        // Next, handle the case where the user cancelled
        // the operation.
        // Note that due to a race condition in
        // the DoWork event handler, the Cancelled
        // flag may not have been set, even though
        // CancelAsync was called.
        resultLabel->Text = "Cancelled";
    }
    else
    {
        // Finally, handle the case where the operation
        // succeeded.
        resultLabel->Text = e->Result->ToString();
    }

    // Enable the UpDown control.
    this->numericUpDown1->Enabled = true;

    // Enable the Start button.
    startAsyncButton->Enabled = true;

    // Disable the Cancel button.
    cancelAsyncButton->Enabled = false;
}
```

```
// This event handler deals with the results of the
// background operation.
private void backgroundWorker1_RunWorkerCompleted(
    object sender, RunWorkerCompletedEventArgs e)
{
    // First, handle the case where an exception was thrown.
    if (e.Error != null)
    {
        MessageBox.Show(e.Error.Message);
    }
    else if (e.Cancelled)
    {
        // Next, handle the case where the user canceled
        // the operation.
        // Note that due to a race condition in
        // the DoWork event handler, the Cancelled
        // flag may not have been set, even though
        // CancelAsync was called.
        resultLabel.Text = "Canceled";
    }
    else
    {
        // Finally, handle the case where the operation
        // succeeded.
        resultLabel.Text = e.Result.ToString();
    }

    // Enable the UpDown control.
    this.numericUpDown1.Enabled = true;

    // Enable the Start button.
    startAsyncButton.Enabled = true;

    // Disable the Cancel button.
    cancelAsyncButton.Enabled = false;
}
```

```

' This event handler deals with the results of the
' background operation.
Private Sub backgroundWorker1_RunWorkerCompleted( _
    ByVal sender As Object, ByVal e As RunWorkerCompletedEventArgs) _
Handles backgroundWorker1.RunWorkerCompleted

    ' First, handle the case where an exception was thrown.
    If (e.Error IsNot Nothing) Then
        MessageBox.Show(e.Error.Message)
    ElseIf e.Cancelled Then
        ' Next, handle the case where the user canceled the
        ' operation.
        ' Note that due to a race condition in
        ' the DoWork event handler, the Cancelled
        ' flag may not have been set, even though
        ' CancelAsync was called.
        resultLabel.Text = "Canceled"
    Else
        ' Finally, handle the case where the operation succeeded.
        resultLabel.Text = e.Result.ToString()
    End If

    ' Enable the UpDown control.
    Me.numericUpDown1.Enabled = True

    ' Enable the Start button.
    startAsyncButton.Enabled = True

    ' Disable the Cancel button.
    cancelAsyncButton.Enabled = False
End Sub 'backgroundWorker1_RunWorkerCompleted

```

Adding Progress Reporting and Support for Cancellation

For asynchronous operations that will take a long time, it is often desirable to report progress to the user and to allow the user to cancel the operation. The [BackgroundWorker](#) class provides an event that allows you to post progress as your background operation proceeds. It also provides a flag that allows your worker code to detect a call to [CancelAsync](#) and interrupt itself.

To implement progress reporting

1. In the **Properties** window, select `backgroundWorker1`. Set the [WorkerReportsProgress](#) and [WorkerSupportsCancellation](#) properties to `true`.
2. Declare two variables in the `FibonacciCalculator` form. These will be used to track progress.

```

int numberToCompute;
int highestPercentageReached;

```

```

private int numberToCompute = 0;
private int highestPercentageReached = 0;

```

```

Private numberToCompute As Integer = 0
Private highestPercentageReached As Integer = 0

```

3. Add an event handler for the [ProgressChanged](#) event. In the [ProgressChanged](#) event handler, update the [ProgressBar](#) with the [ProgressPercentage](#) property of the [ProgressChangedEventArgs](#) parameter.

```
// This event handler updates the progress bar.
void backgroundWorker1_ProgressChanged( Object^ /*sender*/, ProgressChangedEventArgs^ e )
{
    this->progressBar1->Value = e->ProgressPercentage;
}
```

```
// This event handler updates the progress bar.
private void backgroundWorker1_ProgressChanged(object sender,
    ProgressChangedEventArgs e)
{
    this.progressBar1.Value = e.ProgressPercentage;
}
```

```
' This event handler updates the progress bar.
Private Sub backgroundWorker1_ProgressChanged( _
    ByVal sender As Object, ByVal e As ProgressChangedEventArgs ) _
Handles backgroundWorker1.ProgressChanged

    Me.progressBar1.Value = e.ProgressPercentage

End Sub
```

To implement support for cancellation

1. In the `cancelAsyncButton` control's [Click](#) event handler, add the code that cancels the asynchronous operation.

```
void cancelAsyncButton_Click( System::Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    // Cancel the asynchronous operation.
    this->backgroundWorker1->CancelAsync();

    // Disable the Cancel button.
    cancelAsyncButton->Enabled = false;
}
```

```
private void cancelAsyncButton_Click(System.Object sender,
    System.EventArgs e)
{
    // Cancel the asynchronous operation.
    this.backgroundWorker1.CancelAsync();

    // Disable the Cancel button.
    cancelAsyncButton.Enabled = false;
}
```

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

    ' Cancel the asynchronous operation.
    Me.backgroundWorker1.CancelAsync()

    ' Disable the Cancel button.
    cancelAsyncButton.Enabled = False

End Sub 'cancelAsyncButton_Click
```

2. The following code fragments in the `ComputeFibonacci` method report progress and support cancellation.

```
if ( worker->CancellationPending )
{
    e->Cancel = true;
}
```

```
if (worker.CancellationPending)
{
    e.Cancel = true;
}
```

```
If worker.CancellationPending Then
    e.Cancel = True
```

```
// Report progress as a percentage of the total task.
int percentComplete = (int)((float)n / (float)numberToCompute * 100);
if ( percentComplete > highestPercentageReached )
{
    highestPercentageReached = percentComplete;
    worker->ReportProgress( percentComplete );
}
```

```
// Report progress as a percentage of the total task.
int percentComplete =
    (int)((float)n / (float)numberToCompute * 100);
if (percentComplete > highestPercentageReached)
{
    highestPercentageReached = percentComplete;
    worker.ReportProgress(percentComplete);
}
```

```
' Report progress as a percentage of the total task.
Dim percentComplete As Integer = _
    CSng(n) / CSng(numberToCompute) * 100
If percentComplete > highestPercentageReached Then
    highestPercentageReached = percentComplete
    worker.ReportProgress(percentComplete)
End If
```

Checkpoint

At this point, you can compile and run the Fibonacci Calculator application.

To test your project

- Press F5 to compile and run the application.

While the calculation is running in the background, you will see the [ProgressBar](#) displaying the progress of the calculation toward completion. You can also cancel the pending operation.

For small numbers, the calculation should be very fast, but for larger numbers, you should see a noticeable delay. If you enter a value of 30 or greater, you should see a delay of several seconds, depending on the speed of your computer. For values greater than 40, it may take minutes or hours to finish the calculation. While the calculator is busy computing a large Fibonacci number, notice that you can freely move the form around, minimize, maximize, and even dismiss it. This is because the main UI thread is not waiting for the

calculation to finish.

Next Steps

Now that you have implemented a form that uses a [BackgroundWorker](#) component to execute a computation in the background, you can explore other possibilities for asynchronous operations:

- Use multiple [BackgroundWorker](#) objects for several simultaneous operations.
- To debug your multithreaded application, see [How to: Use the Threads Window](#).
- Implement your own component that supports the asynchronous programming model. For more information, see [Event-based Asynchronous Pattern Overview](#).

Caution

When using multithreading of any sort, you potentially expose yourself to very serious and complex bugs. Consult the [Managed Threading Best Practices](#) before implementing any solution that uses multithreading.

See Also

[BackgroundWorker](#)

[Managed Threading Best Practices](#)

[Multithreading in Components](#)

[NOT IN BUILD: Multithreading in Visual Basic](#)

[How to: Implement a Form That Uses a Background Operation](#)

[Walkthrough: Running an Operation in the Background](#)

[BackgroundWorker Component](#)

BindingNavigator Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The `BindingNavigator` control is the navigation and manipulation user interface (UI) for controls that are bound to data. The `BindingNavigator` control enables users to navigate through and manipulate data on a Windows Form.

The topics in this section provide an overview of the `BindingNavigator` control and offer step-by-step instructions how to use the control navigate data and move through a `DataSet`.

In This Section

[BindingNavigator Control Overview](#)

Introduces the general concepts of the `BindingNavigator` control, which enables users to move through the items of a data source.

[How to: Navigate Data with the Windows Forms BindingNavigator Control](#)

Provides steps to bind a `BindingNavigator` control to a data source.

[How to: Move Through a DataSet with the Windows Forms BindingNavigator Control](#)

Demonstrates using a `BindingNavigator` control to move through records in a `DataSet`.

Also see [How to: Add Load, Save, and Cancel Buttons to the Windows Forms BindingNavigator Control](#).

Reference

[BindingNavigator](#)

Provides reference documentation for the `BindingNavigator` control.

[BindingSource](#)

Provides reference documentation for the `BindingSource` control.

Related Sections

[Bind controls to data in Visual Studio](#)

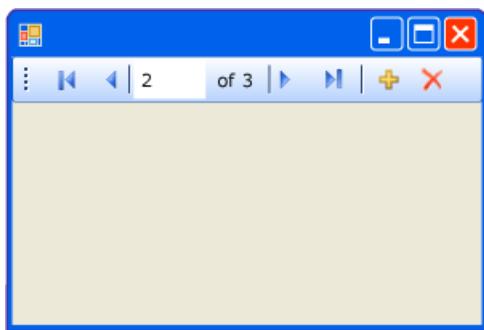
BindingNavigator Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

You can use the [BindingNavigator](#) control to create a standardized means for users to search and change data on a Windows Form. You frequently use [BindingNavigator](#) with the [BindingSource](#) component to enable users to move through data records on a form and interact with the records.

How the BindingNavigator Works

The [BindingNavigator](#) control is composed of a [ToolStrip](#) with a series of [ToolStripItem](#) objects for most of the common data-related actions: adding data, deleting data, and navigating through data. By default, the [BindingNavigator](#) control contains these standard buttons. The following screen shot shows the [BindingNavigator](#) control on a form.



The following table lists the controls and describes their functions.

CONTROL	FUNCTION
AddNewItem button	Inserts a new row into the underlying data source.
DeleteItem button	Deletes the current row from the underlying data source.
MoveFirstItem button	Moves to the first item in the underlying data source.
MoveLastItem button	Moves to the last item in the underlying data source.
MoveNextItem button	Moves to the next item in the underlying data source.
MovePreviousItem button	Moves to the previous item in the underlying data source.
PositionItem text box	Returns the current position within the underlying data source.
CountItem text box	Returns the total number of items in the underlying data source.

For each control in this collection, there is a corresponding member of the [BindingSource](#) component that programmatically provides the same functionality. For example, the [MoveFirstItem](#) button corresponds to the [MoveFirst](#) method of the [BindingSource](#) component, the [DeleteItem](#) button corresponds to the [RemoveCurrent](#) method, and so on.

If the default buttons are not suited to your application, or if you require additional buttons to support other types of functionality, you can supply your own [ToolStrip](#) buttons. Also see [How to: Add Load, Save, and Cancel Buttons to the Windows Forms BindingNavigator Control](#).

See Also

[BindingNavigator](#)

[BindingSource](#)

[BindingNavigator Control](#)

How to: Navigate Data with the Windows Forms BindingNavigator Control

5/4/2018 • 7 min to read • [Edit Online](#)

The advent of the [BindingNavigator](#) control in Windows Forms enables developers to provide end users with a simple data navigation and manipulation user interface on the forms they create.

The [BindingNavigator](#) control is a [ToolStrip](#) control with buttons preconfigured for navigation to the first, last, next, and previous record in a data set, as well as buttons to add and delete records. Adding buttons to the [BindingNavigator](#) control is easy, because it is a [ToolStrip](#) control. Also see [How to: Add Load, Save, and Cancel Buttons to the Windows Forms BindingNavigator Control](#).

For each button on the [BindingNavigator](#) control, there is a corresponding member of the [BindingSource](#) component that programmatically allows the same functionality. For example, the [MoveFirstItem](#) button corresponds to the [MoveFirst](#) method of the [BindingSource](#) component, the [DeleteItem](#) button corresponds to the [RemoveCurrent](#) method, and so on. As a result, enabling the [BindingNavigator](#) control to navigate data records is as simple as setting its [BindingSource](#) property to the appropriate [BindingSource](#) component on the form.

To set up the BindingNavigator control

1. Add a [BindingSource](#) component named `bindingSource1` and two [TextBox](#) controls named `textBox1` and `textBox2`.
2. Bind `bindingSource1` to data, and the textbox controls to `bindingSource1`. To do this, paste the following code into your form and call `LoadData` from the form's constructor or [Load](#) event-handling method.

```
private void LoadData()
{
    // The xml to bind to.
    string xml = @"<US><states>
        + "<state><name>Washington</name><capital>Olympia</capital></state>"
        + "<state><name>Oregon</name><capital>Salem</capital></state>"
        + "<state><name>California</name><capital>Sacramento</capital></state>"
        + "<state><name>Nevada</name><capital>Carson City</capital></state>"
        + "</states></US>";

    // Convert the xml string to bytes and load into a memory stream.
    byte[] xmlBytes = Encoding.UTF8.GetBytes(xml);
    MemoryStream stream = new MemoryStream(xmlBytes, false);

    // Create a DataSet and load the xml into it.
    DataSet set = new DataSet();
    set.ReadXml(stream);

    // Set the DataSource to the DataSet, and the DataMember
    // to state.
    bindingSource1.DataSource = set;
    bindingSource1.DataMember = "state";

    textBox1.DataBindings.Add("Text", bindingSource1, "name");
    textBox2.DataBindings.Add("Text", bindingSource1, "capital");
}
```

```

Private Sub LoadData()
    ' The xml to bind to.
    Dim xml As String = "<US><states>" + "<state><name>Washington</name><capital>Olympia</capital>
    </state>" + "<state><name>Oregon</name><capital>Salem</capital></state>" + "<state>
    <name>California</name><capital>Sacramento</capital></state>" + "<state><name>Nevada</name>
    <capital>Carson City</capital></state>" + "</states></US>

    ' Convert the xml string to bytes and load into a memory stream.
    Dim xmlBytes As Byte() = Encoding.UTF8.GetBytes(xml)
    Dim stream As New MemoryStream(xmlBytes, False)

    ' Create a DataSet and load the xml into it.
    Dim [set] As New DataSet()
    [set].ReadXml(stream)

    ' Set the DataSource to the DataSet, and the DataMember
    ' to state.
    bindingSource1.DataSource = [set]
    bindingSource1.DataMember = "state"

    textBox1.DataBindings.Add("Text", bindingSource1, "name")
    textBox2.DataBindings.Add("Text", bindingSource1, "capital")

End Sub 'LoadData

```

3. Add a [BindingNavigator](#) control named `bindingNavigator1` to your form.

4. Set the [BindingSource](#) property for `bindingNavigator1` to `bindingSource1`. You can do this with the designer or in code.

```
bindingNavigator1.BindingSource = bindingSource1;
```

```
bindingNavigator1.BindingSource = bindingSource1
```

Example

The following code example is the complete example for the steps listed previously.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Windows.Forms;
using System.Xml;
using System.IO;
using System.Text;

namespace System.Windows.Forms.BindingSourceCurrent
{
    class Form1 : Form
    {
        private.IContainer components;
        private BindingNavigator bindingNavigator1;
        private ToolStripButton bindingNavigatorAddNewItem;
        private ToolStripLabel bindingNavigatorCountItem;
        private ToolStripButton bindingNavigatorDeleteItem;
        private ToolStripButton bindingNavigatorMoveFirstItem;
        private ToolStripButton bindingNavigatorMovePreviousItem;
        private ToolStripSeparator bindingNavigatorSeparator;
    }
}

```

```

private ToolStripTextBox bindingNavigatorPositionItem;
private ToolStripSeparator bindingNavigatorSeparator1;
private ToolStripButton bindingNavigatorMoveNextItem;
private ToolStripButton bindingNavigatorMoveLastItem;
private TextBox textBox1;
private TextBox textBox2;
private BindingSource bindingSource1;
private ToolStripSeparator bindingNavigatorSeparator2;

public Form1()
{
    InitializeComponent();
    LoadData();

    bindingNavigator1.BindingSource = bindingSource1;
}

private void LoadData()
{
    // The xml to bind to.
    string xml = @"<US><states>
        + @"<state><name>Washington</name><capital>Olympia</capital></state>" +
        + @"<state><name>Oregon</name><capital>Salem</capital></state>" +
        + @"<state><name>California</name><capital>Sacramento</capital></state>" +
        + @"<state><name>Nevada</name><capital>Carson City</capital></state>" +
        + @"</states></US>";

    // Convert the xml string to bytes and load into a memory stream.
    byte[] xmlBytes = Encoding.UTF8.GetBytes(xml);
    MemoryStream stream = new MemoryStream(xmlBytes, false);

    // Create a DataSet and load the xml into it.
    DataSet set = new DataSet();
    set.ReadXml(stream);

    // Set the DataSource to the DataSet, and the DataMember
    // to state.
    bindingSource1.DataSource = set;
    bindingSource1.DataMember = "state";

    textBox1.DataBindings.Add("Text", bindingSource1, "name");
    textBox2.DataBindings.Add("Text", bindingSource1, "capital");
}

[STAThread]
public static void Main()
{
    Application.EnableVisualStyles();
    Application.Run(new Form1());
}

private void InitializeComponent()
{
    this.components = new System.ComponentModel.Container();
    System.ComponentModel.ComponentResourceManager resources =
        new System.ComponentModel.ComponentResourceManager(typeof(Form1));
    this.bindingNavigator1 = new System.Windows.Forms.BindingNavigator(this.components);
    this.bindingNavigatorAddNewItem = new System.Windows.Forms.ToolStripButton();
    this.bindingNavigatorCountItem = new System.Windows.Forms.ToolStripLabel();
    this.bindingNavigatorDeleteItem = new System.Windows.Forms.ToolStripButton();
    this.bindingNavigatorMoveFirstItem = new System.Windows.Forms.ToolStripButton();
    this.bindingNavigatorMovePreviousItem = new System.Windows.Forms.ToolStripButton();
    this.bindingNavigatorSeparator = new System.Windows.Forms.ToolStripSeparator();
    this.bindingNavigatorPositionItem = new System.Windows.Forms.ToolStripTextBox();
    this.bindingNavigatorSeparator1 = new System.Windows.Forms.ToolStripSeparator();
    this.bindingNavigatorMoveNextItem = new System.Windows.Forms.ToolStripButton();
    this.bindingNavigatorMoveLastItem = new System.Windows.Forms.ToolStripButton();
    this.bindingNavigatorSeparator2 = new System.Windows.Forms.ToolStripSeparator();
}

```

```
this.textBox1 = new System.Windows.Forms.TextBox();
this.textBox2 = new System.Windows.Forms.TextBox();
this.bindingSource1 = new System.Windows.Forms.BindingSource(this.components);
((System.ComponentModel.ISupportInitialize)(this.bindingNavigator1)).BeginInit();
this.bindingNavigator1.SuspendLayout();
((System.ComponentModel.ISupportInitialize)(this.bindingSource1)).BeginInit();
this.SuspendLayout();
//
// bindingNavigator1
//
this.bindingNavigator1.AddNewItem = this.bindingNavigatorAddNewItem;
this.bindingNavigator1.CountItem = this.bindingNavigatorCountItem;
this.bindingNavigator1.CountItemFormat = "of {0}";
this.bindingNavigator1.DeleteItem = this.bindingNavigatorDeleteItem;
this.bindingNavigator1.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
this.bindingNavigatorMoveFirstItem,
this.bindingNavigatorMovePreviousItem,
this.bindingNavigatorSeparator,
this.bindingNavigatorPositionItem,
this.bindingNavigatorCountItem,
this.bindingNavigatorSeparator1,
this.bindingNavigatorMoveNextItem,
this.bindingNavigatorMoveLastItem,
this.bindingNavigatorSeparator2,
this.bindingNavigatorAddNewItem,
this.bindingNavigatorDeleteItem});
this.bindingNavigator1.Location = new System.Drawing.Point(0, 0);
this.bindingNavigator1.MoveFirstItem = this.bindingNavigatorMoveFirstItem;
this.bindingNavigator1.MoveLastItem = this.bindingNavigatorMoveLastItem;
this.bindingNavigator1.MoveNextItem = this.bindingNavigatorMoveNextItem;
this.bindingNavigator1.MovePreviousItem = this.bindingNavigatorMovePreviousItem;
this.bindingNavigator1.Name = "bindingNavigator1";
this.bindingNavigator1.PositionItem = this.bindingNavigatorPositionItem;
this.bindingNavigator1.TabIndex = 2;
this.bindingNavigator1.Text = "bindingNavigator1";
//
// bindingNavigatorAddNewItem
//
this.bindingNavigatorAddNewItem.Image =
((System.Drawing.Image)(resources.GetObject("bindingNavigatorAddNewItem.Image")));
this.bindingNavigatorAddNewItem.Name = "bindingNavigatorAddNewItem";
this.bindingNavigatorAddNewItem.Text = "Add new";
//
// bindingNavigatorCountItem
//
this.bindingNavigatorCountItem.Name = "bindingNavigatorCountItem";
this.bindingNavigatorCountItem.Text = "of {0}";
this.bindingNavigatorCountItem.ToolTipText = "Total number of items";
//
// bindingNavigatorDeleteItem
//
this.bindingNavigatorDeleteItem.Image =
((System.Drawing.Image)(resources.GetObject("bindingNavigatorDeleteItem.Image")));
this.bindingNavigatorDeleteItem.Name = "bindingNavigatorDeleteItem";
this.bindingNavigatorDeleteItem.Text = "Delete";
//
// bindingNavigatorMoveFirstItem
//
this.bindingNavigatorMoveFirstItem.Image =
((System.Drawing.Image)(resources.GetObject("bindingNavigatorMoveFirstItem.Image")));
this.bindingNavigatorMoveFirstItem.Name = "bindingNavigatorMoveFirstItem";
this.bindingNavigatorMoveFirstItem.Text = "Move first";
//
// bindingNavigatorMovePreviousItem
//
this.bindingNavigatorMovePreviousItem.Image =
((System.Drawing.Image)(resources.GetObject("bindingNavigatorMovePreviousItem.Image")));
this.bindingNavigatorMovePreviousItem.Name = "bindingNavigatorMovePreviousItem";
this.bindingNavigatorMovePreviousItem.Text = "Move previous";
```

```
        this.bindingNavigatorPositionItem1.DisplayStyle =
            System.Windows.Forms.ToolStripItemDisplayStyle.ImageAndText;
        this.bindingNavigatorPositionItem1.Name = "bindingNavigatorPositionItem";
        this.bindingNavigatorPositionItem1.Size = new System.Drawing.Size(50, 25);
        this.bindingNavigatorPositionItem1.Text = "0";
        this.bindingNavigatorPositionItem1.ToolTipText = "Current position";
    //
    // bindingNavigatorSeparator1
    //
    this.bindingNavigatorSeparator1.Name = "bindingNavigatorSeparator1";

    // bindingNavigatorMoveNextItem
    //
    this.bindingNavigatorMoveNextItem.Image =
        ((System.Drawing.Image)(resources.GetObject("bindingNavigatorMoveNextItem.Image")));
    this.bindingNavigatorMoveNextItem.Name = "bindingNavigatorMoveNextItem";
    this.bindingNavigatorMoveNextItem.Text = "Move next";
    //
    // bindingNavigatorMoveLastItem
    //
    this.bindingNavigatorMoveLastItem.Image =
        ((System.Drawing.Image)(resources.GetObject("bindingNavigatorMoveLastItem.Image")));
    this.bindingNavigatorMoveLastItem.Name = "bindingNavigatorMoveLastItem";
    this.bindingNavigatorMoveLastItem.Text = "Move last";
    //
    // bindingNavigatorSeparator2
    //
    this.bindingNavigatorSeparator2.Name = "bindingNavigatorSeparator2";
    //
    // textBox1
    //
    this.textBox1.Location = new System.Drawing.Point(46, 64);
    this.textBox1.Name = "textBox1";
    this.textBox1.TabIndex = 3;
    //
    // textBox2
    //
    this.textBox2.Location = new System.Drawing.Point(46, 104);
    this.textBox2.Name = "textBox2";
    this.textBox2.TabIndex = 4;
    //
    // Form1
    //
    this.ClientSize = new System.Drawing.Size(292, 266);
    this.Controls.Add(this.textBox2);
    this.Controls.Add(this.textBox1);
    this.Controls.Add(this.bindingNavigator1);
    this.Name = "Form1";
    ((System.ComponentModel.ISupportInitialize)(this.bindingNavigator1)).EndInit();
    this.bindingNavigator1.ResumeLayout(false);
    this.bindingNavigator1.PerformLayout();
    ((System.ComponentModel.ISupportInitialize)(this.bindingSource1)).EndInit();
    this.ResumeLayout(false);
    this.PerformLayout();
}
}
```

```
Imports System  
Imports System.Collections.Generic
```

```

Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Windows.Forms
Imports System.Xml
Imports System.IO
Imports System.Text


Class Form1
    Inherits Form
    Private components As.IContainer
    Private bindingNavigator1 As BindingNavigator
    Private bindingNavigatorAddNewItem As ToolStripButton
    Private bindingNavigatorCountItem As ToolStripLabel
    Private bindingNavigatorDeleteItem As ToolStripButton
    Private bindingNavigatorMoveFirstItem As ToolStripButton
    Private bindingNavigatorMovePreviousItem As ToolStripButton
    Private bindingNavigatorSeparator As ToolStripSeparator
    Private bindingNavigatorPositionItem As ToolStripTextBox
    Private bindingNavigatorSeparator1 As ToolStripSeparator
    Private bindingNavigatorMoveNextItem As ToolStripButton
    Private bindingNavigatorMoveLastItem As ToolStripButton
    Private textBox1 As TextBox
    Private textBox2 As TextBox
    Private bindingSource1 As BindingSource
    Private bindingNavigatorSeparator2 As ToolStripSeparator


    Public Sub New()
        InitializeComponent()
        LoadData()

        bindingNavigator1.BindingSource = bindingSource1
    End Sub 'New

    Private Sub LoadData()
        ' The xml to bind to.
        Dim xml As String = "<US><states>" + "<state><name>Washington</name><capital>Olympia</capital></state>" +
        "<state><name>Oregon</name><capital>Salem</capital></state>" + "<state><name>California</name>
<capital>Sacramento</capital></state>" + "<state><name>Nevada</name><capital>Carson City</capital></state>" + "
</states></US>"

        ' Convert the xml string to bytes and load into a memory stream.
        Dim xmlBytes As Byte() = Encoding.UTF8.GetBytes(xml)
        Dim stream As New MemoryStream(xmlBytes, False)

        ' Create a DataSet and load the xml into it.
        Dim [set] As New DataSet()
        [set].ReadXml(stream)

        ' Set the DataSource to the DataSet, and the DataMember
        ' to state.
        bindingSource1.DataSource = [set]
        bindingSource1.DataMember = "state"

        textBox1.DataBindings.Add("Text", bindingSource1, "name")
        textBox2.DataBindings.Add("Text", bindingSource1, "capital")
    End Sub 'LoadData

    <STAThread()> _
Public Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())
End Sub 'Main

```

```

Private Sub InitializeComponent()
    Me.components = New System.ComponentModel.Container()
    Dim resources As New System.ComponentModel.ComponentResourceManager(GetType(Form1))
    Me.bindingNavigator1 = New System.Windows.Forms.BindingNavigator(Me.components)
    Me.bindingNavigatorAddNewItem = New System.Windows.Forms.ToolStripButton()
    Me.bindingNavigatorCountItem = New System.Windows.Forms.ToolStripLabel()
    Me.bindingNavigatorDeleteItem = New System.Windows.Forms.ToolStripButton()
    Me.bindingNavigatorMoveFirstItem = New System.Windows.Forms.ToolStripButton()
    Me.bindingNavigatorMovePreviousItem = New System.Windows.Forms.ToolStripButton()
    Me.bindingNavigatorSeparator = New System.Windows.Forms.ToolStripSeparator()
    Me.bindingNavigatorPositionItem = New System.Windows.Forms.ToolStripTextBox()
    Me.bindingNavigatorSeparator1 = New System.Windows.Forms.ToolStripSeparator()
    Me.bindingNavigatorMoveNextItem = New System.Windows.Forms.ToolStripButton()
    Me.bindingNavigatorMoveLastItem = New System.Windows.Forms.ToolStripButton()
    Me.bindingNavigatorSeparator2 = New System.Windows.Forms.ToolStripSeparator()
    Me.textBox1 = New System.Windows.Forms.TextBox()
    Me.textBox2 = New System.Windows.Forms.TextBox()
    Me.bindingSource1 = New System.Windows.Forms.BindingSource(Me.components)
    CType(Me.bindingNavigator1, System.ComponentModel.ISupportInitialize).BeginInit()
    Me.bindingNavigator1.SuspendLayout()
    CType(Me.bindingSource1, System.ComponentModel.ISupportInitialize).BeginInit()
    Me.SuspendLayout()
    '
    ' bindingNavigator1
    '
    '
    ' Me.bindingNavigator1.AddNewItem = Me.bindingNavigatorAddNewItem
    ' Me.bindingNavigator1.CountItem = Me.bindingNavigatorCountItem
    ' Me.bindingNavigator1.CountItemFormat = "of {0}"
    ' Me.bindingNavigator1.DeleteItem = Me.bindingNavigatorDeleteItem
    ' Me.bindingNavigator1.Items.AddRange(New System.Windows.Forms.ToolStripItem())
    {'Me.bindingNavigatorMoveFirstItem, Me.bindingNavigatorMovePreviousItem, Me.bindingNavigatorSeparator,
    Me.bindingNavigatorPositionItem, Me.bindingNavigatorCountItem, Me.bindingNavigatorSeparator1,
    Me.bindingNavigatorMoveNextItem, Me.bindingNavigatorMoveLastItem, Me.bindingNavigatorSeparator2,
    Me.bindingNavigatorAddNewItem, Me.bindingNavigatorDeleteItem})
    Me.bindingNavigator1.Location = New System.Drawing.Point(0, 0)
    Me.bindingNavigator1.MoveFirstItem = Me.bindingNavigatorMoveFirstItem
    Me.bindingNavigator1.MoveLastItem = Me.bindingNavigatorMoveLastItem
    Me.bindingNavigator1.MoveNextItem = Me.bindingNavigatorMoveNextItem
    Me.bindingNavigator1.MovePreviousItem = Me.bindingNavigatorMovePreviousItem
    Me.bindingNavigator1.Name = "bindingNavigator1"
    Me.bindingNavigator1.PositionItem = Me.bindingNavigatorPositionItem
    Me.bindingNavigator1.TabIndex = 2
    Me.bindingNavigator1.Text = "bindingNavigator1"
    '
    ' bindingNavigatorAddNewItem
    '
    '
    ' Me.bindingNavigatorAddNewItem.Image = CType(resources.GetObject("bindingNavigatorAddNewItem.Image"),
    System.Drawing.Image)
    Me.bindingNavigatorAddNewItem.Name = "bindingNavigatorAddNewItem"
    Me.bindingNavigatorAddNewItem.Text = "Add new"
    '
    ' bindingNavigatorCountItem
    '
    '
    ' Me.bindingNavigatorCountItem.Name = "bindingNavigatorCountItem"
    ' Me.bindingNavigatorCountItem.Text = "of {0}"
    ' Me.bindingNavigatorCountItem.ToolTipText = "Total number of items"
    '
    ' bindingNavigatorDeleteItem
    '
    '
    ' Me.bindingNavigatorDeleteItem.Image = CType(resources.GetObject("bindingNavigatorDeleteItem.Image"),
    System.Drawing.Image)
    Me.bindingNavigatorDeleteItem.Name = "bindingNavigatorDeleteItem"
    Me.bindingNavigatorDeleteItem.Text = "Delete"
    '
    ' bindingNavigatorMoveFirstItem
    '
    '
    ' Me.bindingNavigatorMoveFirstItem.Image =

```

```
CType(resources.GetObject("bindingNavigatorMoveFirstItem.Image"), System.Drawing.Image)
Me.bindingNavigatorMoveFirstItem.Name = "bindingNavigatorMoveFirstItem"
Me.bindingNavigatorMoveFirstItem.Text = "Move first"
'
' bindingNavigatorMovePreviousItem
'
Me.bindingNavigatorMovePreviousItem.Image =
CType(resources.GetObject("bindingNavigatorMovePreviousItem.Image"), System.Drawing.Image)
Me.bindingNavigatorMovePreviousItem.Name = "bindingNavigatorMovePreviousItem"
Me.bindingNavigatorMovePreviousItem.Text = "Move previous"
'
' bindingNavigatorSeparator
'
Me.bindingNavigatorSeparator.Name = "bindingNavigatorSeparator"
'
' bindingNavigatorPositionItem
'
Me.bindingNavigatorPositionItem.DisplayStyle =
System.Windows.Forms.ToolStripItemDisplayStyle.ImageAndText
Me.bindingNavigatorPositionItem.Name = "bindingNavigatorPositionItem"
Me.bindingNavigatorPositionItem.Size = New System.Drawing.Size(50, 25)
Me.bindingNavigatorPositionItem.Text = "0"
Me.bindingNavigatorPositionItem.ToolTipText = "Current position"
'
' bindingNavigatorSeparator1
'
Me.bindingNavigatorSeparator1.Name = "bindingNavigatorSeparator1"
'
' bindingNavigatorMoveNextItem
'
Me.bindingNavigatorMoveNextItem.Image =
CType(resources.GetObject("bindingNavigatorMoveNextItem.Image"), System.Drawing.Image)
Me.bindingNavigatorMoveNextItem.Name = "bindingNavigatorMoveNextItem"
Me.bindingNavigatorMoveNextItem.Text = "Move next"
'
' bindingNavigatorMoveLastItem
'
Me.bindingNavigatorMoveLastItem.Image =
CType(resources.GetObject("bindingNavigatorMoveLastItem.Image"), System.Drawing.Image)
Me.bindingNavigatorMoveLastItem.Name = "bindingNavigatorMoveLastItem"
Me.bindingNavigatorMoveLastItem.Text = "Move last"
'
' bindingNavigatorSeparator2
'
Me.bindingNavigatorSeparator2.Name = "bindingNavigatorSeparator2"
'
' textBox1
'
Me.textBox1.Location = New System.Drawing.Point(46, 64)
Me.textBox1.Name = "textBox1"
Me.textBox1.TabIndex = 3
'
' textBox2
'
Me.textBox2.Location = New System.Drawing.Point(46, 104)
Me.textBox2.Name = "textBox2"
Me.textBox2.TabIndex = 4
'
' Form1
'
Me.ClientSize = New System.Drawing.Size(292, 266)
Me.Controls.Add(textBox2)
Me.Controls.Add(textBox1)
Me.Controls.Add(bindingNavigator1)
Me.Name = "Form1"
 CType(Me.bindingNavigator1, System.ComponentModel.ISupportInitialize).EndInit()
Me.bindingNavigator1.ResumeLayout(False)
```

```
Me.bindingNavigator1.PerformLayout()  
 CType(Me.bindingSource1, System.ComponentModel.ISupportInitialize).EndInit()  
 Me.ResumeLayout(False)  
 Me.PerformLayout()  
  
 End Sub 'InitializeComponent  
End Class 'Form1
```

Compiling the Code

This example requires:

- References to the System, System.Data, System.Drawing, System.Windows.Forms and System.Xml assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[BindingNavigator](#)

[BindingNavigator Control](#)

[ToolStrip Control](#)

How to: Move Through a DataSet with the Windows Forms BindingNavigator Control

5/4/2018 • 3 min to read • [Edit Online](#)

As you build data-driven applications, you will often need to display collections of data to users. The [BindingNavigator](#) control, in conjunction with the [BindingSource](#) component, provides a convenient and extensible solution for moving through a collection and displaying items sequentially.

Example

The following code example demonstrates how to use a [BindingNavigator](#) control to move through data. The set is contained in a [DataGridView](#), which is bound to a [TextBox](#) control with a [BindingSource](#) component.

NOTE

Storing sensitive information, such as a password, within the connection string can affect the security of your application. Using Windows Authentication (also known as integrated security) is a more secure way to control access to a database. For more information, see [Protecting Connection Information](#).

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Data.SqlClient;
using System.Windows.Forms;

// This form demonstrates using a BindingNavigator to display
// rows from a database query sequentially.
public class Form1 : Form
{
    // This is the BindingNavigator that allows the user
    // to navigate through the rows in a DataSet.
    BindingNavigator customersBindingNavigator = new BindingNavigator(true);

    // This is the BindingSource that provides data for
    // the TextBox control.
    BindingSource customersBindingSource = new BindingSource();

    // This is the TextBox control that displays the CompanyName
    // field from the DataSet.
    TextBox companyNameTextBox = new TextBox();

    public Form1()
    {
        // Set up the BindingSource component.
        this.customersBindingNavigator.BindingSource = this.customersBindingSource;
        this.customersBindingNavigator.Dock = DockStyle.Top;
        this.Controls.Add(this.customersBindingNavigator);

        // Set up the TextBox control for displaying company names.
        this.companyNameTextBox.Dock = DockStyle.Bottom;
        this.Controls.Add(this.companyNameTextBox);

        // Set up the form.
        this.Size = new Size(800, 200);
    }
}
```

```

        this.Load += new EventHandler(Form1_Load);
    }

    void Form1_Load(object sender, EventArgs e)
    {
        // Open a connection to the database.
        // Replace the value of connectString with a valid
        // connection string to a Northwind database accessible
        // to your system.
        string connectString =
            "Integrated Security=SSPI;Persist Security Info=False;" +
            "Initial Catalog=Northwind;Data Source=localhost";

        using (SqlConnection connection = new SqlConnection(connectString))
        {

            SqlDataAdapter dataAdapter1 =
                new SqlDataAdapter(new SqlCommand("Select * From Customers",connection));
            DataSet ds = new DataSet("Northwind Customers");
            ds.Tables.Add("Customers");
            dataAdapter1.Fill(ds.Tables["Customers"]);

            // Assign the DataSet as the DataSource for the BindingSource.
            this.customersBindingSource.DataSource = ds.Tables["Customers"];

            // Bind the CompanyName field to the TextBox control.
            this.companyNameTextBox.DataBindings.Add(
                new Binding("Text",
                this.customersBindingSource,
                "CompanyName",
                true));
        }
    }

    [STAThread]
    public static void Main()
    {
        Application.EnableVisualStyles();
        Application.Run(new Form1());
    }
}

```

```

Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Data.SqlClient
Imports System.Windows.Forms

' This form demonstrates using a BindingNavigator to display
' rows from a database query sequentially.
Public Class Form1
    Inherits Form
    ' This is the BindingNavigator that allows the user
    ' to navigate through the rows in a DataSet.
    Private customersBindingNavigator As New BindingNavigator(True)

    ' This is the BindingSource that provides data for
    ' the Textbox control.
    Private customersBindingSource As New BindingSource()

    ' This is the TextBox control that displays the CompanyName
    ' field from the DataSet.
    Private companyNameTextBox As New TextBox()

```

```

Public Sub New()
    ' Set up the BindingSource component.
    Me.customersBindingNavigator.BindingSource = Me.customersBindingSource
    Me.customersBindingNavigator.Dock = DockStyle.Top
    Me.Controls.Add(Me.customersBindingNavigator)

    ' Set up the TextBox control for displaying company names.
    Me.companyNameTextBox.Dock = DockStyle.Bottom
    Me.Controls.Add(Me.companyNameTextBox)

    ' Set up the form.
    Me.Size = New Size(800, 200)
    AddHandler Me.Load, AddressOf Form1_Load

End Sub 'New

Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs)
    ' Open a connection to the database.
    ' Replace the value of connectString with a valid
    ' connection string to a Northwind database accessible
    ' to your system.
    Dim connectString As String =
        "Integrated Security=SSPI;Persist Security Info=False;" & _
        "Initial Catalog=Northwind;Data Source=localhost"

    Dim connection As New SqlConnection(connectString)
    Try

        Dim dataAdapter1 As New SqlDataAdapter( _
            New SqlCommand("Select * From Customers", connection))
        Dim ds As New DataSet("Northwind Customers")
        ds.Tables.Add("Customers")
        dataAdapter1.Fill(ds.Tables("Customers"))

        ' Assign the DataSet as the DataSource for the BindingSource.
        Me.customersBindingSource.DataSource = ds.Tables("Customers")

        ' Bind the CompanyName field to the TextBox control.
        Me.companyNameTextBox.DataBindings.Add(New Binding("Text", _
            Me.customersBindingSource, "CompanyName", True))
    Finally
        connection.Dispose()
    End Try

End Sub 'Form1_Load

<STAThread()> _
Public Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())

    End Sub
End Class

```

Compiling the Code

This example requires:

- References to the System, System.Data, System.Drawing, System.Windows.Forms and System.Xml assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code](#)

[Example Using Visual Studio.](#)

See Also

[BindingSource](#)

[DataGridView](#)

[BindingSource](#)

[BindingNavigator Control](#)

[BindingSource Component](#)

[How to: Bind a Windows Forms Control to a Type](#)

How to: Add Load, Save, and Cancel Buttons to the Windows Forms BindingNavigator Control

5/4/2018 • 2 min to read • [Edit Online](#)

The [BindingNavigator](#) control is a special-purpose [ToolStrip](#) control that is intended for navigating and manipulating controls on your form that are bound to data.

Because it is a [ToolStrip](#) control, the [BindingNavigator](#) component can be easily modified to include additional or alternative commands for the user.

In the following procedure, a [TextBox](#) control is bound to data, and the [ToolStrip](#) control that is added to the form is modified to include load, save, and cancel buttons.

To add load, save, and cancel buttons to the BindingNavigator component

1. Add a [TextBox](#) control to your form.
2. Bind it to a [BindingSource](#), which is bound to a data source. For this example, the [BindingSource](#) is bound to a database.
3. After the dataset and table adapter are generated, drag a [BindingNavigator](#) control to the form.
4. Set the [BindingNavigator](#) control's [BindingSource](#) property to the [BindingSource](#) on the form that is bound to the controls.
5. Select the [BindingNavigator](#) control.
6. Click the smart tag glyph (ⓘ) so the **BindingNavigator Tasks** dialog appears and select **Edit Items**.

The **Items Collection Editor** appears.

7. In the **Items Collection Editor**, complete the following:
 - a. Add a [ToolStripSeparator](#) and three [ToolStripButton](#) items by selecting the appropriate type of [ToolStripItem](#) and clicking the **Add** button.
 - b. Set the [Name](#) property of the buttons to **LoadButton**, **SaveButton**, and **CancelButton**, respectively.
 - c. Set the [Text](#) property of the buttons to **Load**, **Save**, and **Cancel**.
 - d. Set the [DisplayStyle](#) property for each of the buttons to **Text**. Alternatively, you can set this property to **Image** or **ImageAndText** and set the image to be displayed in the [Image](#) property.
 - e. Click **OK** to close the dialog box. The buttons are added to the [ToolStrip](#).
8. Right-click the form and choose **View Code**.
9. In the Code Editor, find the line of code that loads data into the table adapter. This code was generated when you set up the data binding in step 2. The code should be similar to the following:

```
TableAdapterName.Fill(DataSetName.TableName)
```

It will most likely be in the form's **Load** event.
10. Create an event handler for the [Click](#) event of the **LoadToolStripButton** you created earlier and move this data-loading code into it.

Your code should now look similar to the following:

```
Private Sub LoadButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles LoadButton.Click
    TableAdapterName.Fill(DataSetName.TableName)
End Sub
```

```
private void LoadButton_Click(System.Object sender,
    System.EventArgs e)
{
    TableAdapterName.Fill(DataSetName.TableName);
}
```

11. Create an event handler for the [Click](#) event of the [SaveToolStripButton](#) you created earlier and write code to update the data within the table the control is bound to.

```
Private Sub SaveButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SaveButton.Click
    TableAdapterName.Update(DataSetName.TableName)
End Sub
```

```
private void SaveButton_Click(System.Object sender,
    System.EventArgs e)
{
    TableAdapterName.Update(DataSetName.TableName);
}
```

NOTE

In some cases, the [BindingNavigator](#) component will already have a [Save](#) button, but no code will have been generated by the Windows Forms Designer. In this case, you can place the preceding code in the [Click](#) event handler for that button, rather than creating an entirely new button on the [ToolStrip](#). However, the button is disabled by default, so you must set the [Enabled](#) property of the button to `true` to have the button function correctly.

12. Create an event handler for the [Click](#) event of the [CancelToolStripButton](#) you created earlier and write code to cancel any changes to the data record that is displayed.

```
Private Sub CancelButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CancelButton.Click
    BindingSourceName.CancelEdit()
End Sub
```

```
private void CancelButton_Click(System.Object sender, System.EventArgs e)
{
    BindingSourceName.CancelEdit();
}
```

NOTE

The [CancelEdit](#) method is scoped to the row of data. Save any changes you make while viewing that individual record before navigating to the next record.

See Also

[BindingNavigator](#)

[BindingSource](#)

[ToolStrip](#)

[BindingNavigator Control](#)

[BindingSource Component Overview](#)

BindingSource Component

5/4/2018 • 2 min to read • [Edit Online](#)

Encapsulates a data source for binding to controls.

The [BindingSource](#) component serves two purposes. First, it provides a layer of indirection when binding the controls on a form to data. This is accomplished by binding the [BindingSource](#) component to your data source, and then binding the controls on your form to the [BindingSource](#) component. All further interaction with the data, including navigating, sorting, filtering, and updating, is accomplished with calls to the [BindingSource](#) component.

Second, the [BindingSource](#) component can act as a strongly typed data source. Adding a type to the [BindingSource](#) component with the [Add](#) method creates a list of that type.

In This Section

[BindingSource Component Overview](#)

Introduces the general concepts of the [BindingSource](#) component, which allows you to bind a data source to a control.

[How to: Bind Windows Forms Controls to DBNull Database Values](#)

Shows how to handle a [DBNull](#) value from the data source using the [BindingSource](#) component.

[How to: Sort and Filter ADO.NET Data with the Windows Forms BindingSource Component](#)

Demonstrates using the [BindingSource](#) component to apply sorts and filters to displayed data.

[How to: Bind to a Web Service Using the Windows Forms BindingSource](#)

Shows how to use the [BindingSource](#) component to bind to a Web service.

[How to: Handle Errors and Exceptions that Occur with Databinding](#)

Demonstrates using the [BindingSource](#) component to gracefully handle errors that occur in a data binding operation.

[How to: Bind a Windows Forms Control to a Type](#)

Demonstrates using a [BindingSource](#) component to bind to a type.

[How to: Bind a Windows Forms Control to a Factory Object](#)

Demonstrates using a [BindingSource](#) component to bind to a factory object or method.

[How to: Customize Item Addition with the Windows Forms BindingSource](#)

Demonstrates using a [BindingSource](#) component to create new items and add them to a data source.

[How to: Raise Change Notifications Using the BindingSource ResetItem Method](#)

Demonstrates using a [BindingSource](#) component to raise change-notification events for data sources that do not support change notification.

[How to: Raise Change Notifications Using a BindingSource and the INotifyPropertyChanged Interface](#)

Demonstrates how to use a type that inherits from the [INotifyPropertyChanged](#) with a [BindingSource](#) control.

[How to: Reflect Data Source Updates in a Windows Forms Control with the BindingSource](#)

Demonstrates how to respond to changes in the data source using the [BindingSource](#) component.

[How to: Share Bound Data Across Forms Using the BindingSource Component](#)

Shows how to use the [BindingSource](#) to bind multiple forms to the same data source.

Reference

[BindingSource](#)

Provides reference documentation for the [BindingSource](#) component.

[BindingNavigator](#)

Provides reference documentation for the [BindingNavigator](#) control.

Related Sections

[Windows Forms Data Binding](#)

Contains links to topics describing the Windows Forms data binding architecture.

Also see [Bind controls to data in Visual Studio](#).

BindingSource Component Overview

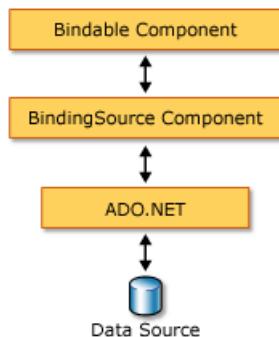
5/4/2018 • 3 min to read • [Edit Online](#)

The [BindingSource](#) component is designed to simplify the process of binding controls to an underlying data source. The [BindingSource](#) component acts as both a conduit and a data source for other controls to bind to. It provides an abstraction of your form's data connection while passing through commands to the underlying list of data. Additionally, you can add data directly to it, so that the component itself functions as a data source.

BindingSource Component as an Intermediary

The [BindingSource](#) component acts as the data source for some or all of the controls on the form. In Visual Studio, the [BindingSource](#) can be bound to a control by means of the [DataBindings](#) property, which is accessible from the **Properties** window. Also see [How to: Bind Windows Forms Controls with the BindingSource Component Using the Designer](#).

You can bind the [BindingSource](#) component to both simple data sources, like a single property of an object or a basic collection like [ArrayList](#), and complex data sources, like a database table. The [BindingSource](#) component acts as an intermediary that provides binding and currency management services. At design time or run time, you can bind a [BindingSource](#) component to a complex data source by setting its [DataSource](#) and [DataMember](#) properties to the database and table, respectively. The following illustration demonstrates where the [BindingSource](#) component fits into the existing data-binding architecture.



NOTE

At design time, some actions, like dragging a database table from a data window onto a blank form, will create the [BindingSource](#) component, bind it to the underlying data source, and add data-aware controls all in one operation. Also see [Bind Windows Forms controls to data in Visual Studio](#).

BindingSource Component as a Data Source

If you start adding items to the [BindingSource](#) component without first specifying a list to be bound to, the component will act like a list-style data source and accept these added items.

Additionally, you can write code to provide custom "AddNew" functionality by means of the [AddingNew](#) event, which is raised when the [AddNew](#) method is called prior to the item being added to the list. For more information, see [BindingSource Component Architecture](#).

Navigation

For users that need to navigate the data on a form, the [BindingNavigator](#) component enables you to navigate and

manipulate data, in coordination with a [BindingSource](#) component. For more information, see [BindingNavigator Control](#).

Data Manipulation

The: [BindingSource](#) acts as a [CurrencyManager](#) for all of its bindings and can, therefore, provide access to currency and position information regarding the data source. The following table shows the members that the [BindingSource](#) component provides for accessing and manipulating the underlying data.

MEMBER	DESCRIPTION
Current property	Gets the current item of the data source.
Position property	Gets or sets the current position in the underlying list.
List property	Gets the list that is the evaluation of the DataSource and DataMember evaluation. If DataMember is not set, returns the list specified by DataSource .
Insert method	Inserts an item in the list at the specified index.
RemoveCurrent method	Removes the current item from the list.
EndEdit method	Applies pending changes to the underlying data source.
CancelEdit method	Cancels the current edit operation.
AddNew method	Adds a new item to the underlying list. If the data source implements IBindingList and returns an item from the AddingNew event, adds this item. Otherwise, the request is passed to the list's AddNew method. If the underlying list is not an IBindingList , the item is automatically created through its public default constructor.

Sorting and Filtering

Usually, you should work with an ordered or filtered view of the data source. The following table shows the members that the [BindingSource](#) component data source provides.

MEMBER	DESCRIPTION
Sort property	If the data source is an IBindingList , gets or sets a column name used for sorting and sort order information. If the data source is an IBindingListView and supports advanced sorting, gets multiple column names used for sorting and sort order information
Filter property	If the data source is an IBindingListView , gets or sets the expression used to filter which rows are viewed.

See Also

[BindingSource](#)
[BindingNavigator](#)
[BindingSource Component Architecture](#)

[BindingSource Component](#)

[BindingNavigator Control](#)

[Windows Forms Data Binding](#)

[Controls to Use on Windows Forms](#)

BindingSource Component Architecture

5/4/2018 • 7 min to read • [Edit Online](#)

With the [BindingSource](#) component, you can universally bind all Windows Forms controls to data sources.

The [BindingSource](#) component simplifies the process of binding controls to a data source and provides the following advantages over traditional data binding:

- Enables design-time binding to business objects.
- Encapsulates [CurrencyManager](#) functionality and exposes [CurrencyManager](#) events at design time.
- Simplifies creating a list that supports the [IBindingList](#) interface by providing list change notification for data sources that do not natively support list change notification.
- Provides an extensibility point for the [IBindingList.AddNew](#) method.
- Provides a level of indirection between the data source and the control. This indirection is important when the data source may change at run time.
- Interoperates with other data-related Windows Forms controls, specifically the [BindingNavigator](#) and the [DataGridView](#) controls.

For these reasons, the [BindingSource](#) component is the preferred way to bind your Windows Forms controls to data sources.

BindingSource Features

The [BindingSource](#) component provides several features for binding controls to data. With these features, you can implement most data-binding scenarios with almost no coding on your part.

The [BindingSource](#) component accomplishes this by providing a consistent interface for accessing many different kinds of data sources. This means that you use the same procedure for binding to any type. For example, you can attach the [DataSource](#) property to a [DataSet](#) or to a business object and in both cases you use the same set of properties, methods, and events to manipulate the data source.

The consistent interface provided by the [BindingSource](#) component greatly simplifies the process of binding data to controls. For data-source types that provide change notification, the [BindingSource](#) component automatically communicates changes between the control and the data source. For data-source types that do not provide change notification, events are provided that let you raise change notifications. The following list shows the features supported by the [BindingSource](#) component:

- Indirection.
- Currency management.
- Data source as a list.
- [BindingSource](#) as an [IBindingList](#).
- Custom item creation.
- Transactional item creation.
- [IEnumerable](#) support.

- Design-time support.
- Static [ListBindingHelper](#) methods.
- Sorting and filtering with the [IBindingListView](#) interface.
- Integration with [BindingNavigator](#).

Indirection

The [BindingSource](#) component provides a level of indirection between a control and a data source. Instead of binding a control directly to a data source, you bind the control to a [BindingSource](#), and you attach the data source to the [BindingSource](#) component's [DataSource](#) property.

With this level of indirection, you can change the data source without resetting the control binding. This gives you the following capabilities:

- You can attach the [BindingSource](#) to different data sources while retaining the current control bindings.
- You can change items in the data source and notify bound controls. For more information, see [How to: Reflect Data Source Updates in a Windows Forms Control with the BindingSource](#).
- You can bind to a [Type](#) instead of an object in memory. For more information, see [How to: Bind a Windows Forms Control to a Type](#). You can then bind to an object at run time.

Currency Management

The [BindingSource](#) component implements the [ICurrencyManagerProvider](#) interface to handle currency management for you. With the [ICurrencyManagerProvider](#) interface, you can also access to the currency manager for a [BindingSource](#), in addition to the currency manager for another [BindingSource](#) bound to the same [DataMember](#).

The [BindingSource](#) component encapsulates [CurrencyManager](#) functionality and exposes the most common [CurrencyManager](#) properties and events. The following table describes some of the members related to currency management.

[CurrencyManager](#) property

Gets the currency manager associated with the [BindingSource](#).

[GetRelatedCurrencyManager](#) method

If there is another [BindingSource](#) bound to the specified data member, gets its currency manager.

[Current](#) property

Gets the current item of the data source.

[Position](#) property

Gets or sets the current position in the underlying list.

[EndEdit](#) method

Applies pending changes to the underlying data source.

[CancelEdit](#) method

Cancels the current edit operation.

Data Source as a List

The [BindingSource](#) component implements the [IBindingListView](#) and [ITypedList](#) interfaces. With this implementation, you can use the [BindingSource](#) component itself as a data source, without any external storage.

When the [BindingSource](#) component is attached to a data source, it exposes the data source as a list.

The [DataSource](#) property can be set to several data sources. These include types, objects, and lists of types. The resulting data source will be exposed as a list. The following table shows some of the common data sources and

the resulting list evaluation.

DATA SOURCE PROPERTY	LIST RESULTS
A null reference (<code>Nothing</code> in Visual Basic)	An empty IBindingList of objects. Adding an item sets the list to the type of the added item.
A null reference (<code>Nothing</code> in Visual Basic) with DataMember set	Not supported; raises ArgumentException .
Non-list type or object of type "T"	An empty IBindingList of type "T".
Array instance	An IBindingList containing the array elements.
IEnumerable instance	An IBindingList containing the IEnumerable items
List instance containing type "T"	An IBindingList instance containing type "T".

Additionally, [DataSource](#) can be set to other list types, such as [IListSource](#) and [ITypedList](#), and the [BindingSource](#) will handle them appropriately. In this case, the type that is contained in the list should have a default constructor.

BindingSource as an [IBindingList](#)

The [BindingSource](#) component provides members for accessing and manipulating the underlying data as an [IBindingList](#). The following table describes some of these members.

MEMBER	DESCRIPTION
List property	Gets the list that results from the evaluation of the DataSource or DataMember properties.
AddNew method	Adds a new item to the underlying list. Applies to data sources that implement the IBindingList interface and allow adding items (that is, the AllowNew property is set to <code>true</code>).

Custom Item Creation

You can handle the [AddingNew](#) event to provide your own item-creation logic. The [AddingNew](#) event occurs before a new object is added to the [BindingSource](#). This event is raised after the [AddNew](#) method is called, but before the new item is added to the underlying list. By handling this event, you can provide custom item creation behavior without deriving from the [BindingSource](#) class. For more information, see [How to: Customize Item Addition with the Windows Forms BindingSource](#).

Transactional Item Creation

The [BindingSource](#) component implements the [ICancelAddNew](#) interface, which enables transactional item creation. After a new item is provisionally created by using a call to [AddNew](#), the addition may be committed or rolled back in the following ways:

- The [EndNew](#) method will explicitly commit the pending addition.
- Performing another collection operation, such as an insertion, removal, or move, will implicitly commit the pending addition.
- The [CancelNew](#) method will roll back the pending addition if the method has not already been committed.

[IEnumerable](#) Support

The [BindingSource](#) component enables binding controls to [IEnumerable](#) data sources. With this component, you

can bind to a data source such as a [System.Data.SqlClient.SqlDataReader](#).

When an [IEnumerable](#) data source is assigned to the [BindingSource](#) component, the [BindingSource](#) creates an [IBindingList](#) and adds the contents of the [IEnumerable](#) data source to the list.

Design-Time Support

Some object types cannot be created at design time, such as objects created from a factory class, or objects returned by a Web service. You may sometimes have to bind your controls to these types at design time, even though there is no object in memory to which your controls can bind. You may, for example, need to label the column headers of a [DataGridView](#) control with the names of your custom type's public properties.

To support this scenario, the [BindingSource](#) component supports binding to a [Type](#). When you assign a [Type](#) to the [DataSource](#) property, the [BindingSource](#) component creates an empty [BindingList<T>](#) of [Type](#) items. Any controls you subsequently bind to the [BindingSource](#) component will be alerted to the presence of the properties or schema of your type at design time, or at run time. For more information, see [How to: Bind a Windows Forms Control to a Type](#).

Static ListBindingHelper Methods

The [System.Windows.Forms.BindingContext](#), [System.Windows.Forms.CurrencyManager](#), and [BindingSource](#) types all share common logic to generate a list from a [DataSource](#) / [DataMember](#) pair. Additionally, this common logic is publicly exposed for use by control authors and other third parties in the following [static](#) methods:

- [GetListItemProperties](#)
- [GetList](#).
- [GetListName](#)
- [GetListItemType](#)

Sorting and Filtering with the [IBindingListView](#) Interface

The [BindingSource](#) component implements the [IBindingListView](#) interface, which extends the [IBindingList](#) interface. The [IBindingList](#) offers single column sorting and the [IBindingListView](#) offers advanced sorting and filtering. With [IBindingListView](#), you can sort and filter items in the data source, if the data source also implements one of these interfaces. The [BindingSource](#) component does not provide a reference implementation of these members. Instead, calls are forwarded to the underlying list.

The following table describes the properties you use for sorting and filtering.

MEMBER	DESCRIPTION
Filter property	If the data source is an IBindingListView , gets or sets the expression used to filter which rows are viewed.
Sort property	If the data source is an IBindingList , gets or sets a column name used for sorting and sort order information. -or- If the data source is an IBindingListView and supports advanced sorting, gets multiple column names used for sorting and sort order

Integration with [BindingNavigator](#)

You can use the [BindingSource](#) component to bind any Windows Forms control to a data source, but the [BindingNavigator](#) control is designed specifically to work with the [BindingSource](#) component. The [BindingNavigator](#) control provides a user interface for controlling the [BindingSource](#) component's current item. By

default, the [BindingNavigator](#) control provides buttons that correspond to the navigation methods on the [BindingSource](#) component. For more information, see [How to: Navigate Data with the Windows Forms BindingNavigator Control](#).

See Also

[BindingSource](#)

[BindingNavigator](#)

[BindingSource Component Overview](#)

[BindingNavigator Control](#)

[Windows Forms Data Binding](#)

[Controls to Use on Windows Forms](#)

[How to: Bind a Windows Forms Control to a Type](#)

[How to: Reflect Data Source Updates in a Windows Forms Control with the BindingSource](#)

How to: Bind a Windows Forms Control to a Factory Object

5/4/2018 • 7 min to read • [Edit Online](#)

When you are building controls that interact with data, you will sometimes find it necessary to bind a control to an object or method that generates other objects. Such an object or method is called a factory. Your data source might be, for example, the return value from a method call, instead of an object in memory or a type. You can bind a control to this kind of data source as long as the source returns a collection.

You can easily bind a control to a factory object by using the [BindingSource](#) control.

Example

The following example demonstrates how to bind a [DataGridView](#) control to a factory method by using a [BindingSource](#) control. The factory method is named `GetOrdersByCustomerId`, and it returns all the orders for a given customer in the Northwind database.

```
#using <System.dll>
#using <System.Data.dll>
#using <System.Drawing.dll>
#using <System.EnterpriseServices.dll>
#using <System.Transactions.dll>
#using <System.Windows.Forms.dll>
#using <System.Xml.dll>
using namespace System;
using namespace System::Collections;
using namespace System::Collections::Generic;
using namespace System::ComponentModel;
using namespace System::Data;
using namespace System::Data::Common;
using namespace System::Data::SqlClient;
using namespace System::Diagnostics;
using namespace System::Drawing;
using namespace System::Windows::Forms;

// This form demonstrates using a BindingSource to bind to a factory
// object.
public ref class Form1: public System::Windows::Forms::Form
{
private:

    // This is the TextBox for entering CustomerID values.
    static TextBox^ customerIdTextBox = gcnew TextBox;

    // This is the DataGridView that displays orders for the
    // specified customer.
    static DataGridView^ customersDataGridView = gcnew DataGridView;

    // This is the BindingSource for binding the database query
    // result set to the DataGridView.
    static BindingSource^ ordersBindingSource = gcnew BindingSource;

public:
    Form1()
    {
        // Set up the CustomerID TextBox.
        this->customerIdTextBox->Dock = DockStyle::Bottom;
        this->customerIdTextBox->Text =
    }
}
```

```

L"Enter a valid Northwind CustomerID, for example: ALFKI,"

L" then TAB or click outside the TextBox";
this->customerIdTextBox->Leave += gcnew EventHandler(
    this, &Form1::customerIdTextBox_Leave );
this->Controls->Add( this->customerIdTextBox );

// Set up the DataGridView.
customersDataGridView->Dock = DockStyle::Top;
this->Controls->Add( customersDataGridView );

// Set up the form.
this->Size = System::Drawing::Size( 800, 800 );
this->Load += gcnew EventHandler( this, &Form1::Form1_Load );
}

private:
// This event handler binds the BindingSource to the DataGridView
// control's DataSource property.
void Form1_Load(
    System::Object^ /*sender*/,
    System::EventArgs^ /*e*/ )
{
// Attach the BindingSource to the DataGridView.
this->customersDataGridView->DataSource =
    this->ordersBindingSource;
}

public:
// This is a static factory method. It queries the Northwind
// database for the orders belonging to the specified
// customer and returns an IList.
static System::Collections::IList^ GetOrdersByCustomerId( String^ id )
{
// Open a connection to the database.
String^ connectString = L"Integrated Security=SSPI;" +
    L"Persist Security Info=False;Initial Catalog=Northwind;" +
    L"Data Source= localhost";
SqlConnection^ connection = gcnew SqlConnection;
connection->ConnectionString = connectString;
connection->Open();

// Execute the query.
String^ queryString = String::Format(
    L"Select * From Orders where CustomerID = '{0}'", id );
SqlCommand^ command = gcnew SqlCommand( queryString, connection );
SqlDataReader^ reader = command->ExecuteReader(
    CommandBehavior::CloseConnection );

// Build an IList from the result set.
List< DbDataRecord^ >^ list = gcnew List< DbDataRecord^ >;
System::Collections::IEnumerator^ e = reader->GetEnumerator();
while ( e->MoveNext() )
{
    list->Add( dynamic_cast<DbDataRecord^>(e->Current) );
}

return list;
}

// This event handler is called when the user tabs or clicks
// out of the customerIdTextBox. The database is then queried
// with the CustomerID in the customerIdTextBox.Text property.
private:
void customerIdTextBox_Leave( Object^ /*sender*/, EventArgs^ /*e*/ )
{
// Attach the data source to the BindingSource control.
this->ordersBindingSource->DataSource = GetOrdersByCustomerId(
    this->customerIdTextBox->Text );
}

```

```

}

public:
[STAThread]
static void main()
{
    Application::EnableVisualStyles();
    Application::Run( gcnew Form1 );
}
};


```

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.Common;
using System.Diagnostics;
using System.Drawing;
using System.Data.SqlClient;
using System.Windows.Forms;

// This form demonstrates using a BindingSource to bind to a factory
// object.
public class Form1 : System.Windows.Forms.Form
{
    // This is the TextBox for entering CustomerID values.
    private TextBox customerIdTextBox = new TextBox();

    // This is the DataGridView that displays orders for the
    // specified customer.
    private DataGridView customersDataGridView = new DataGridView();

    // This is the BindingSource for binding the database query
    // result set to the DataGridView.
    private BindingSource ordersBindingSource = new BindingSource();

    public Form1()
    {
        // Set up the CustomerID TextBox.
        this.customerIdTextBox.Location = new Point(100, 200);
        this.customerIdTextBox.Size = new Size(500, 30);
        this.customerIdTextBox.Text =
            "Enter a valid Northwind CustomerID, for example: ALFKI," +
            " then RETURN or click outside the TextBox";
        this.customerIdTextBox.Leave +=
            new EventHandler(customerIdTextBox_Leave);
        this.customerIdTextBox.KeyDown +=
            new KeyEventHandler(customerIdTextBox_KeyDown);
        this.Controls.Add(this.customerIdTextBox);

        // Set up the DataGridView.
        customersDataGridView.Dock = DockStyle.Top;
        this.Controls.Add(customersDataGridView);

        // Set up the form.
        this.Size = new Size(800, 500);
        this.Load += new EventHandler(Form1_Load);
    }

    // This event handler binds the BindingSource to the DataGridView
    // control's DataSource property.
    private void Form1_Load(
        System.Object sender,
        System.EventArgs e)
    {
        // Attach the BindingSource to the DataGridView.

```

```

        this.customersDataGridView.DataSource =
            this.ordersBindingSource;
    }

    // This is a static factory method. It queries the Northwind
    // database for the orders belonging to the specified
    // customer and returns an IEnumerable.
    public static IEnumerable GetOrdersByCustomerId(string id)
    {
        // Open a connection to the database.
        string connectString = "Integrated Security=SSPI;" +
            "Persist Security Info=False;Initial Catalog=Northwind;" +
            "Data Source= localhost";
        SqlConnection connection = new SqlConnection();

        connection.ConnectionString = connectString;
        connection.Open();

        // Execute the query.
        string queryString =
            String.Format("Select * From Orders where CustomerID = '{0}'",
            id);
        SqlCommand command = new SqlCommand(queryString, connection);
        SqlDataReader reader =
            command.ExecuteReader(CommandBehavior.CloseConnection);
        return reader;
    }

    // These event handlers are called when the user tabs or clicks
    // out of the customerIdTextBox or hits the return key.
    // The database is then queried with the CustomerID
    // in the customerIdTextBox.Text property.
    void customerIdTextBox_Leave(object sender, EventArgs e)
    {
        // Attach the data source to the BindingSource control.
        this.ordersBindingSource.DataSource =
            GetOrdersByCustomerId(this.customerIdTextBox.Text);
    }

    void customerIdTextBox_KeyDown(object sender, KeyEventArgs e)
    {
        if (e.KeyCode == Keys.Return)
        {
            // Attach the data source to the BindingSource control.
            this.ordersBindingSource.DataSource =
                GetOrdersByCustomerId(this.customerIdTextBox.Text);
        }
    }

    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.Run(new Form1());
    }
}

```

```

Imports System
Imports System.Collections
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Data
Imports System.Data.Common
Imports System.Diagnostics
Imports System.Drawing
Imports System.Data.SqlClient

```

```

Imports System.Windows.Forms

' This form demonstrates using a BindingSource to bind to a factory
' object.

Public Class Form1
    Inherits System.Windows.Forms.Form
    ' This is the TextBox for entering CustomerID values.
    Private WithEvents customerIdTextBox As New TextBox()

    ' This is the DataGridView that displays orders for the
    ' specified customer.
    Private customersDataGridView As New DataGridView()

    ' This is the BindingSource for binding the database query
    ' result set to the DataGridView.
    Private ordersBindingSource As New BindingSource()

    Public Sub New()

        ' Set up the CustomerID TextBox.
        Me.customerIdTextBox.Location = New Point(100, 200)
        Me.customerIdTextBox.Size = New Size(500, 30)
        Me.customerIdTextBox.Text = _
            "Enter a valid Northwind CustomerID, for example: ALFKI," & _
            " then RETURN or click outside the TextBox"
        Me.Controls.Add(Me.customerIdTextBox)

        ' Set up the DataGridView.
        customersDataGridView.Dock = DockStyle.Top
        Me.Controls.Add(customersDataGridView)

        ' Set up the form.
        Me.Size = New Size(800, 500)
    End Sub

    ' This event handler binds the BindingSource to the DataGridView
    ' control's DataSource property.
    Private Sub Form1_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles Me.Load

        ' Attach the BindingSource to the DataGridView.
        Me.customersDataGridView.DataSource = Me.ordersBindingSource
    End Sub

    ' This is a static factory method. It queries the Northwind
    ' database for the orders belonging to the specified
    ' customer and returns an IEnumerable.
    Public Shared Function GetOrdersByCustomerId(ByVal id As String) _
        As IEnumerable

        ' Open a connection to the database.
        Dim connectionString As String = "Integrated Security=SSPI;" & _
            "Persist Security Info=False;Initial Catalog=Northwind;" & _
            "Data Source= localhost"
        Dim connection As New SqlConnection()

        connection.ConnectionString = connectionString
        connection.Open()

        ' Execute the query.
        Dim queryString As String = _
            String.Format("Select * From Orders where CustomerID = '{0}'", id)
        Dim command As New SqlCommand(queryString, connection)
        Dim reader As SqlDataReader = _
            command.ExecuteReader(CommandBehavior.CloseConnection)
        Return reader
    End Function

```

```

End Function

' These event handlers are called when the user tabs or clicks
' out of the customerIdTextBox or hits the return key.
' The database is then queried with the CustomerID
' in the customerIdTextBox.Text property.
Private Sub customerIdTextBox_Leave(ByVal sender As Object, _
    ByVal e As EventArgs) Handles customerIdTextBox.Leave

    ' Attach the data source to the BindingSource control.
    Me.ordersBindingSource.DataSource = _
        GetOrdersByCustomerId(Me.customerIdTextBox.Text)

End Sub

Private Sub customerIdTextBox_KeyDown(ByVal sender As Object, _
    ByVal e As KeyEventArgs) Handles customerIdTextBox.KeyDown

    If e.KeyCode = Keys.Return Then

        ' Attach the data source to the BindingSource control.
        Me.ordersBindingSource.DataSource = _
            GetOrdersByCustomerId(Me.customerIdTextBox.Text)
    End If

End Sub

<STAThread()> _
Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())

End Sub
End Class

```

Compiling the Code

This example requires:

- References to the System, System.Data, System.Drawing and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[BindingNavigator](#)

[DataGridView](#)

[BindingSource](#)

[BindingSource Component](#)

[How to: Bind a Windows Forms Control to a Type](#)

How to: Bind a Windows Forms Control to a Type

5/4/2018 • 3 min to read • [Edit Online](#)

When you are building controls that interact with data, you will sometimes find it necessary to bind a control to a type, rather than an object. This situation arises especially at design time, when data may not be available, but your data-bound controls still need to display information from a type's public interface. For example, you may bind a [DataGridView](#) control to an object exposed by a Web service and want the [DataGridView](#) control to label its columns at design time with the member names of a custom type.

You can easily bind a control to a type with the [BindingSource](#) component.

Example

The following code example demonstrates how to bind a [DataGridView](#) control to a custom type by using a [BindingSource](#) component. When you run the example, you'll notice the [DataGridView](#) has labeled columns that reflect the properties of a `Customer` object, before the control is populated with data. The example has an Add Customer button to add data to the [DataGridView](#) control. When you click the button, a new `Customer` object is added to the [BindingSource](#). In a real-world scenario, the data might be obtained by a call to a Web service or other data source.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Windows.Forms;

class Form1 : Form
{
    BindingSource bSource = new BindingSource();
    private Button button1;
    DataGridView dgv = new DataGridView();

    public Form1()
    {
        this.button1 = new System.Windows.Forms.Button();
        this.button1.Location = new System.Drawing.Point(140, 326);
        this.button1.Name = "button1";
        this.button1.AutoSize = true;
        this.button1.Text = "Add Customer";
        this.button1.Click += new System.EventHandler(this.button1_Click);
        this.ClientSize = new System.Drawing.Size(362, 370);
        this.Controls.Add(this.button1);

        // Bind the BindingSource to the DemoCustomer type.
        bSource.DataSource = typeof(DemoCustomer);

        // Set up the DataGridView control.
        dgv.Dock = DockStyle.Top;
        this.Controls.Add(dgv);

        // Bind the DataGridView control to the BindingSource.
        dgv.DataSource = bSource;
    }

    public static void Main()
    {
        Application.Run(new Form1());
    }
}
```

```

        }

    private void button1_Click(object sender, EventArgs e)
    {
        bSource.Add(new DemoCustomer(DateTime.Today));
    }
}

// This simple class is used to demonstrate binding to a type.
public class DemoCustomer
{
    public DemoCustomer()
    {
        idValue = Guid.NewGuid();
    }

    public DemoCustomer(DateTime FirstOrderDate)
    {
        FirstOrder = FirstOrderDate;
        idValue = Guid.NewGuid();
    }

    // These fields hold the data that backs the public properties.
    private DateTime firstOrderDateValue;
    private Guid idValue;
    private string custNameValue;

    public string CustomerName
    {
        get { return custNameValue; }
        set { custNameValue = value; }
    }

    // This is a property that represents a birth date.
    public DateTime FirstOrder
    {
        get
        {
            return this.firstOrderDateValue;
        }
        set
        {
            if (value != this.firstOrderDateValue)
            {
                this.firstOrderDateValue = value;
            }
        }
    }

    // This is a property that represents a customer ID.
    public Guid ID
    {
        get
        {
            return this.idValue;
        }
    }
}

```

```

Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Windows.Forms

Class Form1

```

```

Inherits Form
Private bSource As New BindingSource()
Private WithEvents button1 As Button
Private dgv As New DataGridView()

Public Sub New()
    Me.button1 = New System.Windows.Forms.Button()
    Me.button1.Location = New System.Drawing.Point(140, 326)
    Me.button1.Name = "button1"
    Me.button1.AutoSize = True
    Me.button1.Text = "Add Customer"
    Me.ClientSize = New System.Drawing.Size(362, 370)
    Me.Controls.Add(Me.button1)

    ' Bind the BindingSource to the DemoCustomer type.
    bSource.DataSource = GetType(DemoCustomer)

    ' Set up the DataGridView control.
    dgv.Dock = DockStyle.Top
    Me.Controls.Add(dgv)

    ' Bind the DataGridView control to the BindingSource.
    dgv.DataSource = bSource

End Sub

Public Shared Sub Main()
    Application.Run(New Form1())
End Sub

Private Sub button1_Click(ByVal sender As Object, ByVal e As EventArgs) _
Handles button1.Click
    bSource.Add(New DemoCustomer(DateTime.Today))
End Sub
End Class

' This simple class is used to demonstrate binding to a type.
Public Class DemoCustomer

    Public Sub New()
        idValue = Guid.NewGuid()
    End Sub

    Public Sub New(ByVal FirstOrderDate As DateTime)
        FirstOrder = FirstOrderDate
        idValue = Guid.NewGuid()
    End Sub

    ' These fields hold the data that backs the public properties.
    Private firstOrderDateValue As DateTime
    Private idValue As Guid
    Private custNameValue As String

    Public Property CustomerName() As String
        Get
            Return custNameValue
        End Get
        Set(ByVal value As String)
            custNameValue = value
        End Set
    End Property

    ' This is a property that represents the first order date.
    Public Property FirstOrder() As DateTime
        Get
            Return Me.firstOrderDateValue
        End Get
    End Property

```

```
Set(ByVal value As DateTime)
    If value <> Me.firstOrderDateValue Then
        Me.firstOrderDateValue = value
    End If
End Set
End Property

' This is a property that represents a customer ID.
Public ReadOnly Property ID() As Guid
    Get
        Return Me.idValue
    End Get
End Property
End Class
```

Compiling the Code

This example requires:

- References to the System and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[BindingNavigator](#)

[DataGridView](#)

[BindingSource](#)

[BindingSource Component](#)

How to: Bind a Windows Forms Control to a Type Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

When you are building controls that interact with data, you sometimes need to bind a control to a type, rather than an object. You typically need to bind a control to a type at design time, when data may not be available, but you still want your data-bound controls to display data from a type's public interface. The following procedures demonstrate how to create a new [BindingSource](#) that is bound to a type, and then how to bind one of the type's properties to the [Text](#) property of a [TextBox](#).

To bind the BindingSource to a type

1. Create a Windows Forms project.

For more information, see [How to: Create a Windows Application Project](#).

2. In **Design** view, drag a [BindingSource](#) component onto the form.
3. In the **Properties** window, click the arrow for the [DataSource](#) property.
4. In the **DataSource UI Type Editor**, click **Add Project Data Source**.
5. On the **Choose a Data Source Type** page, select **Object** and click **Next**.
6. Select the type to bind to:
 - If the type you want to bind to is in the current project, or the assembly that contains the type is already added as a reference, expand the nodes to find the type you want, and then select it.
-or-
 - If the type you want to bind to is in another assembly, not currently in the list of references, click **Add Reference**, and then click the **Projects** tab. Select the project that contains the business object you want and click **OK**. This project will appear in the list of assemblies, so you can expand the nodes to find the type you want, and then select it.

NOTE

If you want to bind to a type in a framework or Microsoft assembly, clear the **Hide assemblies that begin with Microsoft or System** check box.

7. Click **Next**, and then click **Finish**.

To bind the control to the BindingSource

1. Add a [TextBox](#) to the form.
2. In the **Properties** window, expand the **(DataBindings)** node.
3. Click the arrow next to the [Text](#) property.
4. In the **DataSource UI Type Editor**, expand the node for the [BindingSource](#) added previously, and select the property of the bound type you want to bind to the [Text](#) property of the [TextBox](#).

See Also

[BindingSource Component](#)

[How to: Bind a Windows Forms Control to a Type](#)

[Bind controls to data in Visual Studio](#)

How to: Bind to a Web Service Using the Windows Forms BindingSource

5/4/2018 • 10 min to read • [Edit Online](#)

If you want to bind a Windows Form control to the results obtained from calling an XML Web service, you can use a [BindingSource](#) component. This procedure is similar to binding a [BindingSource](#) component to a type. You must create a client-side proxy that contains the methods and types exposed by the Web service. You generate a client-side proxy from the Web service (.asmx) itself, or its Web Services Description Language (WSDL) file. Additionally, your client-side proxy must expose the fields of complex types used by the Web service as public properties. You then bind the [BindingSource](#) to one of the types exposed in the Web service proxy.

To create and bind to a client-side proxy

1. Create a Windows Form in the directory of your choice, with an appropriate namespace.
2. Add a [BindingSource](#) component to the form.
3. Open the Windows Software Development Kit (SDK) command prompt, and navigate to the same directory that your form is located in.
4. Using the WSDL tool, enter `wsdl` and the URL for the .asmx or WSDL file for the Web service, followed by the namespace of your application, and optionally the language you are working in.

The following code example uses the Web service located at

<http://webservices.eraserver.net/zipcoderesolver/zipcoderesolver.asmx>. For example, for C# type

```
wsdl http://webservices.eraserver.net.zipcoderesolver.zipcoderesolver.asmx /n:BindToWebService , or for
```

Visual Basic type

```
wsdl http://webservices.eraserver.net.zipcoderesolver.zipcoderesolver.asmx /n:BindToWebService  
/language:VB
```

. Passing the path as an argument to the WSDL tool will generate a client-side proxy in the same directory and namespace as your application, in the specified language. If you are using Visual Studio, add the file to your project.

5. Select a type in the client-side proxy to bind to.

This is typically a type returned by a method offered by the Web service. The fields of the chosen type must be exposed as public properties for binding purposes.

```
[System::SerializableAttribute, System::Xml::Serialization::XmlTypeAttribute(  
Namespace="http://webservices.eraserver.net/")]  
public ref class USPSAddress  
{  
  
private:  
    String^ streetField;  
  
    String^ cityField;  
  
    String^ stateField;  
  
    String^ shortZIPField;  
  
    String^ fullZIPField;  
  
public:
```

```

property String^ Street
{
    String^ get()
    {
        return this->streetField;
    }
    void set( String^ value )
    {
        this->streetField = value;
    }
}

property String^ City
{
    String^ get()
    {
        return this->cityField;
    }
    void set( String^ value )
    {
        this->cityField = value;
    }
}
property String^ State
{
    String^ get()
    {
        return this->stateField;
    }
    void set( String^ value )
    {
        this->stateField = value;
    }
}

property String^ ShortZIP
{
    String^ get()
    {
        return this->shortZIPField;
    }
    void set( String^ value )
    {
        this->shortZIPField = value;
    }
}

property String^ FullZIP
{
    String^ get()
    {
        return this->fullZIPField;
    }
    void set( String^ value )
    {
        this->fullZIPField = value;
    }
};


```

```

[System.SerializableAttribute, System.Xml.Serialization.XmlTypeAttribute(
Namespace="http://webservices.eraserver.net/")]
public class USPSAddress
{

    private string streetField;

```

```
private string cityField;

private string stateField;

private string shortZIPField;

private string fullZIPField;

public string Street
{
    get
    {
        return this.streetField;
    }
    set
    {
        this.streetField = value;
    }
}

public string City
{
    get
    {
        return this.cityField;
    }
    set
    {
        this.cityField = value;
    }
}

public string State
{
    get
    {
        return this.stateField;
    }
    set
    {
        this.stateField = value;
    }
}

public string ShortZIP
{
    get
    {
        return this.shortZIPField;
    }
    set
    {
        this.shortZIPField = value;
    }
}

public string FullZIP
{
    get
    {
        return this.fullZIPField;
    }
    set
    {
```

```
    this.fullZIPField = value;
}
}
}
```

```

<System.SerializableAttribute(), _
System.Xml.Serialization.XmlTypeAttribute( _
[Namespace]:="http://webservices.eraserver.net/")> _
Public Class USPSAddress

    Private streetField As String

    Private cityField As String

    Private stateField As String

    Private shortZIPField As String

    Private fullZIPField As String


    Public Property Street() As String
        Get
            Return Me.streetField
        End Get
        Set(ByVal value As String)
            Me.streetField = value
        End Set
    End Property

    Public Property City() As String
        Get
            Return Me.cityField
        End Get
        Set(ByVal value As String)
            Me.cityField = value
        End Set
    End Property

    Public Property State() As String
        Get
            Return Me.stateField
        End Get
        Set(ByVal value As String)
            Me.stateField = value
        End Set
    End Property

    Public Property ShortZIP() As String
        Get
            Return Me.shortZIPField
        End Get
        Set(ByVal value As String)
            Me.shortZIPField = value
        End Set
    End Property

    Public Property FullZIP() As String
        Get
            Return Me.fullZIPField
        End Get
        Set(ByVal value As String)
            Me.fullZIPField = value
        End Set
    End Property
End Class

```

6. Set the **DataSource** property of the **BindingSource** to the type you want that is contained in the Web service

client-side proxy.

```
BindingSource1->DataSource = USPSAddress::typeid;
```

```
BindingSource1.DataSource = typeof(USPSAddress);
```

```
BindingSource1.DataSource = GetType(USPSAddress)
```

To bind controls to the BindingSource that is bound to a Web service

- Bind controls to the [BindingSource](#), passing the public property of the Web service type that you want as a parameter.

```
textBox1->DataBindings->Add("Text", this->BindingSource1, "FullZIP", true);
```

```
textBox1.DataBindings.Add("Text", this.BindingSource1, "FullZIP", true);
```

```
textBox1.DataBindings.Add("Text", Me.BindingSource1, "FullZIP", True)
```

Example

The following code example demonstrates how to bind a [BindingSource](#) component to a Web service, and then how to bind a text box to the [BindingSource](#) component. When you click the button, a Web service method is called and the results will appear in `textbox1`.

```
#using <System.Windows.Forms.dll>
#using <System.dll>
#using <System.Drawing.dll>
#using <System.Web.Services.dll>
#using <System.Xml.dll>

using namespace System;
using namespace System::Collections::Generic;
using namespace System::ComponentModel;
using namespace System::Drawing;
using namespace System::Windows::Forms;

namespace BindToWebService {

[System::SerializableAttribute, System::Xml::Serialization::XmlTypeAttribute(
    Namespace="http://webservices.eraserver.net/")]
public ref class USPSAddress
{

private:
    String^ streetField;

    String^ cityField;

    String^ stateField;

    String^ shortZIPField;

    String^ fullZIPField;
```

```

public:
    property String^ Street
    {
        String^ get()
        {
            return this->streetField;
        }
        void set( String^ value )
        {
            this->streetField = value;
        }
    }

    property String^ City
    {
        String^ get()
        {
            return this->cityField;
        }
        void set( String^ value )
        {
            this->cityField = value;
        }
    }
    property String^ State
    {
        String^ get()
        {
            return this->stateField;
        }
        void set( String^ value )
        {
            this->stateField = value;
        }
    }

    property String^ ShortZIP
    {
        String^ get()
        {
            return this->shortZIPField;
        }
        void set( String^ value )
        {
            this->shortZIPField = value;
        }
    }

    property String^ FullZIP
    {
        String^ get()
        {
            return this->fullZIPField;
        }
        void set( String^ value )
        {
            this->fullZIPField = value;
        }
    }
};

[System::Web::Services::WebServiceBindingAttribute(Name="ZipCodeResolverSoap",
Namespace="http://webservices.eraserver.net/")]
public ref class ZipCodeResolver:
public System::Web::Services::Protocols::SoapHttpClientProtocol

```

```

1

public:
ZipCodeResolver() : SoapHttpClientProtocol()
{
this->Url =
"http://webservices.eraserver.net/zipcoderesolver/zipcoderesolver.asmx";
}

//''<remarks/>
[System::Web::Services::Protocols::SoapDocumentMethodAttribute
("http://webservices.eraserver.net/CorrectedAddressXml",
RequestNamespace="http://webservices.eraserver.net/",
ResponseNamespace="http://webservices.eraserver.net/",
Use=System::Web::Services::Description::SoapBindingUse::Literal,
ParameterStyle=System::Web::Services::Protocols::SoapParameterStyle::Wrapped)]
USPSAddress^ CorrectedAddressXml(String^ accessCode,
String^ address, String^ city, String^ state)
{
array<Object^>^ results = this->Invoke("CorrectedAddressXml",
gcnew array<Object^>{accessCode, address, city, state});
return ((USPSAddress^) results[0]);
}

//''<remarks/>
System::IAsyncResult^ BeginCorrectedAddressXml(String^ accessCode,
String^ address, String^ city, String^ state,
System:: AsyncCallback^ callback, Object^ asyncState)
{
return this->BeginInvoke("CorrectedAddressXml",
gcnew array<Object^>{accessCode, address, city, state}, callback, asyncState);
}

USPSAddress^ EndCorrectedAddressXml(System::IAsyncResult^ asyncResult)
{
array<Object^>^ results = this->EndInvoke(asyncResult);
return ((USPSAddress^) results[0]);
}

};

ref class Form1: public Form
{
public:
[STAThread]
static void Main()
{
Application::EnableVisualStyles();
Application::Run(gcnew Form1());
}

private:
BindingSource^ BindingSource1;

TextBox^ textBox1;

TextBox^ textBox2;

Button^ button1;

public:
Form1()
{
this->Load += gcnew EventHandler(this, &Form1::Form1_Load);
textBox1->Location = System::Drawing::Point(118, 131);
}

```

```

textBox1->ReadOnly = true;
button1->Location = System::Drawing::Point(133, 60);
button1->Click += gcnew EventHandler(this, &Form1::button1_Click);
button1->Text = "Get zipcode";
ClientSize = System::Drawing::Size(292, 266);
Controls->Add(this->button1);
Controls->Add(this->textBox1);
    BindingSource1 = gcnew BindingSource();
    textBox1 = gcnew TextBox();
    textBox2 = gcnew TextBox();
    button1 = gcnew Button();
}

private:
    void button1_Click(Object^ sender, EventArgs^ e)
{
    textBox1->Text = "Calling Web service..";
    ZipCodeResolver^ resolver = gcnew ZipCodeResolver();
    BindingSource1->Add(resolver->CorrectedAddressXml("0",
        "One Microsoft Way", "Redmond", "WA"));

}

public:
    void Form1_Load(Object^ sender, EventArgs^ e)
{
    BindingSource1->DataSource = USPSAddress::typeid;
    textBox1->DataBindings->Add("Text", this->BindingSource1, "FullZIP", true);
}
};

public ref class CorrectedAddressXmlCompletedEventArgs :
    public System::ComponentModel::AsyncCompletedEventArgs

{
private:
    array<Object^>^ results;

internal:
    CorrectedAddressXmlCompletedEventArgs(array<Object^>^ results,
        System::Exception^ exception, bool cancelled, Object^ userState) :
        AsyncCompletedEventArgs(exception, cancelled, userState)
    {
        this->results = results;
    }

public:
    property USPSAddress^ Result
    {
        USPSAddress^ get()
        {
            this->RaiseExceptionIfNecessary();
            return ((USPSAddress^) this->results[0]);
        }
    }

    delegate void CorrectedAddressXmlCompletedEventHandler(Object^ sender,
        CorrectedAddressXmlCompletedEventArgs^ args);

};

int main()
{
    BindToWebService::Form1::Main();
}

```

```
    return 1;  
}
```

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Drawing;  
using System.Windows.Forms;  
  
namespace BindToWebService {  
    class Form1: Form  
  
    {  
        [STAThread]  
        public static void Main()  
        {  
            Application.EnableVisualStyles();  
            Application.Run(new Form1());  
        }  
  
        private BindingSource BindingSource1 = new BindingSource();  
        private TextBox textBox1 = new TextBox();  
        private TextBox textBox2 = new TextBox();  
        private Button button1 = new Button();  
  
        public Form1()  
        {  
            this.Load += new EventHandler(Form1_Load);  
            textBox1.Location = new System.Drawing.Point(118, 131);  
            textBox1.ReadOnly = true;  
            button1.Location = new System.Drawing.Point(133, 60);  
            button1.Click += new EventHandler(button1_Click);  
            button1.Text = "Get zipcode";  
            ClientSize = new System.Drawing.Size(292, 266);  
            Controls.Add(this.button1);  
            Controls.Add(this.textBox1);  
        }  
  
        private void button1_Click(object sender, EventArgs e)  
        {  
  
            textBox1.Text = "Calling Web service..";  
            ZipCodeResolver resolver = new ZipCodeResolver();  
            BindingSource1.Add(resolver.CorrectedAddressXml("0",  
                "One Microsoft Way", "Redmond", "WA"));  
  
        }  
  
        public void Form1_Load(object sender, EventArgs e)  
        {  
            BindingSource1.DataSource = typeof(USPSAddress);  
            textBox1.DataBindings.Add("Text", this.BindingSource1, "FullZIP", true);  
        }  
    }  
  
    [System.Web.Services.WebServiceBindingAttribute(Name="ZipCodeResolverSoap",  
        Namespace="http://webservices.eraserver.net/")]  
    public class ZipCodeResolver:  
        System.Web.Services.Protocols.SoapHttpClientProtocol  
  
    {  
  
        public ZipCodeResolver() : base()  
        {  
            this.Url =  
                "http://webservices.eraserver.net/zippcoderesolver/zippcoderesolver.asmx";  
        }  
    }
```

```

//''<remarks/>
[System.Web.Services.Protocols.SoapDocumentMethodAttribute
 ("http://webservices.eraserver.net/CorrectedAddressXml",
 RequestNamespace="http://webservices.eraserver.net/",
 ResponseNamespace="http://webservices.eraserver.net/",
 Use=System.Web.Services.Description.SoapBindingUse.Literal,
 ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
public USPSAddress CorrectedAddressXml(string accessCode,
 string address, string city, string state)
{
    object[] results = this.Invoke("CorrectedAddressXml",
        new object[]{accessCode, address, city, state});
    return ((USPSAddress) results[0]);
}

//''<remarks/>
public System.IAsyncResult BeginCorrectedAddressXml(string accessCode,
 string address, string city, string state,
 System.AsyncCallback callback, object asyncState)
{
    return this.BeginInvoke("CorrectedAddressXml",
        new object[]{accessCode, address, city, state}, callback, asyncState);
}

public USPSAddress EndCorrectedAddressXml(System.IAsyncResult asyncResult)
{
    object[] results = this.EndInvoke(asyncResult);
    return ((USPSAddress) results[0]);
}

}

[System.SerializableAttribute, System.Xml.Serialization.XmlTypeAttribute(
 Namespace="http://webservices.eraserver.net/")]
public class USPSAddress
{

    private string streetField;

    private string cityField;

    private string stateField;

    private string shortZIPField;

    private string fullZIPField;

    public string Street
    {
        get
        {
            return this.streetField;
        }
        set
        {
            this.streetField = value;
        }
    }

    public string City
    {
        get

```

```

{
    return this.cityField;
}
set
{
    this.cityField = value;
}

public string State
{
get
{
    return this.stateField;
}
set
{
    this.stateField = value;
}
}

public string ShortZIP
{
get
{
    return this.shortZIPField;
}
set
{
    this.shortZIPField = value;
}
}

public string FullZIP
{
get
{
    return this.fullZIPField;
}
set
{
    this.fullZIPField = value;
}
}

public delegate void CorrectedAddressXmlCompletedEventHandler(object sender,
CorrectedAddressXmlCompletedEventArgs args);

public class CorrectedAddressXmlCompletedEventArgs:
System.ComponentModel.AsyncCompletedEventArgs

{
private object[] results;

internal CorrectedAddressXmlCompletedEventArgs(object[] results,
System.Exception exception, bool cancelled, object userState) :
base(exception, cancelled, userState)
{
    this.results = results;
}

public USPSAddress Result
{
get

```

```

    {
        this.RaiseExceptionIfNecessary();
        return ((USPSAddress) this.results[0]);
    }
}
}
}
}

```

```

Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Drawing
Imports System.Windows.Forms

Namespace BindToWebService
    Class Form1
        Inherits Form

        <STAThread()> _
        Shared Sub Main()
            Application.EnableVisualStyles()
            Application.Run(New Form1())
        End Sub

        Private BindingSource1 As New BindingSource()
        Private textBox1 As New TextBox()
        Private textBox2 As New TextBox()
        Private WithEvents button1 As New Button()

        Public Sub New()

            textBox1.Location = New System.Drawing.Point(118, 131)
            textBox1.ReadOnly = True
            button1.Location = New System.Drawing.Point(133, 60)
            button1.Text = "Get zipcode"
            ClientSize = New System.Drawing.Size(292, 266)
            Controls.Add(Me.button1)
            Controls.Add(Me.textBox1)
        End Sub

        Private Sub button1_Click(ByVal sender As Object, ByVal e As EventArgs) _
            Handles button1.Click

            textBox1.Text = "Calling Web service.."
            Dim resolver As New ZipCodeResolver()
            BindingSource1.Add(resolver.CorrectedAddressXml("0", "One Microsoft Way", "Redmond", "WA"))

        End Sub

        Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
            BindingSource1.DataSource = GetType(USPSAddress)
            textBox1.DataBindings.Add("Text", Me.BindingSource1, "FullZIP", True)
        End Sub

    End Class

    <System.Diagnostics.DebuggerStepThrough(), _
    System.ComponentModel.DesignerCategoryAttribute("code"), _
    System.Web.Services.WebServiceBindingAttribute(Name:="ZipCodeResolverSoap", _
    [Namespace]:="http://webservices.eraserver.net/")> _
    Public Class ZipCodeResolver
        Inherits System.Web.Services.Protocols.SoapHttpClientProtocol

        Private CorrectedAddressXmlOperationCompleted As _
            System.Threading.SendOrPostCallback

```

```

Public Sub New()
    MyBase.New()
    Me.Url = _
        "http://webservices.eraserver.net/zipcoderesolver/zipcoderesolver.asmx"
End Sub

Public Event CorrectedAddressXmlCompleted As _
    CorrectedAddressXmlCompletedEventHandler

<System.Web.Services.Protocols.SoapDocumentMethodAttribute( _
    "http://webservices.eraserver.net/CorrectedAddressXml", _
    RequestNamespace:="http://webservices.eraserver.net/", _
    ResponseNamespace:="http://webservices.eraserver.net/", _
    Use:=System.Web.Services.Description.SoapBindingUse.Literal, _
    ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)> _
Public Function CorrectedAddressXml(ByVal accessCode As String, _
    ByVal address As String, ByVal city As String, ByVal state As String) _
    As USPSAddress
    Dim results() As Object = Me.Invoke("CorrectedAddressXml", _
        New Object() {accessCode, address, city, state})
    Return CType(results(0), USPSAddress)
End Function

'''<remarks/>
Public Function BeginCorrectedAddressXml(ByVal accessCode As String, _
    ByVal address As String, ByVal city As String, ByVal state As String, _
    ByVal callback As System.AsyncCallback, ByVal asyncState As Object) _
    As System.IAsyncResult

    Return Me.BeginInvoke("CorrectedAddressXml", _
        New Object() {accessCode, address, city, state}, callback, asyncState)
End Function

Public Function EndCorrectedAddressXml(ByVal asyncResult _ _
    As System.IAsyncResult) As USPSAddress
    Dim results() As Object = Me.EndInvoke(asyncResult)
    Return CType(results(0), USPSAddress)
End Function
End Class

<System.SerializableAttribute(), _
System.Xml.Serialization.XmlTypeAttribute( _
    [Namespace]:="http://webservices.eraserver.net/")> _
Public Class USPSAddress

    Private streetField As String

    Private cityField As String

    Private stateField As String

    Private shortZIPField As String

    Private fullZIPField As String

    Public Property Street() As String
        Get
            Return Me.streetField
        End Get
        Set(ByVal value As String)
            Me.streetField = value
        End Set
    End Property

    Public Property City() As String

```

```

    Get
        Return Me.cityField
    End Get
    Set(ByVal value As String)
        Me.cityField = value
    End Set
End Property

Public Property State() As String
    Get
        Return Me.stateField
    End Get
    Set(ByVal value As String)
        Me.stateField = value
    End Set
End Property

Public Property ShortZIP() As String
    Get
        Return Me.shortZIPField
    End Get
    Set(ByVal value As String)
        Me.shortZIPField = value
    End Set
End Property

Public Property FullZIP() As String
    Get
        Return Me.fullZIPField
    End Get
    Set(ByVal value As String)
        Me.fullZIPField = value
    End Set
End Property
End Class

Public Delegate Sub CorrectedAddressXmlCompletedEventHandler(ByVal sender As Object, _
    ByVal args As CorrectedAddressXmlCompletedEventArgs)

Public Class CorrectedAddressXmlCompletedEventArgs
    Inherits System.ComponentModel.AsyncCompletedEventArgs

    Private results() As Object

    Friend Sub New(ByVal results() As Object, ByVal exception As System.Exception, _
        ByVal cancelled As Boolean, ByVal userState As Object)
        MyBase.New(exception, cancelled, userState)
        Me.results = results
    End Sub

    Public ReadOnly Property Result() As USPSAddress
        Get
            Me.RaiseExceptionIfNecessary()
            Return CType(Me.results(0), USPSAddress)
        End Get
    End Property
End Class

End Namespace

```

Compiling the Code

This is a complete example that includes a `Main` method, and a shortened version of client-side proxy code.

This example requires:

- References to the System, System.Drawing, System.Web.Services, System.Windows.Forms and System.Xml assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[BindingSource Component](#)

[How to: Bind a Windows Forms Control to a Type](#)

How to: Bind Windows Forms Controls to DBNull Database Values

5/4/2018 • 4 min to read • [Edit Online](#)

When you bind Windows Forms controls to a data source and the data source returns a `DBNull` value, you can substitute an appropriate value without handling, formatting, or parsing events. The `NullValue` property will convert `DBNull` to a specified object when formatting or parsing the data source values.

Example

The following example demonstrates how to bind a `DBNull` value in two different situations. The first demonstrates how to set the `NullValue` for a string property; the second demonstrates how to set the `NullValue` for an image property.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Data.SqlClient;
using System.Windows.Forms;

namespace DBNullCS
{
    public class Form1 : Form
    {
        public Form1()
        {
            this.Load += new EventHandler(Form1_Load);
        }

        // The controls and components we need for the form.
        private Button button1;
        private PictureBox pictureBox1;
        private BindingSource bindingSource1;
        private TextBox textBox1;
        private TextBox textBox2;

        // Data table to hold the database data.
        DataTable employeeTable = new DataTable();

        void Form1_Load(object sender, EventArgs e)
        {
            // Basic form setup.
            this.pictureBox1 = new PictureBox();
            this.bindingSource1 = new BindingSource();
            this.textBox1 = new TextBox();
            this.textBox2 = new TextBox();
            this.button1 = new Button();
            this.pictureBox1.Location = new System.Drawing.Point(20, 20);
            this.pictureBox1.Size = new System.Drawing.Size(174, 179);
            this.textBox1.Location = new System.Drawing.Point(25, 215);
            this.textBox1.ReadOnly = true;
            this.textBox2.Location = new System.Drawing.Point(25, 241);
            this.textBox2.ReadOnly = true;
            this.button1.Location = new System.Drawing.Point(200, 103);
            this.button1.Text = "Move Next";
        }
    }
}
```

```

        this.button1.Text = "Move Next";
        this.button1.Click += new System.EventHandler(this.button1_Click);
        this.ClientSize = new System.Drawing.Size(292, 273);
        this.Controls.Add(this.button1);
        this.Controls.Add(this.textBox2);
        this.Controls.Add(this.textBox1);
        this.Controls.Add(this.pictureBox1);
        this.ResumeLayout(false);
        this.PerformLayout();
        this.PerformLayout();

        // Create the connection string and populate the data table
        // with data.
        string connectionString = "Integrated Security=SSPI;" +
        "Persist Security Info = False;Initial Catalog=Northwind;" +
        "Data Source = localhost";
        SqlConnection connection = new SqlConnection();
        connection.ConnectionString = connectionString;
        SqlDataAdapter employeeAdapter =
            new SqlDataAdapter(new SqlCommand("Select * from Employees", connection));
        connection.Open();
        employeeAdapter.Fill(employeeTable);

        // Set the DataSource property of the BindingSource to the employee table.
        bindingSource1.DataSource = employeeTable;

        // Set up the binding to the ReportsTo column.
        Binding reportsToBinding = textBox2.DataBindings.Add("Text", bindingSource1,
            "ReportsTo", true);

        // Set the NullValue property for this binding.
        reportsToBinding.NullValue = "No Manager";

        // Set up the binding for the PictureBox using the Add method, setting
        // the null value in method call.
        pictureBox1.DataBindings.Add("Image", bindingSource1, "Photo", true,
            DataSourceUpdateMode.Never, new Bitmap(typeof(Button), "Button.bmp"));

        // Set up the remaining binding.
        textBox1.DataBindings.Add("Text", bindingSource1, "LastName", true);
    }

    // Move through the data when the button is clicked.
    private void button1_Click(object sender, EventArgs e)
    {
        bindingSource1.MoveNext();
    }
}

[STAThread]
static void Main()
{
    Application.EnableVisualStyles();
    Application.Run(new Form1());
}
}

```

```

Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Text
Imports System.Data.SqlClient
Imports System.Windows.Forms

```

```

Public Class Form1
    Inherits Form

    Public Sub New()
        End Sub

        ' The controls and components we need for the form.
        Private WithEvents button1 As Button
        Private pictureBox1 As PictureBox
        Private bindingSource1 As BindingSource
        Private textBox1 As TextBox
        Private textBox2 As TextBox

        ' Data table to hold the database data.
        Private employeeTable As New DataTable()

        Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs) _
            Handles Me.Load

            ' Basic form setup.
            Me.pictureBox1 = New PictureBox()
            Me.bindingSource1 = New BindingSource()
            Me.textBox1 = New TextBox()
            Me.textBox2 = New TextBox()
            Me.button1 = New Button()
            Me.pictureBox1.Location = New System.Drawing.Point(20, 20)
            Me.pictureBox1.Size = New System.Drawing.Size(174, 179)
            Me.textBox1.Location = New System.Drawing.Point(25, 215)
            Me.textBox1.ReadOnly = True
            Me.textBox2.Location = New System.Drawing.Point(25, 241)
            Me.textBox2.ReadOnly = True
            Me.button1.Location = New System.Drawing.Point(200, 103)
            Me.button1.Text = "Move Next"
            Me.ClientSize = New System.Drawing.Size(292, 273)
            Me.Controls.Add(Me.button1)
            Me.Controls.Add(Me.textBox2)
            Me.Controls.Add(Me.textBox1)
            Me.Controls.Add(Me.pictureBox1)
            Me.ResumeLayout(False)
            Me.PerformLayout()

            ' Create the connection string and populate the data table
            ' with data.
            Dim connectionString As String = "Integrated Security=SSPI;" & _
                "Persist Security Info = False;Initial Catalog=Northwind;" & _
                "Data Source = localhost"
            Dim connection As New SqlConnection()
            connection.ConnectionString = connectionString
            Dim employeeAdapter As New SqlDataAdapter _
                (New SqlCommand("Select * from Employees", connection))
            connection.Open()
            employeeAdapter.Fill(employeeTable)

            ' Set the DataSource property of the BindingSource to the employee table.
            bindingSource1.DataSource = employeeTable

            ' Set up the binding to the ReportsTo column.
            Dim reportsToBinding As Binding = _
                textBox2.DataBindings.Add("Text", bindingSource1, "ReportsTo", _
                True)

            ' Set the NullValue property for this binding.
            reportsToBinding.NullValue = "No Manager"

            ' Set up the binding for the PictureBox using the Add method, setting
            ' the null value in method call.
            pictureBox1.DataBindings.Add("Image", bindingSource1, "Photo", _
                True, DataSourceUpdateMode.Never)

```

```

        True, DataSourceUpdateMode.Never, _
        New Bitmap(GetType(Button), "Button.bmp"))

    ' Set up the remaining binding.
    textBox1.DataBindings.Add("Text", bindingSource1, "LastName", True)

End Sub

' Move through the data when the button is clicked.
Private Sub button1_Click(ByVal sender As Object, _
    ByVal e As EventArgs) Handles button1.Click

    bindingSource1.MoveNext()

End Sub

<STAThread()> _
Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())

End Sub
End Class

```

The types of the bound property and the [NullValue](#) property must be the same or an error will result, and no further [NullValue](#) values will be processed. In this situation, an exception will not be thrown.

Compiling the Code

This example requires:

- References to the System, System.Data, System.Drawing and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[BindingSource Component](#)

[How to: Handle Errors and Exceptions that Occur with Databinding](#)

[How to: Bind a Windows Forms Control to a Type](#)

How to: Bind Windows Forms Controls with the BindingSource Component Using the Designer

5/4/2018 • 2 min to read • [Edit Online](#)

After you have added controls to your form and determined the user interface for your application, you can bind the controls to a data source, so that, at run time, users can alter and save data related to the application.

Binding a control or series of controls in Windows Forms is most easily accomplished using the [BindingSource](#) control as a bridge between the controls on the form and the data source.

One or more controls on a form can be bound to data; in the following procedure, a [TextBox](#) control is bound to a data source.

To complete the procedure, it is assumed that you will bind to a data source derived from a database. For more information on creating data sources from other stores of data, see [Add new data sources](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To bind a control at design time

1. Drag a [TextBox](#) control on to the form.
2. In the **Properties** window:
 - a. Expand the **(DataBindings)** node.
 - b. Click the arrow next to the [Text](#) property.

The **DataSource** UI type editor opens.

If a data source has previously been configured for the project or form, it will appear.
3. Click **Add Project Data Source** to connect to data and create a data source.
4. On the **Data Source Configuration Wizard** welcome page, click **Next**.
5. On the **Choose a Data Source Type** page, select **Database**.
6. On the **Choose Your Data Connection** page, select a data connection from the list of available connections. If your desired data connection is not available select **New Connection** to create a new data connection.
7. Select **Yes, save the connection** to save the connection string in the application configuration file.
8. Select the database objects to bring into your application. In this case, select a field in a table that you would like the [TextBox](#) to display.
9. Replace the default dataset name if you want.
10. Click **Finish**.
11. In the **Properties** window, click the arrow next to the [Text](#) property again. In the **DataSource** UI type editor,

select the name of the field to bind the [TextBox](#) to.

The **DataSource** UI type editor closes and the data set, [BindingSource](#) and table adapter specific to that data connection are added to your form.

See Also

[BindingSource](#)

[BindingNavigator](#)

[Add new data sources](#)

[Data Sources Window](#)

How to: Create a Lookup Table with the Windows Forms BindingSource Component

5/4/2018 • 3 min to read • [Edit Online](#)

A lookup table is a table of data that has a column that displays data from records in a related table. In the following procedures, a [ComboBox](#) control is used to display the field with the foreign-key relationship from the parent to the child table.

To help visualize these two tables and this relationship, here is an example of a parent and child table:

CustomersTable (parent table)

CUSTOMERID	CUSTOMERNAME
712	Paul Koch
713	Tamara Johnston

OrdersTable (child table)

ORDERID	ORDERDATE	CUSTOMERID
903	February 12, 2004	712
904	February 13, 2004	713

In this scenario, one table, *CustomersTable*, stores the actual information you want to display and save. But to save space, the table leaves out data that adds clarity. The other table, *OrdersTable*, contains only appearance-related information about which customer ID number is equivalent to which order date and order ID. There is no mention of the customers' names.

Four important properties are set on the [ComboBox Control](#) control to create the lookup table.

- The [DataSource](#) property contains the name of the table.
- The [DisplayMember](#) property contains the data column of that table that you want to display for the control text (the customer's name).
- The [ValueMember](#) property contains the data column of that table with the stored information (the ID number in the parent table).
- The [SelectedValue](#) property provides the lookup value for the child table, based on the [ValueMember](#).

The procedures below show you how to lay out your form as a lookup table and bind data to the controls on it. To successfully complete the procedures, you must have a data source with parent and child tables that have a foreign-key relationship, as mentioned previously.

To create the user interface

1. From the [ToolBox](#), drag a [ComboBox](#) control onto the form.

This control will display the column from parent table.

2. Drag other controls to display details from the child table. The format of the data in the table should

determine which controls you choose. For more information, see [Windows Forms Controls by Function](#).

3. Drag a [BindingNavigator](#) control onto the form; this will allow you to navigate the data in the child table.

To connect to the data and bind it to controls

1. Select the [ComboBox](#) and click the Smart Task glyph to display the Smart Task dialog box.
2. Select **Use data bound items**.
3. Click the arrow next to the **Data Source** drop-down box. If a data source has previously been configured for the project or form, it will appear; otherwise, complete the following steps (This example uses the Customers and Orders tables of the Northwind sample database and refers to them in parentheses).
 - a. Click **Add Project Data Source** to connect to data and create a data source.
 - b. On the **Data Source Configuration Wizard** welcome page, click **Next**.
 - c. Select **Database** on the **Choose a Data Source Type** page.
 - d. Select a data connection from the list of available connections on the **Choose Your Data Connection** page. If your desired data connection is not available, select **New Connection** to create a new data connection.
 - e. Click **Yes, save the connection** to save the connection string in the application configuration file.
 - f. Select the database objects to bring into your application. In this case, select a parent table and child table (for example, Customers and Orders) with a foreign key relationship.
 - g. Replace the default dataset name if you want.
 - h. Click **Finish**.
4. In the **Display Member** drop-down box, select the column name (for example, ContactName) to be displayed in the combo box.
5. In the **Value Member** drop-down box, select the column (for example, CustomerID) to perform the lookup operation in the child table.
6. In the **Selected Value** drop-down box, navigate to **Project Data Sources** and the dataset you just created that contains the parent and child tables. Select the same property of the child table that is the Value Member of the parent table (for example, Orders.CustomerID). The appropriate [BindingSource](#), data set, and table adapter components will be created and added to the form.
7. Bind the [BindingNavigator](#) control to the [BindingSource](#) of the child table (for example, `OrdersBindingSource`).
8. Bind the controls other than the [ComboBox](#) and [BindingNavigator](#) control to the details fields from the child table's [BindingSource](#) (for example, `OrdersBindingSource`) that you want to display.

See Also

- [BindingSource](#)
- [BindingSource Component](#)
- [ComboBox Control](#)
- [Bind controls to data in Visual Studio](#)

How to: Customize Item Addition with the Windows Forms BindingSource

5/4/2018 • 7 min to read • [Edit Online](#)

When you use a [BindingSource](#) component to bind a Windows Forms control to a data source, you may find it necessary to customize the creation of new items. The [BindingSource](#) component makes this straightforward by providing the [AddingNew](#) event, which is typically raised when the bound control needs to create a new item. Your event handler can provide whatever custom behavior is required (for example, calling a method on a Web service or getting a new object from a class factory).

NOTE

When an item is added by handling the [AddingNew](#) event, the addition cannot be canceled.

Example

The following example demonstrates how to bind a [DataGridView](#) control to a class factory by using a [BindingSource](#) component. When the user clicks the [DataGridView](#) control's new row, the [AddingNew](#) event is raised. The event handler creates a new `DemoCustomer` object, which is assigned to the [AddingNewEventArgs.NewObject](#) property. This causes the new `DemoCustomer` object to be added to the [BindingSource](#) component's list and to be displayed in the new row of the [DataGridView](#) control.

```
#using <System.dll>
#using <System.Drawing.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::ComponentModel;
using namespace System::Drawing;
using namespace System::Globalization;
using namespace System::Windows::Forms;

namespace DataConnectorAddingNewExample
{
    // This class implements a simple customer type.
    public ref class DemoCustomer
    {
    private:
        // These fields hold the values for the public properties.
        Guid idValue;
        String^ customerName;
        String^ companyNameValue;
        String^ phoneNumberValue;

        // The constructor is private to enforce the factory pattern.
        DemoCustomer()
        {
            idValue = Guid::.NewGuid();
            customerName = String::Empty;
            companyNameValue = String::Empty;
            phoneNumberValue = String::Empty;
            customerName = "no data";
            companyNameValue = "no data";
            phoneNumberValue = "no data";
        }
    }
}
```

```

public:
    // This is the public factory method.
    static DemoCustomer^ CreateNewCustomer()
    {
        return gcnew DemoCustomer();
    }

    property Guid ID
    {
        // This property represents an ID, suitable
        // for use as a primary key in a database.
        Guid get()
        {
            return this->idValue;
        }
    }

    property String^ CompanyName
    {
        String^ get()
        {
            return this->companyNameValue;
        }

        void set(String^ value)
        {
            this->companyNameValue = value;
        }
    }

    property String^ PhoneNumber
    {
        String^ get()
        {
            return this->phoneNumberValue;
        }

        void set(String^ value)
        {
            this->phoneNumberValue = value;
        }
    }
};

// This form demonstrates using a BindingSource to provide
// data from a collection of custom types
// to a DataGridView control.
public ref class MainForm: public System::Windows::Forms::Form
{
private:

    // This is the BindingSource that will provide data for
    // the DataGridView control.
    BindingSource^ customersBindingSource;

    // This is the DataGridView control
    // that will display our data.
    DataGridView^ customersDataGridView;

    // Set up the StatusBar for displaying ListChanged events.
    StatusBar^ status;

public:

    MainForm()
    {
        customersBindingSource = gcnew BindingSource;
        customersDataGridView = gcnew DataGridView;
    }
}

```

```

status = gcnew StatusBar;

// Set up the form.
this->Size = System::Drawing::Size(600, 400);
this->Text = "BindingSource.AddingNew sample";
this->Load +=
    gcnew EventHandler(this, &MainForm::OnMainFormLoad);
this->Controls->Add(status);

// Set up the DataGridView control.
this->customersDataGridView->Dock = DockStyle::Fill;
this->Controls->Add(this->customersDataGridView);

// Attach an event handler for the AddingNew event.
this->customersBindingSource->AddingNew +=
    gcnew AddingNewEventHandler(this,
        &MainForm::OnCustomersBindingSourceAddingNew);

// Attach an event handler for the ListChanged event.
this->customersBindingSource->ListChanged +=
    gcnew ListChangedEventHandler(this,
        &MainForm::OnCustomersBindingSourceListChanged);
}

private:

void OnMainFormLoad(Object^ sender, EventArgs^ e)
{
    // Add a DemoCustomer to cause a row to be displayed.
    this->customersBindingSource->AddNew();

    // Bind the BindingSource to the DataGridView
    // control's DataSource.
    this->customersDataGridView->DataSource =
        this->customersBindingSource;
}

// This event handler provides custom item-creation behavior.
void OnCustomersBindingSourceAddingNew(Object^ sender,
    AddingEventArgs^ e)
{
    e->NewObject = DemoCustomer::CreateNewCustomer();
}

// This event handler detects changes in the BindingSource
// list or changes to items within the list.
void OnCustomersBindingSourceListChanged(Object^ sender,
    ListChangedEventArgs^ e)
{
    status->Text = Convert::ToString(e->ListChangedType,
        CultureInfo::CurrentCulture);
}
};

}

[STAThread]
int main()
{
    Application::EnableVisualStyles();
    Application::Run(gcnew DataConnectorAddingNewExample::MainForm);
}

```

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;

```

```
using System.Data.SqlClient;
using System.Windows.Forms;

// This form demonstrates using a BindingSource to provide
// data from a collection of custom types to a DataGridView control.
public class Form1 : System.Windows.Forms.Form
{
    // This is the BindingSource that will provide data for
    // the DataGridView control.
    private BindingSource customersBindingSource = new BindingSource();

    // This is the DataGridView control that will display our data.
    private DataGridView customersDataGridView = new DataGridView();

    // Set up the StatusBar for displaying ListChanged events.
    private StatusBar status = new StatusBar();

    public Form1()
    {
        // Set up the form.
        this.Size = new Size(800, 800);
        this.Load += new EventHandler(Form1_Load);
        this.Controls.Add(status);

        // Set up the DataGridView control.
        this.customersDataGridView.Dock = DockStyle.Fill;
        this.Controls.Add(customersDataGridView);

        // Attach an event handler for the AddingNew event.
        this.customersBindingSource.AddingNew +=
            new AddingNewEventHandler(customersBindingSource_AddingNew);

        // Attach an event handler for the ListChanged event.
        this.customersBindingSource.ListChanged +=
            new ListChangedEventHandler(customersBindingSource_ListChanged);
    }

    private void Form1_Load(System.Object sender, System.EventArgs e)
    {
        // Add a DemoCustomer to cause a row to be displayed.
        this.customersBindingSource.AddNew();

        // Bind the BindingSource to the DataGridView
        // control's DataSource.
        this.customersDataGridView.DataSource =
            this.customersBindingSource;
    }

    // This event handler provides custom item-creation behavior.
    void customersBindingSource_AddingNew(
        object sender,
        AddingNewEventArgs e)
    {
        e.NewObject = DemoCustomer.CreateNewCustomer();
    }

    // This event handler detects changes in the BindingSource
    // list or changes to items within the list.
    void customersBindingSource_ListChanged(
        object sender,
        ListChangedEventArgs e)
    {
        status.Text = e.ListChangedType.ToString();
    }

    [STAThread]
    static void Main()
    {
```

```

        Application.EnableVisualStyles();
        Application.Run(new Form1());
    }

}

// This class implements a simple customer type.
public class DemoCustomer
{
    // These fields hold the values for the public properties.
    private Guid idValue = Guid.NewGuid();
    private string customerName = String.Empty;
    private string companyNameValue = String.Empty;
    private string phoneNumberValue = String.Empty;

    // The constructor is private to enforce the factory pattern.
    private DemoCustomer()
    {
        customerName = "no data";
        companyNameValue = "no data";
        phoneNumberValue = "no data";
    }

    // This is the public factory method.
    public static DemoCustomer CreateNewCustomer()
    {
        return new DemoCustomer();
    }

    // This property represents an ID, suitable
    // for use as a primary key in a database.
    public Guid ID
    {
        get
        {
            return this.idValue;
        }
    }

    public string CompanyName
    {
        get
        {
            return this.companyNameValue;
        }

        set
        {
            this.companyNameValue = value;
        }
    }

    public string PhoneNumber
    {
        get
        {
            return this.phoneNumberValue;
        }

        set
        {
            this.phoneNumberValue = value;
        }
    }
}

```

Imports System
Imports System.Collections.Generic

```
Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Data.SqlClient
Imports System.Windows.Forms

' This form demonstrates using a BindingSource to provide
' data from a collection of custom types to a DataGridView control.
Public Class Form1
    Inherits System.Windows.Forms.Form

    ' This is the BindingSource that will provide data for
    ' the DataGridView control.
    Private WithEvents customersBindingSource As New BindingSource()

    ' This is the DataGridView control that will display our data.
    Private customersDataGridView As New DataGridView()

    ' Set up the StatusBar for displaying ListChanged events.
    Private status As New StatusBar()

    Public Sub New()

        ' Set up the form.
        Me.Size = New Size(800, 800)
        AddHandler Me.Load, AddressOf Form1_Load
        Me.Controls.Add(status)

        ' Set up the DataGridView control.
        Me.customersDataGridView.Dock = DockStyle.Fill
        Me.Controls.Add(customersDataGridView)

    End Sub

    Private Sub Form1_Load( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs)

        ' Add a DemoCustomer to cause a row to be displayed.
        Me.customersBindingSource.AddNew()

        ' Bind the BindingSource to the DataGridView
        ' control's DataSource.
        Me.customersDataGridView.DataSource = Me.customersBindingSource

    End Sub

    ' This event handler provides custom item-creation behavior.
    Private Sub customersBindingSource AddingNew( _
        ByVal sender As Object, _
        ByVal e As AddingNewEventArgs) _
        Handles customersBindingSource.AddingNew

        e.NewObject = DemoCustomer.CreateNewCustomer()

    End Sub

    ' This event handler detects changes in the BindingSource
    ' list or changes to items within the list.
    Private Sub customersBindingSource_ListChanged( _
        ByVal sender As Object, _
        ByVal e As ListChangedEventArgs) _
        Handles customersBindingSource.ListChanged

        status.Text = e.ListChangedType.ToString()

    End Sub

    <STAThread()> _
```

```

Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())
End Sub
End Class

' This class implements a simple customer type.
Public Class DemoCustomer

    ' These fields hold the values for the public properties.
    Private idValue As Guid = Guid.NewGuid()
    Private customerName As String = String.Empty
    Private companyNameValue As String = String.Empty
    Private phoneNumberValue As String = String.Empty

    ' The constructor is private to enforce the factory pattern.
    Private Sub New()
        customerName = "no data"
        companyNameValue = "no data"
        phoneNumberValue = "no data"
    End Sub

    ' This is the public factory method.
    Public Shared Function CreateNewCustomer() As DemoCustomer
        Return New DemoCustomer()
    End Function

    ' This property represents an ID, suitable
    ' for use as a primary key in a database.
    Public ReadOnly Property ID() As Guid
        Get
            Return Me.idValue
        End Get
    End Property

    Public Property CompanyName() As String
        Get
            Return Me.companyNameValue
        End Get

        Set(ByVal value As String)
            Me.companyNameValue = Value
        End Set
    End Property

    Public Property PhoneNumber() As String
        Get
            Return Me.phoneNumberValue
        End Get

        Set(ByVal value As String)
            Me.phoneNumberValue = Value
        End Set
    End Property
End Class

```

Compiling the Code

This example requires:

- References to the System, System.Data, System.Drawing and System.Windows.Forms assemblies.

For information about building this example from the command line for visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code](#)

[Example Using Visual Studio.](#)

See Also

[BindingNavigator](#)

[DataGridView](#)

[BindingSource](#)

[BindingSource Component](#)

[How to: Bind a Windows Forms Control to a Type](#)

How to: Handle Errors and Exceptions that Occur with Databinding

5/4/2018 • 7 min to read • [Edit Online](#)

Oftentimes exceptions and errors occur on the underlying business objects when you bind them to controls. You can intercept these errors and exceptions and then either recover or pass the error information to the user by handling the [BindingComplete](#) event for a particular [Binding](#), [BindingSource](#), or [CurrencyManager](#) component.

Example

This code example demonstrates how to handle errors and exceptions that occur during a data-binding operation. It demonstrates how to intercept errors by handling the [Binding.BindingComplete](#) event of the [Binding](#) objects. In order to intercept errors and exceptions by handling this event, you must enable formatting for the binding. You can enable formatting when the binding is constructed or added to the binding collection, or by setting the [FormattingEnabled](#) property to `true`.

```
// Represents a business object that throws exceptions when
// invalid values are entered for some of its properties.
public ref class Part
{
private:
    String^ name;
    int number;
    double price;

public:
    Part(String^ name, int number, double price)
    {
        PartName = name;
        PartNumber = number;
        PartPrice = price;
    }

    property String^ PartName
    {
        String^ get()
        {
            return name;
        }

        void set(String^ value)
        {
            if (value->Length <= 0)
            {
                throw gcnew Exception(
                    "Each part must have a name.");
            }
            else
            {
                name = value;
            }
        }
    }

    property double PartPrice
    {
        double get()
        {
```

```

        return price;
    }

    void set(double value)
    {
        price = value;
    }
}

property int PartNumber
{
    int get()
    {
        return number;
    }

    void set(int value)
    {
        if (value < 100)
        {
            throw gcnew Exception(
                "Invalid part number." \
                "Part numbers must be " \
                "greater than 100.");
        }
        else
        {
            number = value;
        }
    }
}

};

ref class MainForm: public Form
{
private:
    BindingSource^ bindingSource;
    TextBox^ partNameTextBox;
    TextBox^ partNumberTextBox;
    TextBox^ partPriceTextBox;

public:
    MainForm()
    {
        bindingSource = gcnew BindingSource;
        partNameTextBox = gcnew TextBox;
        partNumberTextBox = gcnew TextBox;
        partPriceTextBox = gcnew TextBox;

        //Set up the textbox controls.
        this->partNameTextBox->Location = Point(82, 13);
        this->partNameTextBox->TabIndex = 1;
        this->partNumberTextBox->Location = Point(81, 47);
        this->partNumberTextBox->TabIndex = 2;
        this->partPriceTextBox->Location = Point(81, 83);
        this->partPriceTextBox->TabIndex = 3;

        // Add the textbox controls to the form
        this->Controls->Add(this->partNumberTextBox);
        this->Controls->Add(this->partNameTextBox);
        this->Controls->Add(this->partPriceTextBox);

        // Handle the form's Load event.
        this->Load += gcnew EventHandler(this,
            &MainForm::OnMainFormLoad);
    }
}

private:

```

```

void OnMainFormLoad(Object^ sender, EventArgs^ e)
{
    // Set the DataSource of bindingSource to the Part type.
    bindingSource->DataSource = Part::typeid;

    // Bind the textboxes to the properties of the Part type,
    // enabling formatting.
    partNameTextBox->DataBindings->Add(
        "Text", bindingSource, "PartName", true);
    partNumberTextBox->DataBindings->Add(
        "Text", bindingSource, "PartNumber", true);

    //Bind the textbox to the PartPrice value
    // with currency formatting.
    partPriceTextBox->DataBindings->Add("Text", bindingSource, "PartPrice", true,
        DataSourceUpdateMode::OnPropertyChanged, nullptr, "C");

    // Handle the BindingComplete event for bindingSource and
    // the partNameBinding.
    bindingSource->BindingComplete +=
        gcnew BindingCompleteEventHandler(this,
            &MainForm::OnBindingSourceBindingComplete);
    bindingSource->BindingComplete +=
        gcnew BindingCompleteEventHandler(this,
            &MainForm::OnPartNameBindingBindingComplete);

    // Add a new part to bindingSource.
    bindingSource->Add(gcnew Part("Widget", 1234, 12.45));
}

// Handle the BindingComplete event to catch errors and
// exceptions in binding process.
void OnBindingSourceBindingComplete(Object^ sender,
    BindingCompleteEventArgs^ e)
{
    if (e->BindingCompleteState ==
        BindingCompleteState::Exception)
    {
        MessageBox::Show(String::Format(
            CultureInfo::CurrentCulture,
            "bindingSource: {0}", e->Exception->Message));
    }

    if (e->BindingCompleteState ==
        BindingCompleteState::DataError)
    {
        MessageBox::Show(String::Format(
            CultureInfo::CurrentCulture,
            "bindingSource: {0}", e->Exception->Message));
    }
}

// Handle the BindingComplete event to catch errors and
// exceptions in binding process.
void OnPartNameBindingBindingComplete(Object^ sender,
    BindingCompleteEventArgs^ e)
{
    if (e->BindingCompleteState ==
        BindingCompleteState::Exception)
    {
        MessageBox::Show(String::Format(
            CultureInfo::CurrentCulture,
            "PartNameBinding: {0}", e->Exception->Message));
    }

    if (e->BindingCompleteState ==
        BindingCompleteState::DataError)
}

```

```

        MessageBox::Show(String::Format(
            CultureInfo::CurrentCulture,
            "PartNameBinding: {0}", e->Exception->Message));
    }
}
};


```

```

using System;
using System.Drawing;
using System.Windows.Forms;

class Form1 : Form
{
    private BindingSource BindingSource1 = new BindingSource();
    private TextBox textBox1 = new TextBox();
    private TextBox textBox2 = new TextBox();
    private TextBox textBox3 = new TextBox();

    public Form1()
    {
        //Set up the textbox controls.
        this.textBox1.Location = new System.Drawing.Point(82, 13);
        this.textBox1.TabIndex = 1;
        this.textBox2.Location = new System.Drawing.Point(81, 47);
        this.textBox2.TabIndex = 2;
        this.textBox3.Location = new System.Drawing.Point(81, 83);
        this.textBox3.TabIndex = 3;

        // Add the textbox controls to the form
        this.Controls.Add(this.textBox2);
        this.Controls.Add(this.textBox1);
        this.Controls.Add(this.textBox3);

        // Handle the form's Load event.
        this.Load += new System.EventHandler(this.Form1_Load);
    }
    Binding partNameBinding;
    Binding partNumberBinding;

    private void Form1_Load(object sender, EventArgs e)
    {
        // Set the DataSource of BindingSource1 to the Part type.
        BindingSource1.DataSource = typeof(Part);

        // Bind the textboxes to the properties of the Part type,
        // enabling formatting.
        partNameBinding = textBox1.DataBindings.Add("Text",
            BindingSource1, "PartName", true);

        partNumberBinding = textBox2.DataBindings.Add("Text", BindingSource1, "PartNumber",
            true);

        //Bind the textbox to the PartPrice value with currency formatting.
        textBox3.DataBindings.Add("Text", BindingSource1, "PartPrice", true,
            DataSourceUpdateMode.OnPropertyChanged, 0, "C");

        // Handle the BindingComplete event for BindingSource1 and
        // the partNameBinding.
        partNumberBinding.BindingComplete +=
            new BindingCompleteEventHandler(partNumberBinding_BindingComplete);
        partNameBinding.BindingComplete +=
            new BindingCompleteEventHandler(partNameBinding_BindingComplete);

        // Add a new part to BindingSource1.
        BindingSource1.Add(new Part("Widget", 1234, 12.45));
    }
}


```

```

        BindingSource1.EndEdit();
    }

    // Handle the BindingComplete event to catch errors and exceptions
    // in binding process.
    void partNumberBinding_BindingComplete(object sender,
        BindingCompleteEventArgs e)
    {
        if (e.BindingCompleteState != BindingCompleteState.Success)
            MessageBox.Show("partNumberBinding: " + e.ErrorText);
    }

    // Handle the BindingComplete event to catch errors and
    // exceptions in binding process.
    void partNameBinding_BindingComplete(object sender,
        BindingCompleteEventArgs e)
    {
        if (e.BindingCompleteState != BindingCompleteState.Success)
            MessageBox.Show("partNameBinding: " + e.ErrorText);
    }

    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.Run(new Form1());
    }
}

// Represents a business object that throws exceptions when invalid values are
// entered for some of its properties.
public class Part
{
    private string name;
    private int number;
    private double price;

    public Part(string name, int number, double price)
    {
        PartName = name;
        PartNumber = number;
        PartPrice = price;
    }

    public string PartName
    {
        get { return name; }
        set
        {
            if (value.Length <= 0)
                throw new Exception("Each part must have a name.");
            else
                name = value;
        }
    }

    public double PartPrice
    {
        get { return price; }
        set { price = value; }
    }

    public int PartNumber
    {
        get { return number; }
        set
        {
            if (value < 100)
                throw new Exception("Invalid part number." +
                    " Part numbers must be greater than 100.");
            else

```

```
        else
            number = value;
    }
}
}
```

```
Imports System
Imports System.Drawing
Imports System.Windows.Forms

Class Form1
    Inherits Form

    Private BindingSource1 As New BindingSource()
    Private textBox1 As New TextBox()
    Private textBox2 As New TextBox()
    Private textBox3 As New TextBox()

    Public Sub New()

        'Set up the textbox controls.
        Me.textBox1.Location = New System.Drawing.Point(82, 13)
        Me.textBox1.TabIndex = 1
        Me.textBox2.Location = New System.Drawing.Point(81, 47)
        Me.textBox2.TabIndex = 2
        Me.textBox3.Location = New System.Drawing.Point(81, 83)
        Me.textBox3.TabIndex = 3

        ' Add the textbox controls to the form
        Me.Controls.Add(Me.textBox2)
        Me.Controls.Add(Me.textBox1)
        Me.Controls.Add(Me.textBox3)

    End Sub

    Private WithEvents partNameBinding As Binding
    Private WithEvents partNumberBinding As Binding

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs) _
        Handles Me.Load

        ' Set the DataSource of BindingSource1 to the Part type.
        BindingSource1.DataSource = GetType(Part)

        ' Bind the textboxes to the properties of the Part type,
        ' enabling formatting.
        partNameBinding = textBox1.DataBindings.Add("Text", BindingSource1, _
            "PartName", True)
        partNumberBinding = textBox2.DataBindings.Add("Text", BindingSource1, _
            "PartNumber", True)

        'Bind the textbox to the PartPrice value with currency formatting.
        textBox3.DataBindings.Add("Text", BindingSource1, "PartPrice", _
            True, DataSourceUpdateMode.OnPropertyChanged, 0, "C")

        ' Add a new part to BindingSource1.
        BindingSource1.Add(New Part("Widget", 1234, 12.45))
    End Sub

    ' Handle the BindingComplete event to catch errors and exceptions
    ' in binding process.
    Sub partNumberBinding_BindingComplete(ByVal sender As Object, _
        ByVal e As BindingCompleteEventArgs) _
        Handles partNumberBinding.BindingComplete

        If Not e.BindingCompleteState = BindingCompleteState.Success Then
            MessageBox.Show("partNumberBinding: " + e.ErrorText)
        End If
    End Sub

```

```

    End If

End Sub

' Handle the BindingComplete event to catch errors and exceptions
' in binding process.
Sub partNameBinding_BindingComplete(ByVal sender As Object, _
    ByVal e As BindingCompleteEventArgs) _
Handles partNameBinding.BindingComplete

    If Not e.BindingCompleteState = BindingCompleteState.Success Then
        MessageBox.Show("partNameBinding: " + e.ErrorText)
    End If

End Sub

<STAThread()> _
Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())
End Sub

End Class

' Represents a business object that throws exceptions when invalid
' values are entered for some of its properties.
Public Class Part
    Private name As String
    Private number As Integer
    Private price As Double

    Public Sub New(ByVal name As String, ByVal number As Integer, _
        ByVal price As Double)

        PartName = name
        PartNumber = number
        PartPrice = price
    End Sub

    Public Property PartName() As String
        Get
            Return name
        End Get
        Set(ByVal value As String)
            If Value.Length <= 0 Then
                Throw New Exception("Each part must have a name.")
            Else
                name = Value
            End If
        End Set
    End Property

    Public Property PartPrice() As Double
        Get
            Return price
        End Get
        Set(ByVal value As Double)
            price = Value
        End Set
    End Property

    Public Property PartNumber() As Integer
        Get
            Return number
        End Get
        Set(ByVal value As Integer)
            If Value < 100 Then
                Throw New Exception("Invalid part number." _
                    & " Part numbers must be greater than 100.")
            End If
        End Set
    End Property

```

```
& " Part numbers must be greater than 100." )  
Else  
    number = Value  
End If  
End Set  
End Property  
End Class
```

When the code is running and an empty string is entered for the part name or a value less than 100 is entered for the part number, a message box appears. This is a result of handling the [Binding.BindingComplete](#) event for these textbox bindings.

Compiling the Code

This example requires:

- References to the System, System.Drawing and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[Binding.BindingComplete](#)
[BindingSource.BindingComplete](#)
[BindingSource Component](#)

How to: Raise Change Notifications Using a BindingSource and the INotifyPropertyChanged Interface

5/4/2018 • 6 min to read • [Edit Online](#)

The [BindingSource](#) component will automatically detect changes in a data source when the type contained in the data source implements the [INotifyPropertyChanged](#) interface and raises [PropertyChanged](#) events when a property value is changed. This is useful because controls bound to the [BindingSource](#) will then automatically update as the data source values change.

NOTE

If your data source implements [INotifyPropertyChanged](#) and you are performing asynchronous operations, you should not make changes to the data source on a background thread. Instead, you should read the data on a background thread and merge the data into a list on the UI thread.

Example

The following code example demonstrates a simple implementation of the [INotifyPropertyChanged](#) interface. It also shows how the [BindingSource](#) automatically passes a data source change to a bound control when the [BindingSource](#) is bound to a list of the [INotifyPropertyChanged](#) type.

If you use the `CallerMemberName` attribute, calls to the `NotifyPropertyChanged` method don't have to specify the property name as a string argument. For more information, see [Caller Information](#).

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Runtime.CompilerServices;
using System.Windows.Forms;

// Change the namespace to the project name.
namespace TestNotifyPropertyChangedCS
{
    // This form demonstrates using a BindingSource to bind
    // a list to a DataGridView control. The list does not
    // raise change notifications. However the DemoCustomer type
    // in the list does.
    public partial class Form1 : Form
    {
        // This button causes the value of a list element to be changed.
        private Button changeItemBtn = new Button();

        // This DataGridView control displays the contents of the list.
        private DataGridView customersDataGridView = new DataGridView();

        // This BindingSource binds the list to the DataGridView control.
        private BindingSource customersBindingSource = new BindingSource();

        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

```

        // Set up the "Change Item" button.
        this.changeItemBtn.Text = "Change Item";
        this.changeItemBtn.Dock = DockStyle.Bottom;
        this.changeItemBtn.Click +=
            new EventHandler(changeItemBtn_Click);
        this.Controls.Add(this.changeItemBtn);

        // Set up the DataGridView.
        customersDataGridView.Dock = DockStyle.Top;
        this.Controls.Add(customersDataGridView);

        this.Size = new Size(400, 200);
    }

private void Form1_Load(object sender, EventArgs e)
{
    // Create and populate the list of DemoCustomer objects
    // which will supply data to the DataGridView.
    BindingList<DemoCustomer> customerList = new BindingList<DemoCustomer>();
    customerList.Add(DemoCustomer.CreateNewCustomer());
    customerList.Add(DemoCustomer.CreateNewCustomer());
    customerList.Add(DemoCustomer.CreateNewCustomer());

    // Bind the list to the BindingSource.
    this.customersBindingSource.DataSource = customerList;

    // Attach the BindingSource to the DataGridView.
    this.customersDataGridView.DataSource =
        this.customersBindingSource;
}

// Change the value of the CompanyName property for the first
// item in the list when the "Change Item" button is clicked.
void changeItemBtn_Click(object sender, EventArgs e)
{
    // Get a reference to the list from the BindingSource.
    BindingList<DemoCustomer> customerList =
        this.customersBindingSource.DataSource as BindingList<DemoCustomer>;

    // Change the value of the CompanyName property for the
    // first item in the list.
    customerList[0].CustomerName = "Tailspin Toys";
    customerList[0].PhoneNumber = "(708)555-0150";
}

}

// This is a simple customer class that
// implements the IPropertyChange interface.
public class DemoCustomer : INotifyPropertyChanged
{
    // These fields hold the values for the public properties.
    private Guid idValue = Guid.NewGuid();
    private string customerNameValue = String.Empty;
    private string phoneNumberValue = String.Empty;

    public event PropertyChangedEventHandler PropertyChanged;

    // This method is called by the Set accessor of each property.
    // The CallerMemberName attribute that is applied to the optional propertyName
    // parameter causes the property name of the caller to be substituted as an argument.
    private void NotifyPropertyChanged([CallerMemberName] String propertyName = "")
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}

```

```

// The constructor is private to enforce the factory pattern.
private DemoCustomer()
{
    customerNameValue = "Customer";
    phoneNumberValue = "(312)555-0100";
}

// This is the public factory method.
public static DemoCustomer CreateNewCustomer()
{
    return new DemoCustomer();
}

// This property represents an ID, suitable
// for use as a primary key in a database.
public Guid ID
{
    get
    {
        return this.idValue;
    }
}

public string CustomerName
{
    get
    {
        return this.customerNameValue;
    }

    set
    {
        if (value != this.customerNameValue)
        {
            this.customerNameValue = value;
            NotifyPropertyChanged();
        }
    }
}

public string PhoneNumber
{
    get
    {
        return this.phoneNumberValue;
    }

    set
    {
        if (value != this.phoneNumberValue)
        {
            this.phoneNumberValue = value;
            NotifyPropertyChanged();
        }
    }
}
}

```

```

Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Drawing
Imports System.Runtime.CompilerServices
Imports System.Windows.Forms

```

```

' This form demonstrates using a BindingSource to bind
' a list to a DataGridView control. The list does not
' raise change notifications. However the DemoCustomer type
' in the list does.

Public Class Form1
    Inherits System.Windows.Forms.Form
    ' This button causes the value of a list element to be changed.
    Private changeItemBtn As New Button()

    ' This DataGridView control displays the contents of the list.
    Private customersDataGridView As New DataGridView()

    ' This BindingSource binds the list to the DataGridView control.
    Private customersBindingSource As New BindingSource()

    Public Sub New()
        InitializeComponent()

        ' Set up the "Change Item" button.
        Me.changeItemBtn.Text = "Change Item"
        Me.changeItemBtn.Dock = DockStyle.Bottom
        AddHandler Me.changeItemBtn.Click, AddressOf changeItemBtn_Click
        Me.Controls.Add(Me.changeItemBtn)

        ' Set up the DataGridView.
        customersDataGridView.Dock = DockStyle.Top
        Me.Controls.Add(customersDataGridView)

        Me.Size = New Size(400, 200)
    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles Me.Load

        ' Create and populate the list of DemoCustomer objects
        ' which will supply data to the DataGridView.
        Dim customerList As New BindingList(Of DemoCustomer)

        customerList.Add(DemoCustomer.CreateNewCustomer())
        customerList.Add(DemoCustomer.CreateNewCustomer())
        customerList.Add(DemoCustomer.CreateNewCustomer())

        ' Bind the list to the BindingSource.
        Me.customersBindingSource.DataSource = customerList

        ' Attach the BindingSource to the DataGridView.
        Me.customersDataGridView.DataSource = Me.customersBindingSource
    End Sub

    ' This event handler changes the value of the CompanyName
    ' property for the first item in the list.
    Private Sub changeItemBtn_Click(ByVal sender As Object, ByVal e As EventArgs)
        ' Get a reference to the list from the BindingSource.
        Dim customerList As BindingList(Of DemoCustomer) = _
            CType(customersBindingSource.DataSource, BindingList(Of DemoCustomer))

        ' Change the value of the CompanyName property for the
        ' first item in the list.
        customerList(0).CompanyName = "Tailspin Toys"
        customerList(0).PhoneNumber = "(708)555-0150"
    End Sub
End Class

' This class implements a simple customer type
' that implements the IPropertyChanged interface.
Public Class DemoCustomer
    Implements INotifyPropertyChanged

```

```

' These fields hold the values for the public properties.
Private idValue As Guid = Guid.NewGuid()
Private customerNameValue As String = String.Empty
Private phoneNumberValue As String = String.Empty

Public Event PropertyChanged As PropertyChangedEventHandler _
    Implements INotifyPropertyChanged.PropertyChanged

' This method is called by the Set accessor of each property.
' The CallerMemberName attribute that is applied to the optional propertyName
' parameter causes the property name of the caller to be substituted as an argument.
Private Sub NotifyPropertyChanged(<CallerMemberName()> Optional ByVal propertyName As String = Nothing)
    RaiseEvent PropertyChanged(Me, New PropertyChangedEventArgs(propertyName))
End Sub

' The constructor is private to enforce the factory pattern.
Private Sub New()
    customerNameValue = "Customer"
    phoneNumberValue = "(312)555-0100"
End Sub

' This is the public factory method.
Public Shared Function CreateNewCustomer() As DemoCustomer
    Return New DemoCustomer()
End Function

' This property represents an ID, suitable
' for use as a primary key in a database.
Public ReadOnly Property ID() As Guid
    Get
        Return Me.idValue
    End Get
End Property

Public Property CustomerName() As String
    Get
        Return Me.customerNameValue
    End Get

    Set(ByVal value As String)
        If Not (value = customerNameValue) Then
            Me.customerNameValue = value
            NotifyPropertyChanged()
        End If
    End Set
End Property

Public Property PhoneNumber() As String
    Get
        Return Me.phoneNumberValue
    End Get

    Set(ByVal value As String)
        If Not (value = phoneNumberValue) Then
            Me.phoneNumberValue = value
            NotifyPropertyChanged()
        End If
    End Set
End Property
End Class

```

Compiling the Code

This example requires:

- References to the System, System.Data, System.Drawing and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [HYPERLINK "http://msdn.microsoft.com/library/Bb129228\(v=vs.110\)" How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio.](#)

See Also

[INotifyPropertyChanged](#)

[BindingSource Component](#)

[How to: Raise Change Notifications Using the BindingSource ResetItem Method](#)

How to: Raise Change Notifications Using the BindingSource ResetItem Method

5/4/2018 • 7 min to read • [Edit Online](#)

Some data sources for your controls do not raise change notifications when items are changed, added, or deleted. With the [BindingSource](#) component, you can bind to such data sources and raise a change notification from your code.

Example

This form demonstrates using a [BindingSource](#) component to bind a list to a [DataGridView](#) control. The list does not raise change notifications, so the [ResetItem](#) method on the [BindingSource](#) is called when an item in the list is changed. .

```
#using <System.dll>
#using <System.Data.dll>
#using <System.Drawing.dll>
#using <System.EnterpriseServices.dll>
#using <System.Transactions.dll>
#using <System.Windows.Forms.dll>
#using <System.Xml.dll>

using namespace System;
using namespace System::Collections::Generic;
using namespace System::ComponentModel;
using namespace System::Data;
using namespace System::Data::Common;
using namespace System::Data::SqlClient;
using namespace System::Diagnostics;
using namespace System::Drawing;
using namespace System::Windows::Forms;

// This class implements a simple customer type.
public ref class DemoCustomer
{
private:
    // These fields hold the values for the public properties.
    Guid idValue;
    String^ customerName;
    String^ companyNameValue;
    String^ phoneNumberValue;

    // The constructor is private to enforce the factory pattern.
    DemoCustomer()
    {
        idValue = Guid::.NewGuid();
        customerName = L"no data";
        companyNameValue = L"no data";
        phoneNumberValue = L"no data";
    }

public:
    // This is the public factory method.
    static DemoCustomer^ CreateNewCustomer()
    {
        return gcnew DemoCustomer();
    }

    property Guid ID
```

```

property Guid ID
{
    // This property represents an ID, suitable
    // for use as a primary key in a database.
    Guid get()
    {
        return this->idValue;
    }
}

property String^ CompanyName
{
    String^ get()
    {
        return this->companyNameValue;
    }

    void set( String^ value )
    {
        this->companyNameValue = value;
    }
}

property String^ PhoneNumber
{
    String^ get()
    {
        return this->phoneNumberValue;
    }

    void set( String^ value )
    {
        this->phoneNumberValue = value;
    }
}
};

// This form demonstrates using a BindingSource to bind
// a list to a DataGridView control. The list does not
// raise change notifications, so the ResetItem method
// on the BindingSource is used.
public ref class Form1: public System::Windows::Forms::Form
{
private:
    // This button causes the value of a list element to be changed.
    Button^ changeItemBtn;

    // This is the DataGridView control that displays the contents
    // of the list.
    DataGridView^ customersDataGridView;

    // This is the BindingSource used to bind the list to the
    // DataGridView control.
    BindingSource^ customersBindingSource;

public:
    Form1()
    {
        changeItemBtn = gcnew Button;
        customersDataGridView = gcnew DataGridView;
        customersBindingSource = gcnew BindingSource;

        // Set up the "Change Item" button.
        this->changeItemBtn->Text = L"Change Item";
        this->changeItemBtn->Dock = DockStyle::Bottom;
        this->changeItemBtn->Click += gcnew EventHandler(
            this, &Form1::changeItemBtn_Click );
        this->Controls->Add( this->changeItemBtn );
    }
}

```

```

// Set up the DataGridView.
customersDataGridView->Dock = DockStyle::Top;
this->Controls->Add( customersDataGridView );
this->Size = System::Drawing::Size( 800, 200 );
this->Load += gcnew EventHandler( this, &Form1::Form1_Load );
}

private:
void Form1_Load( Object^ /*sender*/, EventArgs^ /*e*/ )
{
    // Create and populate the list of DemoCustomer objects
    // which will supply data to the DataGridView.
    List< DemoCustomer^ >^ customerList = gcnew List< DemoCustomer^ >;
    customerList->Add( DemoCustomer::CreateNewCustomer() );
    customerList->Add( DemoCustomer::CreateNewCustomer() );
    customerList->Add( DemoCustomer::CreateNewCustomer() );

    // Bind the list to the BindingSource.
    this->customersBindingSource->DataSource = customerList;

    // Attach the BindingSource to the DataGridView.
    this->customersDataGridView->DataSource =
        this->customersBindingSource;
}

// This event handler changes the value of the CompanyName
// property for the first item in the list.
void changeItemBtn_Click( Object^ /*sender*/, EventArgs^ /*e*/ )
{
    // Get a reference to the list from the BindingSource.
    List< DemoCustomer^ >^ customerList =
        static_cast<List< DemoCustomer^ >>(
            this->customersBindingSource->DataSource);

    // Change the value of the CompanyName property for the
    // first item in the list.
    customerList->default[ 0 ]->CompanyName = L"Tailspin Toys";

    // Call ResetItem to alert the BindingSource that the
    // list has changed.
    this->customersBindingSource->ResetItem( 0 );
}
};

int main()
{
    Application::EnableVisualStyles();
    Application::Run( gcnew Form1 );
}

```

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.Common;
using System.Diagnostics;
using System.Drawing;
using System.Data.SqlClient;
using System.Windows.Forms;

// This form demonstrates using a BindingSource to bind
// a list to a DataGridView control. The list does not
// raise change notifications, so the ResetItem method
// on the BindingSource is used.
public class Form1 : System.Windows.Forms.Form
{
    // This button causes the value of a list element to be changed.
    private Button changeItemBtn = new Button();

```

```
private Button changeItemBtn = new Button();  
  
// This is the DataGridView control that displays the contents  
// of the list.  
private DataGridView customersDataGridView = new DataGridView();  
  
// This is the BindingSource used to bind the list to the  
// DataGridView control.  
private BindingSource customersBindingSource = new BindingSource();  
  
public Form1()  
{  
    // Set up the "Change Item" button.  
    this.changeItemBtn.Text = "Change Item";  
    this.changeItemBtn.Dock = DockStyle.Bottom;  
    this.changeItemBtn.Click +=  
        new EventHandler(changeItemBtn_Click);  
    this.Controls.Add(this.changeItemBtn);  
  
    // Set up the DataGridView.  
    customersDataGridView.Dock = DockStyle.Top;  
    this.Controls.Add(customersDataGridView);  
    this.Size = new Size(800, 200);  
    this.Load += new EventHandler(Form1_Load);  
}  
  
private void Form1_Load(System.Object sender, System.EventArgs e)  
{  
    // Create and populate the list of DemoCustomer objects  
    // which will supply data to the DataGridView.  
    List<DemoCustomer> customerList = new List<DemoCustomer>();  
    customerList.Add(DemoCustomer.CreateNewCustomer());  
    customerList.Add(DemoCustomer.CreateNewCustomer());  
    customerList.Add(DemoCustomer.CreateNewCustomer());  
  
    // Bind the list to the BindingSource.  
    this.customersBindingSource.DataSource = customerList;  
  
    // Attach the BindingSource to the DataGridView.  
    this.customersDataGridView.DataSource =  
        this.customersBindingSource;  
}  
  
// This event handler changes the value of the CompanyName  
// property for the first item in the list.  
void changeItemBtn_Click(object sender, EventArgs e)  
{  
    // Get a reference to the list from the BindingSource.  
    List<DemoCustomer> customerList =  
        this.customersBindingSource.DataSource as List<DemoCustomer>;  
  
    // Change the value of the CompanyName property for the  
    // first item in the list.  
    customerList[0].CompanyName = "Tailspin Toys";  
  
    // Call ResetItem to alert the BindingSource that the  
    // list has changed.  
    this.customersBindingSource.ResetItem(0);  
}  
  
[STAThread]  
static void Main()  
{  
    Application.EnableVisualStyles();  
    Application.Run(new Form1());  
}  
}
```

```

// This class implements a simple customer type.
public class DemoCustomer
{
    // These fields hold the values for the public properties.
    private Guid idValue = Guid.NewGuid();
    private string customerName = String.Empty;
    private string companyNameValue = String.Empty;
    private string phoneNumberValue = String.Empty;

    // The constructor is private to enforce the factory pattern.
    private DemoCustomer()
    {
        customerName = "no data";
        companyNameValue = "no data";
        phoneNumberValue = "no data";
    }

    // This is the public factory method.
    public static DemoCustomer CreateNewCustomer()
    {
        return new DemoCustomer();
    }

    // This property represents an ID, suitable
    // for use as a primary key in a database.
    public Guid ID
    {
        get
        {
            return this.idValue;
        }
    }

    public string CompanyName
    {
        get
        {
            return this.companyNameValue;
        }

        set
        {
            this.companyNameValue = value;
        }
    }

    public string PhoneNumber
    {
        get
        {
            return this.phoneNumberValue;
        }

        set
        {
            this.phoneNumberValue = value;
        }
    }
}

```

```

Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Diagnostics
Imports System.Drawing
Imports System.Windows.Forms

```

```

' This form demonstrates using a BindingSource to bind
' a list to a DataGridView control. The list does not
' raise change notifications, so the ResetItem method
' on the BindingSource is used.
Public Class Form1
    Inherits System.Windows.Forms.Form

    ' This button causes the value of a list element to be changed.
    Private WithEvents changeItemBtn As New Button()

    ' This is the DataGridView control that displays the contents
    ' of the list.
    Private customersDataGridView As New DataGridView()

    ' This is the BindingSource used to bind the list to the
    ' DataGridView control.
    Private WithEvents customersBindingSource As New BindingSource()

    Public Sub New()
        ' Set up the "Change Item" button.
        Me.changeItemBtn.Text = "Change Item"
        Me.changeItemBtn.Dock = DockStyle.Bottom
        Me.Controls.Add(Me.changeItemBtn)

        ' Set up the DataGridView.
        customersDataGridView.Dock = DockStyle.Top
        Me.Controls.Add(customersDataGridView)
        Me.Size = New Size(800, 200)
    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles Me.Load
        ' Create and populate the list of DemoCustomer objects
        ' which will supply data to the DataGridView.
        Dim customerList As List(Of DemoCustomer) = _
            New List(Of DemoCustomer)
        customerList.Add(DemoCustomer.CreateNewCustomer())
        customerList.Add(DemoCustomer.CreateNewCustomer())
        customerList.Add(DemoCustomer.CreateNewCustomer())

        ' Bind the list to the BindingSource.
        Me.customersBindingSource.DataSource = customerList

        ' Attach the BindingSource to the DataGridView.
        Me.customersDataGridView.DataSource = Me.customersBindingSource
    End Sub

    ' This event handler changes the value of the CompanyName
    ' property for the first item in the list.
    Private Sub changeItemBtn_Click(ByVal sender As Object, ByVal e As EventArgs) _
        Handles changeItemBtn.Click

        ' Get a reference to the list from the BindingSource.
        Dim customerList As List(Of DemoCustomer) = _
            CType(Me.customersBindingSource.DataSource, List(Of DemoCustomer))

        ' Change the value of the CompanyName property for the
        ' first item in the list.
        customerList(0).CompanyName = "Tailspin Toys"

        ' Call ResetItem to alert the BindingSource that the
        ' list has changed.
        Me.customersBindingSource.ResetItem(0)
    End Sub

    <STAThread()> _
    Shared Sub Main()

```

```

        Application.EnableVisualStyles()
        Application.Run(New Form1())
    End Sub
End Class

' This class implements a simple customer type.
Public Class DemoCustomer

    ' These fields hold the values for the public properties.
    Private idValue As Guid = Guid.NewGuid()
    Private customerName As String = String.Empty
    Private companyNameValue As String = String.Empty
    Private phoneNumberValue As String = String.Empty

    ' The constructor is private to enforce the factory pattern.
    Private Sub New()
        customerName = "no data"
        companyNameValue = "no data"
        phoneNumberValue = "no data"
    End Sub

    ' This is the public factory method.
    Public Shared Function CreateNewCustomer() As DemoCustomer
        Return New DemoCustomer()
    End Function

    ' This property represents an ID, suitable
    ' for use as a primary key in a database.
    Public ReadOnly Property ID() As Guid
        Get
            Return Me.idValue
        End Get
    End Property

    Public Property CompanyName() As String
        Get
            Return Me.companyNameValue
        End Get

        Set(ByVal value As String)
            Me.companyNameValue = Value
        End Set
    End Property

    Public Property PhoneNumber() As String
        Get
            Return Me.phoneNumberValue
        End Get

        Set(ByVal value As String)
            Me.phoneNumberValue = Value
        End Set
    End Property
End Class

```

Compiling the Code

This example requires:

- References to the System, System.Data, System.Drawing and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code](#)

[Example Using Visual Studio.](#)

See Also

[BindingNavigator](#)

[DataGridView](#)

[BindingSource](#)

[BindingSource Component](#)

[How to: Bind a Windows Forms Control to a Type](#)

How to: Reflect Data Source Updates in a Windows Forms Control with the BindingSource

5/4/2018 • 6 min to read • [Edit Online](#)

When you use data-bound controls, you sometimes have to respond to changes in the data source when the data source does not raise list-changed events. When you use the [BindingSource](#) component to bind your data source to a Windows Forms control, you can notify the control that your data source has changed by calling the [ResetBindings](#) method.

Example

The following code example demonstrates using the [ResetBindings](#) method to notify a bound control about an update in the data source.

```
#using <System.Xml.dll>
#using <System.dll>
#using <System.Data.dll>
#using <System.Drawing.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Collections::Generic;
using namespace System::ComponentModel;
using namespace System::Data;
using namespace System::Drawing;
using namespace System::Text;
using namespace System::Xml;
using namespace System::Windows::Forms;
using namespace System::IO;

namespace System_Windows_Forms_UpdateBinding
{
    public ref class Form1: public Form
    {
    public:
        Form1()
        {
            InitializeComponent();
        }

        [STAThread]
        static void Main()
        {
            Application::EnableVisualStyles();
            Application::Run( gcnew Form1 );
        }

    private:
        void Form1_Load( System::Object^ /*sender*/, System::EventArgs^ /*e*/ )
        {
            // The xml to bind to.
            String^ xml = "<US><states>" +
                "<state><name>Washington</name><capital>Olympia</capital></state>" +
                "<state><name>Oregon</name><capital>Salem</capital></state>" +
                "<state><name>California</name><capital>Sacramento</capital></state>" +
                "<state><name>Nevada</name><capital>Carson City</capital></state>" +
                "</states></US>";

            // Convert the xml string to bytes and load into a memory stream
        }
    }
}
```

```

// Convert the XML string to bytes and load into a memory stream.
array<Byte>^ xmlBytes = Encoding::UTF8->GetBytes( xml );
MemoryStream^ stream = gcnew MemoryStream( xmlBytes, false );

// Create a DataSet and load the XML into it.
dataSet1->ReadXml( stream );

// Set the DataSource to the DataSet, and the DataMember
// to state.
bindingSource1->DataSource = dataSet1;
bindingSource1->DataMember = "state";

dataGridView1->DataSource = bindingSource1;
}

private:
void button1_Click( Object^ /*sender*/, EventArgs^ /*e*/ )
{
    String^ xml = "<US><states>
        + <state><name>Washington</name><capital>Olympia</capital> "
        + "<flower>Coast Rhododendron</flower></state>"
        + "<state><name>Oregon</name><capital>Salem</capital>"
        + "<flower>Oregon Grape</flower></state>"
        + "<state><name>California</name><capital>Sacramento</capital>"
        + "<flower>California Poppy</flower></state>"
        + "<state><name>Nevada</name><capital>Carson City</capital>"
        + "<flower>Sagebrush</flower></state>"
        + "</states></US>";

    // Convert the XML string to bytes and load into a memory stream.
    array<Byte>^ xmlBytes = Encoding::UTF8->GetBytes( xml );
    MemoryStream^ stream = gcnew MemoryStream( xmlBytes, false );

    // Create a DataSet and load the XML into it.
    dataSet2->ReadXml( stream );

    // Set the data source.
    bindingSource1->DataSource = dataSet2;
    bindingSource1->ResetBindings( true );
}

System::Windows::Forms::Button^ button1;
System::Windows::Forms::DataGridView^ dataGridView1;
System::Windows::Forms::BindingSource^ bindingSource1;
System::Data::DataSet^ dataSet1;
DataSet^ dataSet2;

#pragma region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
void InitializeComponent()
{
    this->button1 = gcnew System::Windows::Forms::Button;
    this->dataGridView1 = gcnew System::Windows::Forms::DataGridView;
    this->bindingSource1 = gcnew System::Windows::Forms::BindingSource;
    this->dataSet1 = gcnew System::Data::DataSet;
    this->dataSet2 = gcnew System::Data::DataSet;
    ( (System::ComponentModel::ISupportInitialize^)(this->dataGridView1) )->BeginInit();
    ( (System::ComponentModel::ISupportInitialize^)(this->bindingSource1) )->BeginInit();
    ( (System::ComponentModel::ISupportInitialize^)(this->dataSet1) )->BeginInit();
    ( (System::ComponentModel::ISupportInitialize^)(this->dataSet2) )->BeginInit();
    this->SuspendLayout();

    //
    // button1
    //
}

```

```

this->button1->Location = System::Drawing::Point( 98, 222 );
this->button1->Name = "button1";
this->button1->TabIndex = 0;
this->button1->Text = "button1";
this->button1->Click += gcnew System::EventHandler( this, &Form1::button1_Click );

//
// dataGridView1
//
this->dataGridView1->Dock = System::Windows::Forms::DockStyle::Top;
this->dataGridView1->Location = System::Drawing::Point( 0, 0 );
this->dataGridView1->Name = "dataGridView1";
this->dataGridView1->Size = System::Drawing::Size( 292, 150 );
this->dataGridView1->TabIndex = 1;

//
// dataSet1
//
this->dataSet1->DataSetName = "NewDataSet";
this->dataSet1->Locale = gcnew System::Globalization::CultureInfo( "en-US" );

//
// dataSet2
//
this->dataSet2->DataSetName = "NewDataSet";
this->dataSet2->Locale = gcnew System::Globalization::CultureInfo( "en-US" );

//
// Form1
//
this->ClientSize = System::Drawing::Size( 292, 273 );
this->Controls->Add( this->dataGridView1 );
this->Controls->Add( this->button1 );
this->Name = "Form1";
this->Text = "Form1";
this->Load += gcnew EventHandler( this, &Form1::Form1_Load );
( (System::ComponentModel::ISupportInitialize^)(this->dataGridView1) )->EndInit();
( (System::ComponentModel::ISupportInitialize^)(this->bindingSource1) )->EndInit();
( (System::ComponentModel::ISupportInitialize^)(this->dataSet1) )->EndInit();
( (System::ComponentModel::ISupportInitialize^)(this->dataSet2) )->EndInit();
this->ResumeLayout( false );
}

#pragma endregion
};

}

int main()
{
    System_Windows_Forms_UpdateBinding::Form1::Main();
}

```

```

using System;
using System.ComponentModel;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Collections;

namespace System_Windows_Forms_UpdateBinding
{
    class Form1 : Form
    {
        // Declare the objects on the form.
        private Label label1;
        private Label label2;
        private TextBox textBox1;
        private TextBox textBox2;

```

```

private TextBox textBox2;
private Button button1;
private BindingSource bindingSource1;
ArrayList states;

public Form1()
{
    // Basic form setup.
    this.button1 = new System.Windows.Forms.Button();
    this.textBox1 = new System.Windows.Forms.TextBox();
    this.label1 = new System.Windows.Forms.Label();
    this.label2 = new System.Windows.Forms.Label();
    this.textBox2 = new System.Windows.Forms.TextBox();
    this.button1.Location = new System.Drawing.Point(12, 18);
    this.button1.Size = new System.Drawing.Size(119, 38);
    this.button1.Text = "RemoveAt(0)";
    this.button1.Click += new System.EventHandler(this.button1_Click);
    this.textBox1.Location = new System.Drawing.Point(55, 75);
    this.textBox1.ReadOnly = true;
    this.textBox1.Size = new System.Drawing.Size(119, 20);
    this.label1.Location = new System.Drawing.Point(12, 110);
    this.label1.Size = new System.Drawing.Size(43, 14);
    this.label1.Text = "Capital:";
    this.label2.Location = new System.Drawing.Point(12, 78);
    this.label2.Size = new System.Drawing.Size(34, 14);
    this.label2.Text = "State:";
    this.textBox2.Location = new System.Drawing.Point(55, 110);
    this.textBox2.Size = new System.Drawing.Size(119, 20);
    this.textBox2.ReadOnly = true;
    this.ClientSize = new System.Drawing.Size(292, 273);
    this.Controls.Add(this.textBox2);
    this.Controls.Add(this.label2);
    this.Controls.Add(this.label1);
    this.Controls.Add(this.textBox1);
    this.Controls.Add(this.button1);
    this.Text = "Form1";

    // Create an ArrayList containing some of the State objects.
    states = new ArrayList();
    states.Add(new State("California", "Sacramento"));
    states.Add(new State("Oregon", "Salem"));
    states.Add(new State("Washington", "Olympia"));
    states.Add(new State("Idaho", "Boise"));
    states.Add(new State("Utah", "Salt Lake City"));
    states.Add(new State("Hawaii", "Honolulu"));
    states.Add(new State("Colorado", "Denver"));
    states.Add(new State("Montana", "Helena"));

    bindingSource1 = new BindingSource();

    // Bind BindingSource1 to the list of states.
    bindingSource1.DataSource = states;

    // Bind the two text boxes to properties of State.
    textBox1.DataBindings.Add("Text", bindingSource1, "Name");
    textBox2.DataBindings.Add("Text", bindingSource1, "Capital");
}

private void button1_Click(object sender, EventArgs e)
{
    // If items remain in the list, remove the first item.
    if (states.Count > 0)
    {
        states.RemoveAt(0);

        // Call ResetBindings to update the textboxes.
        bindingSource1.ResetBindings(false);
    }
}

```

```

[STAThread]
static void Main()
{
    Application.EnableVisualStyles();
    Application.Run(new Form1());
}

// The State class to add to the ArrayList.
private class State
{
    private string stateName;
    public string Name
    {
        get {return stateName;}
    }

    private string stateCapital;
    public string Capital
    {
        get {return stateCapital;}
    }

    public State ( string name, string capital)
    {
        stateName = name;
        stateCapital = capital;
    }
}

}

```

```

Imports System
Imports System.ComponentModel
Imports System.Drawing
Imports System.Text
Imports System.Windows.Forms
Imports System.Collections

Class Form1
    Inherits Form

    ' Declare the objects on the form.
    Private label1 As Label
    Private label2 As Label
    Private textBox1 As TextBox
    Private textBox2 As TextBox
    Private WithEvents button1 As Button
    Private bindingSource1 As BindingSource
    Private states As ArrayList

    Public Sub New()

        ' Basic form setup.
        Me.button1 = New System.Windows.Forms.Button()
        Me.textBox1 = New System.Windows.Forms.TextBox()
        Me.label1 = New System.Windows.Forms.Label()
        Me.label2 = New System.Windows.Forms.Label()
        Me.textBox2 = New System.Windows.Forms.TextBox()
        Me.button1.Location = New System.Drawing.Point(12, 18)
        Me.button1.Size = New System.Drawing.Size(119, 38)
        Me.button1.Text = "RemoveAt(0)"
        Me.textBox1.Location = New System.Drawing.Point(55, 75)
        Me.textBox1.ReadOnly = True
    
```

```

Me.textBox1.Size = New System.Drawing.Size(119, 20)
Me.label1.Location = New System.Drawing.Point(12, 110)
Me.label1.Size = New System.Drawing.Size(43, 14)
Me.label1.Text = "Capital:"
Me.label2.Location = New System.Drawing.Point(12, 78)
Me.label2.Size = New System.Drawing.Size(34, 14)
Me.label2.Text = "State:"
Me.textBox2.Location = New System.Drawing.Point(55, 110)
Me.textBox2.Size = New System.Drawing.Size(119, 20)
Me.textBox2.ReadOnly = True
Me.ClientSize = New System.Drawing.Size(292, 273)
Me.Controls.Add(Me.textBox2)
Me.Controls.Add(Me.label2)
Me.Controls.Add(Me.label1)
Me.Controls.Add(Me.textBox1)
Me.Controls.Add(Me.button1)
Me.Text = "Form1"

' Create an ArrayList containing some of the State objects.
states = New ArrayList()
states.Add(New State("California", "Sacramento"))
states.Add(New State("Oregon", "Salem"))
states.Add(New State("Washington", "Olympia"))
states.Add(New State("Idaho", "Boise"))
states.Add(New State("Utah", "Salt Lake City"))
states.Add(New State("Hawaii", "Honolulu"))
states.Add(New State("Colorado", "Denver"))
states.Add(New State("Montana", "Helena"))

bindingSource1 = New BindingSource()

' Bind BindingSource1 to the list of states.
bindingSource1.DataSource = states

' Bind the two text boxes to properties of State.
textBox1.DataBindings.Add("Text", bindingSource1, "Name")
textBox2.DataBindings.Add("Text", bindingSource1, "Capital")

End Sub

Private Sub button1_Click(ByVal sender As Object, ByVal e As EventArgs) _
Handles button1.Click

' If items remain in the list, remove the first item.
If states.Count > 0 Then
    states.RemoveAt(0)

    ' Call ResetBindings to update the textboxes.
    bindingSource1.ResetBindings(False)
End If

End Sub 'button1_Click

<STAThread()> _
Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())
End Sub

' The State class to add to the ArrayList.
Private Class State
    Private stateName As String

    Public ReadOnly Property Name() As String
        Get
            Return stateName
        End Get
    End Property
End Class

```

```
Private stateCapital As String

Public ReadOnly Property Capital() As String
    Get
        Return stateCapital
    End Get
End Property

Public Sub New(ByVal name As String, ByVal capital As String)
    stateName = name
    stateCapital = capital
End Sub
End Class
End Class
```

Compiling the Code

This example requires:

- References to the System, System.Drawing and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[BindingNavigator](#)

[DataGridView](#)

[BindingSource](#)

[BindingSource Component](#)

[How to: Bind a Windows Forms Control to a Type](#)

How to: Share Bound Data Across Forms Using the BindingSource Component

5/4/2018 • 5 min to read • [Edit Online](#)

You can easily share data across forms using the [BindingSource](#) component. For example, you may want to display one read-only form that summarizes the data-source data and another editable form that contains detailed information about the currently selected item in the data source. This example demonstrates this scenario.

Example

The following code example demonstrates how to share a [BindingSource](#) and its bound data across forms. In this example, the shared [BindingSource](#) is passed to the constructor of the child form. The child form allows the user to edit the data for the currently selected item in the main form.

NOTE

The [BindingComplete](#) event for the [BindingSource](#) component is handled in the example to ensure that the two forms remain synchronized. For more information about why this is done, see [How to: Ensure Multiple Controls Bound to the Same Data Source Remain Synchronized](#).

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Data;

namespace BindingSourceMultipleForms
{
    public class MainForm : Form
    {
        public MainForm()
        {
            this.Load += new EventHandler(MainForm_Load);
        }

        private BindingSource bindingSource1;
        private Button button1;

        private void MainForm_Load(object sender, EventArgs e)
        {
            InitializeData();
        }

        private void InitializeData()
        {
            bindingSource1 = new System.Windows.Forms.BindingSource();

            // Handle the BindingComplete event to ensure the two forms
            // remain synchronized.
            bindingSource1.BindingComplete +=
                new BindingCompleteEventHandler(bindingSource1_BindingComplete);
            ClientSize = new System.Drawing.Size(292, 266);
            DataSet dataset1 = new DataSet();

            // Some xml data to populate the DataSet with.
            string musicXml =
                "<?xml version='1.0' encoding='UTF-8'?>" +
```

```

"<music>" +
    "<recording><artist>Dave Matthews</artist>" +
    "<cd>Under the Table and Dreaming</cd>" +
    "<releaseDate>1994</releaseDate><rating>3.5</rating></recording>" +
    "<recording><artist>Coldplay</artist><cd>X&amp;Y</cd>" +
    "<releaseDate>2005</releaseDate><rating>4</rating></recording>" +
    "<recording><artist>Dave Matthews</artist>" +
    "<cd>Live at Red Rocks</cd>" +
    "<releaseDate>1997</releaseDate><rating>4</rating></recording>" +
    "<recording><artist>U2</artist>" +
    "<cd>Joshua Tree</cd><releaseDate>1987</releaseDate>" +
    "<rating>5</rating></recording>" +
    "<recording><artist>U2</artist>" +
    "<cd>How to Dismantle an Atomic Bomb</cd>" +
    "<releaseDate>2004</releaseDate><rating>4.5</rating></recording>" +
    "<recording><artist>Natalie Merchant</artist>" +
    "<cd>Tigerlily</cd><releaseDate>1995</releaseDate>" +
    "<rating>3.5</rating></recording>" +
    "</music>";

// Read the xml.
System.IO.StringReader reader = new System.IO.StringReader(musicXml);
dataset1.ReadXml(reader);

// Get a DataView of the table contained in the dataset.
DataTableCollection tables = dataset1.Tables;
DataView view1 = new DataView(tables[0]);

// Create a DataGridView control and add it to the form.
DataGridView datagridview1 = new DataGridView();
datagridview1.ReadOnly = true;
datagridview1.AutoGenerateColumns = true;
datagridview1.Width = 300;
this.Controls.Add(datagridview1);
bindingSource1.DataSource = view1;
datagridview1.DataSource = bindingSource1;
datagridview1.Columns.Remove("artist");
datagridview1.Columns.Remove("releaseDate");

// Create and add a button to the form.
button1 = new Button();
button1.AutoSize = true;
button1.Text = "Show/Edit Details";
this.Controls.Add(button1);
button1.Location = new Point(50, 200);
button1.Click += new EventHandler(button1_Click);
}

// Handle the BindingComplete event to ensure the two forms
// remain synchronized.
private void bindingSource1_BindingComplete(object sender, BindingCompleteEventArgs e)
{
    if (e.BindingCompleteContext == BindingCompleteContext.DataSourceUpdate
        && e.Exception == null)
        e.Binding.BindingManagerBase.EndCurrentEdit();
}

// The detailed form will be shown when the button is clicked.
private void button1_Click(object sender, EventArgs e)
{
    DetailForm detailForm = new DetailForm(bindingSource1);
    detailForm.Show();
}

[STAThread]
static void Main()
{
    Application.EnableVisualStyles();
    Application.Run(new MainForm());
}

```

```

        }

    }

    // The detail form class.
    public class DetailForm : Form
    {
        private BindingSource formDataSource;

        // The constructor takes a BindingSource object.
        public DetailForm(BindingSource dataSource)
        {
            formDataSource = dataSource;
            this.ClientSize = new Size(240, 200);
            TextBox textBox1 = new TextBox();
            this.Text = "Selection Details";
            textBox1.Width = 220;
            TextBox textBox2 = new TextBox();
            TextBox textBox3 = new TextBox();
            TextBox textBox4 = new TextBox();
            textBox4.Width = 30;
            textBox3.Width = 50;

            // Associate each text box with a column from the data source.
            textBox1.DataBindings.Add("Text", formDataSource, "cd", true,
DataSourceUpdateMode.OnPropertyChanged);

            textBox2.DataBindings.Add("Text", formDataSource, "artist", true);
            textBox3.DataBindings.Add("Text", formDataSource, "releaseDate", true);
            textBox4.DataBindings.Add("Text", formDataSource, "rating", true);
            textBox1.Location = new Point(10, 10);
            textBox2.Location = new Point(10, 40);
            textBox3.Location = new Point(10, 80);
            textBox4.Location = new Point(10, 120);
            this.Controls.AddRange(new Control[] { textBox1, textBox2, textBox3, textBox4 });
        }

    }
}

```

```

Imports System
Imports System.Drawing
Imports System.Windows.Forms
Imports System.Data

Public Class MainForm
    Inherits Form

    Public Sub New()
        End Sub

    Private WithEvents bindingSource1 As BindingSource
    Private WithEvents button1 As Button

    Private Sub MainForm_Load(ByVal sender As Object, ByVal e As EventArgs) _
        Handles Me.Load

        InitializeData()
    End Sub


    Private Sub InitializeData()
        bindingSource1 = New System.Windows.Forms.BindingSource()
        Dim dataset1 As New DataSet()
        ClientSize = New System.Drawing.Size(292, 266)

        ' Some xml data to populate the DataSet with.
        ' Some xml data to populate the DataSet with.
    End Sub

```

```

Dim musicXml As String = "<?xml version='1.0' encoding='UTF-8'?>" & _
    "<music><recording><artist>Dave Matthews</artist>" & _
    "<cd>Under the Table and Dreaming</cd>" & _
    "<releaseDate>1994</releaseDate><rating>3.5</rating></recording>" & _
    "<recording><artist>Coldplay</artist><cd>X&amp;Y</cd>" & _
    "<releaseDate>2005</releaseDate><rating>4</rating></recording>" & _
    "<recording><artist>Dave Matthews</artist>" & _
    "<cd>Live at Red Rocks</cd>" & _
    "<releaseDate>1997</releaseDate><rating>4</rating></recording>" & _
    "<recording><artist>U2</artist>" & _
    "<cd>Joshua Tree</cd><releaseDate>1987</releaseDate>" & _
    "<rating>5</rating></recording>" & _
    "<recording><artist>U2</artist>" & _
    "<cd>How to Dismantle an Atomic Bomb</cd>" & _
    "<releaseDate>2004</releaseDate><rating>4.5</rating></recording>" & _
    "<recording><artist>Natalie Merchant</artist>" & _
    "<cd>Tigerlily</cd><releaseDate>1995</releaseDate>" & _
    "<rating>3.5</rating></recording>" & _
    "</music>

' Read the xml.
Dim reader As New System.IO.StringReader(musicXml)
dataset1.ReadXml(reader)

' Get a DataView of the table contained in the dataset.
Dim tables As DataTableCollection = dataset1.Tables
Dim view1 As New DataView(tables(0))

' Create a DataGridView control and add it to the form.
Dim datagridview1 As New DataGridView()
datagridview1.ReadOnly = True
datagridview1.AutoGenerateColumns = True
datagridview1.Width = 300
Me.Controls.Add(datagridview1)
bindingSource1.DataSource = view1
datagridview1.DataSource = bindingSource1
datagridview1.Columns.Remove("artist")
datagridview1.Columns.Remove("releaseDate")

' Create and add a button to the form.
button1 = New Button()
button1.AutoSize = True
button1.Text = "Show/Edit Details"
Me.Controls.Add(button1)
button1.Location = New Point(50, 200)

End Sub

' Handle the BindingComplete event to ensure the two forms
' remain synchronized.
Private Sub bindingSource1_BindingComplete(ByVal sender As Object, _
    ByVal e As BindingCompleteEventArgs) Handles bindingSource1.BindingComplete
If e.BindingCompleteContext = BindingCompleteContext.DataSourceUpdate _
    AndAlso e.Exception Is Nothing Then
    e.Binding.BindingManagerBase.EndCurrentEdit()
End If

End Sub

' The detailed form will be shown when the button is clicked.
Private Sub button1_Click(ByVal sender As Object, ByVal e As EventArgs) _
    Handles button1.Click

    Dim detailForm As New DetailForm(bindingSource1)
    detailForm.Show()
End Sub

<STAThread()> "

```

```

Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New MainForm())

End Sub
End Class

' The detail form class.
Public Class DetailForm
    Inherits Form
    Private formDataSource As BindingSource

    ' The constructor takes a BindingSource object.
    Public Sub New(ByVal dataSource As BindingSource)
        formDataSource = dataSource
        Me.ClientSize = New Size(240, 200)
        Dim textBox1 As New TextBox()
        Me.Text = "Selection Details"
        textBox1.Width = 220
        Dim textBox2 As New TextBox()
        Dim textBox3 As New TextBox()
        Dim textBox4 As New TextBox()
        textBox4.Width = 30
        textBox3.Width = 50

        ' Associate each text box with a column from the data source.
        textBox1.DataBindings.Add("Text", formDataSource, "cd", _
            True, DataSourceUpdateMode.OnPropertyChanged)

        textBox2.DataBindings.Add("Text", formDataSource, "artist", True)
        textBox3.DataBindings.Add("Text", formDataSource, "releaseDate", True)
        textBox4.DataBindings.Add("Text", formDataSource, "rating", True)
        textBox1.Location = New Point(10, 10)
        textBox2.Location = New Point(10, 40)
        textBox3.Location = New Point(10, 80)
        textBox4.Location = New Point(10, 120)
        Me.Controls.AddRange(New Control() {textBox1, textBox2, textBox3, _
            textBox4})
    End Sub
End Class

```

Compiling the Code

This example requires:

- References to the System, System.Windows.Forms, System.Drawing, System.Data, and System.Xml assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[BindingSource Component](#)

[Windows Forms Data Binding](#)

[How to: Handle Errors and Exceptions that Occur with Databinding](#)

How to: Sort and Filter ADO.NET Data with the Windows Forms BindingSource Component

5/4/2018 • 2 min to read • [Edit Online](#)

You can expose the sorting and filtering capability of [BindingSource](#) control through the [Sort](#) and [Filter](#) properties. You can apply simple sorting when the underlying data source is an [IBindingList](#), and you can apply filtering and advanced sorting when the data source is an [IBindingListView](#). The [Sort](#) property requires standard ADO.NET syntax: a string representing the name of a column of data in the data source followed by [ASC](#) or [DESC](#) to indicate whether the list should be sorted in ascending or descending order. You can set advanced sorting or multiple-column sorting by separating each column with a comma separator. The [Filter](#) property takes a string expression.

NOTE

Storing sensitive information, such as a password, within the connection string can affect the security of your application. Using Windows Authentication (also known as integrated security) is a more secure way to control access to a database. For more information, see [Protecting Connection Information](#).

To filter data with the BindingSource

- Set the [Filter](#) property to expression that you want.

In the following code example, the expression is a column name followed by value that you want for the column.

```
BindingSource1.Filter = "ContactTitle='Owner'"
```

```
BindingSource1.Filter = "ContactTitle='Owner'"
```

To sort data with the BindingSource

- Set the [Sort](#) property to the column name that you want followed by [ASC](#) or [DESC](#) to indicate the ascending or descending order.
- Separate multiple columns with a comma.

```
BindingSource1.Sort = "Country DESC, Address ASC";
```

```
BindingSource1.Sort = "Country DESC, Address ASC"
```

Example

The following code example loads data from the Customers table of the Northwind sample database into a [DataGridView](#) control, and filters and sorts the displayed data.

```

private void InitializeSortedFilteredBindingSource()
{
    // Create the connection string, data adapter and data table.
    SqlConnection connectionString =
        new SqlConnection("Initial Catalog=Northwind;" +
            "Data Source=localhost;Integrated Security=SSPI;");
    SqlDataAdapter customersTableAdapter =
        new SqlDataAdapter("Select * from Customers", connectionString);
    DataTable customerTable = new DataTable();

    // Fill the adapter with the contents of the customer table.
    customersTableAdapter.Fill(customerTable);

    // Set data source for BindingSource1.
    BindingSource1.DataSource = customerTable;

    // Filter the items to show contacts who are owners.
    BindingSource1.Filter = "ContactTitle='Owner'";

    // Sort the items on the company name in descending order.
    BindingSource1.Sort = "Country DESC, Address ASC";

    // Set the data source for dataGridView1 to BindingSource1.
    dataGridView1.DataSource = BindingSource1;

}

```

```

Private Sub InitializeSortedFilteredBindingSource()

    ' Create the connection string, data adapter and data table.
    Dim connectionString As New SqlConnection("Initial Catalog=Northwind;" & _
        "Data Source=localhost;Integrated Security=SSPI;")
    Dim customersTableAdapter As New SqlDataAdapter("Select * from Customers", _
        connectionString)
    Dim customerTable As New DataTable()

    ' Fill the adapter with the contents of the customer table.
    customersTableAdapter.Fill(customerTable)

    ' Set data source for BindingSource1.
    BindingSource1.DataSource = customerTable

    ' Filter the items to show contacts who are owners.
    BindingSource1.Filter = "ContactTitle='Owner'"
    ' Sort the items on the company name in descending order.
    BindingSource1.Sort = "Country DESC, Address ASC"

    ' Set the data source for dataGridView1 to BindingSource1.
    dataGridView1.DataSource = BindingSource1

End Sub

```

Compiling the Code

To run this example, paste the code into a form that contains a **BindingSource** named `BindingSource1` and a **DataGridView** named `dataGridView1`. Handle the **Load** event for the form and call `InitializeSortedFilteredBindingSource` in the load event handler method.

See Also

[Sort](#)

[Filter](#)

[How to: Install Sample Databases](#)

[BindingSource Component](#)

Button Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `Button` control allows the user to click it to perform an action. The `Button` control can display both text and images. When the button is clicked, it looks as if it is being pushed in and released.

In This Section

[Button Control Overview](#)

Explains what this control is and its key features and properties.

[How to: Respond to Windows Forms Button Clicks](#)

Explains the most basic use of a button on a Windows Form.

[How to: Designate a Windows Forms Button as the Accept Button](#)

Explains how to designate a `Button` control to be the accept button, also known as the default button.

[How to: Designate a Windows Forms Button as the Cancel Button](#)

Explains how to designate a `Button` control to be the cancel button, which is clicked whenever the user presses the ESC key.

[Ways to Select a Windows Forms Button Control](#)

Lists methods of selecting a button.

Also see [How to: Designate a Windows Forms Button as the Accept Button Using the Designer](#) and [How to: Designate a Windows Forms Button as the Cancel Button Using the Designer](#).

Reference

[Button class](#)

Describes this class and has links to all its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

Also see [User Input to Dialog Boxes](#) and [How to: Close Dialog Boxes and Retain User Input](#).

Button Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [Button](#) control allows the user to click it to perform an action. When the button is clicked, it looks as if it is being pushed in and released. Whenever the user clicks a button, the [Click](#) event handler is invoked. You place code in the [Click](#) event handler to perform any action you choose.

The text displayed on the button is contained in the [Text](#) property. If your text exceeds the width of the button, it will wrap to the next line. However, it will be clipped if the control cannot accommodate its overall height. For more information, see [How to: Set the Text Displayed by a Windows Forms Control](#). The [Text](#) property can contain an access key, which allows a user to "click" the control by pressing the ALT key with the access key. For details, see [How to: Create Access Keys for Windows Forms Controls](#). The appearance of the text is controlled by the [Font](#) property and the [TextAlign](#) property.

The [Button](#) control can also display images using the [Image](#) and [ImageList](#) properties. For more information, see [How to: Set the Image Displayed by a Windows Forms Control](#).

See Also

[Button](#)

[How to: Respond to Windows Forms Button Clicks](#)

[Ways to Select a Windows Forms Button Control](#)

[How to: Designate a Windows Forms Button as the Accept Button Using the Designer](#)

[How to: Designate a Windows Forms Button as the Cancel Button Using the Designer](#)

[Button Control](#)

How to: Designate a Windows Forms Button as the Accept Button

5/4/2018 • 1 min to read • [Edit Online](#)

On any Windows Form, you can designate a [Button](#) control to be the accept button, also known as the default button. Whenever the user presses the ENTER key, the default button is clicked regardless of which other control on the form has the focus.

NOTE

The exceptions to this are when the control with focus is another button — in that case, the button with the focus will be clicked — or a multiline text box, or a custom control that traps the ENTER key.

To designate the accept button

1. Set the form's [AcceptButton](#) property to the appropriate [Button](#) control.

```
Private Sub SetDefault(ByVal myDefaultBtn As Button)
    Me.AcceptButton = myDefaultBtn
End Sub
```

```
private void SetDefault(Button myDefaultBtn)
{
    this.AcceptButton = myDefaultBtn;
}
```

```
private:
void SetDefault(Button ^ myDefaultBtn)
{
    this->AcceptButton = myDefaultBtn;
}
```

See Also

- [AcceptButton](#)
- [Button Control Overview](#)
- [Ways to Select a Windows Forms Button Control](#)
- [How to: Respond to Windows Forms Button Clicks](#)
- [How to: Designate a Windows Forms Button as the Cancel Button](#)
- [Button Control](#)

How to: Designate a Windows Forms Button as the Accept Button Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

On any Windows Form, you can designate a [Button](#) control to be the accept button, also known as the default button. Whenever the user presses the ENTER key, the default button is clicked regardless of which other control on the form has the focus. The exceptions to this are when the control with focus is another button — in that case, the button with the focus will be clicked — or a multiline text box, or a custom control that traps the ENTER key.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To designate the accept button

1. Select the form on which the button resides.
2. In the **Properties** window, set the form's [AcceptButton](#) property to the [Button](#) control's name.

See Also

[AcceptButton](#)

[Button Control Overview](#)

[Ways to Select a Windows Forms Button Control](#)

[How to: Respond to Windows Forms Button Clicks](#)

[How to: Designate a Windows Forms Button as the Cancel Button Using the Designer](#)

[Button Control](#)

How to: Designate a Windows Forms Button as the Cancel Button

5/4/2018 • 1 min to read • [Edit Online](#)

On any Windows Form, you can designate a [Button](#) control to be the cancel button. A cancel button is clicked whenever the user presses the ESC key, regardless of which other control on the form has the focus. Such a button is usually programmed to enable the user to quickly exit an operation without committing to any action.

To designate the cancel button

1. Set the form's [CancelButton](#) property to the appropriate [Button](#) control.

```
Private Sub SetCancelButton(ByVal myCancelButton As Button)
    Me.CancelButton = myCancelButton
End Sub
```

```
private void SetCancelButton(Button myCancelButton)
{
    this.CancelButton = myCancelButton;
}
```

```
private:
void SetCancelButton(Button ^ myCancelButton)
{
    this->CancelButton = myCancelButton;
}
```

See Also

- [CancelButton](#)
- [Button Control Overview](#)
- [Ways to Select a Windows Forms Button Control](#)
- [How to: Respond to Windows Forms Button Clicks](#)
- [How to: Designate a Windows Forms Button as the Accept Button](#)
- [Button Control](#)

How to: Designate a Windows Forms Button as the Cancel Button Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

On any Windows Form, you can designate a [Button](#) control to be the cancel button. A cancel button is clicked whenever the user presses the ESC key, regardless of which other control on the form has the focus. Such a button is usually programmed to enable the user to quickly exit an operation without committing to any action.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To designate the cancel button

1. Select the form on which the button resides.
2. In the **Properties** window, set the form's [CancelButton](#) property to the [Button](#) control's name.

See Also

[CancelButton](#)

[Button Control Overview](#)

[Ways to Select a Windows Forms Button Control](#)

[How to: Respond to Windows Forms Button Clicks](#)

[How to: Designate a Windows Forms Button as the Accept Button Using the Designer](#)

[Button Control](#)

How to: Respond to Windows Forms Button Clicks

5/4/2018 • 1 min to read • [Edit Online](#)

The most basic use of a Windows Forms [Button](#) control is to run some code when the button is clicked.

Clicking a [Button](#) control also generates a number of other events, such as the [MouseEnter](#), [MouseDown](#), and [MouseUp](#) events. If you intend to attach event handlers for these related events, be sure that their actions do not conflict. For example, if clicking the button clears information that the user has typed in a text box, pausing the mouse pointer over the button should not display a tool tip with that now-nonexistent information.

If the user attempts to double-click the [Button](#) control, each click will be processed separately; that is, the control does not support the double-click event.

To respond to a button click

- In the button's `Click` [EventHandler](#) write the code to run. `Button1_Click` must be bound to the control.

For more information, see [How to: Create Event Handlers at Run Time for Windows Forms](#).

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
    Button1.Click
    MessageBox.Show("Button1 was clicked")
End Sub
```

```
private void button1_Click(object sender, System.EventArgs e)
{
    MessageBox.Show("button1 was clicked");
}
```

```
private:
void button1_Click(System::Object ^ sender,
    System::EventArgs ^ e)
{
    MessageBox::Show("button1 was clicked");
}
```

See Also

[Button Control Overview](#)

[Ways to Select a Windows Forms Button Control](#)

[Button Control](#)

Ways to Select a Windows Forms Button Control

5/4/2018 • 1 min to read • [Edit Online](#)

A Windows Forms button can be selected in the following ways:

- Use a mouse to click the button.
- Invoke the button's [Click](#) event in code.
- Move the focus to the button by pressing the TAB key, and then choose the button by pressing the SPACEBAR or ENTER.
- Press the access key (ALT + the underlined letter) for the button. For more information about access keys, see [How to: Create Access Keys for Windows Forms Controls](#).
- If the button is the "accept" button of the form, pressing ENTER chooses the button, even if another control has the focus — except if that other control is another button, a multi-line text box, or a custom control that traps the enter key.
- If the button is the "cancel" button of the form, pressing ESC chooses the button, even if another control has the focus.
- Call the [PerformClick](#) method to select the button programmatically.

See Also

[Button Control Overview](#)

[How to: Respond to Windows Forms Button Clicks](#)

[Button Control](#)

CheckBox Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `CheckBox` control indicates whether a particular condition is on or off. It is commonly used to present a Yes/No or True/False selection to the user. You can use check box controls in groups to display multiple choices from which the user can select one or more. It is similar to the `RadioButton` control, but any number of grouped `CheckBox` controls may be selected.

In This Section

[CheckBox Control Overview](#)

Explains what this control is and its key features and properties.

[How to: Respond to Windows Forms CheckBox Clicks](#)

Explains how to use a check box to determine your application's actions.

[How to: Set Options with Windows Forms CheckBox Controls](#)

Describes how to use a check box to set options such as properties of an object.

Reference

[CheckBox class](#)

Describes this class and has links to all its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

CheckBox Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [CheckBox](#) control indicates whether a particular condition is on or off. It is commonly used to present a Yes/No or True/False selection to the user. You can use check box controls in groups to display multiple choices from which the user can select one or more.

The check box control is similar to the radio button control in that each is used to indicate a selection that is made by the user. They differ in that only one radio button in a group can be selected at a time. With the check box control, however, any number of check boxes may be selected.

A check box may be connected to elements in a database using simple data binding. Multiple check boxes may be grouped using the [GroupBox](#) control. This is useful for visual appearance and also for user interface design, since grouped controls can be moved around together on the form designer. For more information, see [Windows Forms Data Binding](#) and [GroupBox Control](#).

The [CheckBox](#) control has two important properties, [Checked](#) and [CheckState](#). The [Checked](#) property returns either `true` or `false`. The [CheckState](#) property returns either [Checked](#) or [Unchecked](#); or, if the [ThreeState](#) property is set to `true`, [CheckState](#) may also return [Indeterminate](#). In the indeterminate state, the box is displayed with a dimmed appearance to indicate the option is unavailable.

See Also

[CheckBox](#)

[How to: Set Options with Windows Forms CheckBox Controls](#)

[How to: Respond to Windows Forms CheckBox Clicks](#)

[CheckBox Control](#)

How to: Respond to Windows Forms CheckBox Clicks

5/4/2018 • 2 min to read • [Edit Online](#)

Whenever a user clicks a Windows Forms [CheckBox](#) control, the [Click](#) event occurs. You can program your application to perform some action depending upon the state of the check box.

To respond to CheckBox clicks

1. In the [Click](#) event handler, use the [Checked](#) property to determine the control's state, and perform any necessary action.

```
Private Sub CheckBox1_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles CheckBox1.Click
    ' The CheckBox control's Text property is changed each time the
    ' control is clicked, indicating a checked or unchecked state.
    If CheckBox1.Checked = True Then
        CheckBox1.Text = "Checked"
    Else
        CheckBox1.Text = "Unchecked"
    End If
End Sub
```

```
private void checkBox1_Click(object sender, System.EventArgs e)
{
    // The CheckBox control's Text property is changed each time the
    // control is clicked, indicating a checked or unchecked state.
    if (checkBox1.Checked)
    {
        checkBox1.Text = "Checked";
    }
    else
    {
        checkBox1.Text = "Unchecked";
    }
}
```

```
private:
void checkBox1_CheckedChanged(System::Object ^ sender,
    System::EventArgs ^ e)
{
    if (checkBox1->Checked)
    {
        checkBox1->Text = "Checked";
    }
    else
    {
        checkBox1->Text = "Unchecked";
    }
}
```

NOTE

If the user attempts to double-click the [CheckBox](#) control, each click will be processed separately; that is, the [CheckBox](#) control does not support the double-click event.

NOTE

When the `AutoCheck` property is `true` (the default), the `CheckBox` is automatically selected or cleared when it is clicked. Otherwise, you must manually set the `Checked` property when the `Click` event occurs.

You can also use the `CheckBox` control to determine a course of action.

To determine a course of action when a check box is clicked

1. Use a case statement to query the value of the `CheckState` property to determine a course of action. When the `ThreeState` property is set to `true`, the `CheckState` property may return three possible values, which represent the box being checked, the box being unchecked, or a third indeterminate state in which the box is displayed with a dimmed appearance to indicate the option is unavailable.

```
Private Sub CheckBox1_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles CheckBox1.Click
    Select Case CheckBox1.CheckState
        Case CheckState.Checked
            ' Code for checked state.
        Case CheckState.Unchecked
            ' Code for unchecked state.
        Case CheckState.Indeterminate
            ' Code for indeterminate state.
    End Select
End Sub
```

```
private void checkBox1_Click(object sender, System.EventArgs e)
{
    switch(checkBox1.CheckState)
    {
        case CheckState.Checked:
            // Code for checked state.
            break;
        case CheckState.Unchecked:
            // Code for unchecked state.
            break;
        case CheckState.Indeterminate:
            // Code for indeterminate state.
            break;
    }
}
```

```
private:
void checkBox1_CheckedChanged(System::Object ^ sender,
System::EventArgs ^ e)
{
    switch(checkBox1->CheckState) {
        case CheckState::Checked:
            // Code for checked state.
            break;
        case CheckState::Unchecked:
            // Code for unchecked state.
            break;
        case CheckState::Indeterminate:
            // Code for indeterminate state.
            break;
    }
}
```

NOTE

When the `ThreeState` property is set to `true`, the `Checked` property returns `true` for both `Checked` and `Indeterminate`.

See Also

[CheckBox](#)

[CheckBox Control Overview](#)

[How to: Set Options with Windows Forms CheckBox Controls](#)

[CheckBox Control](#)

How to: Set Options with Windows Forms CheckBox Controls

5/4/2018 • 1 min to read • [Edit Online](#)

A Windows Forms [CheckBox](#) control is used to give users True/False or Yes/No options. The control displays a check mark when it is selected.

To set options with CheckBox controls

1. Examine the value of the [Checked](#) property to determine its state, and use that value to set an option.

In the code sample below, when the [CheckBox](#) control's [CheckedChanged](#) event is raised, the form's [AllowDrop](#) property is set to `false` if the check box is checked. This is useful for situations where you want to restrict user interaction.

```
Private Sub CheckBox1_CheckedChanged(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles CheckBox1.CheckedChanged
    ' Determine the CheckState of the check box.
    If CheckBox1.CheckState = CheckState.Checked Then
        ' If checked, do not allow items to be dragged onto the form.
        Me.AllowDrop = False
    End If
End Sub
```

```
private void checkBox1_CheckedChanged(object sender, System.EventArgs e)
{
    // Determine the CheckState of the check box.
    if (checkBox1.CheckState == CheckState.Checked)
    {
        // If checked, do not allow items to be dragged onto the form.
        this.AllowDrop = false;
    }
}
```

```
private:
void checkBox1_CheckedChanged(System::Object ^ sender,
    System::EventArgs ^ e)
{
    // Determine the CheckState of the check box.
    if (checkBox1->CheckState == CheckState::Checked)
    {
        // If checked, do not allow items to be dragged onto the form.
        this->AllowDrop = false;
    }
}
```

See Also

[CheckBox](#)

[CheckBox Control Overview](#)

[How to: Respond to Windows Forms CheckBox Clicks](#)

[CheckBox Control](#)

CheckedListBox Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `CheckedListBox` control displays a list of items, like the [ListBox](#) control, and also can display a check mark next to items in the list.

In This Section

[CheckedListBox Control Overview](#)

Explains what this control is and its key features and properties.

[How to: Determine Checked Items in the Windows Forms CheckedListBox Control](#)

Describes how to step through a list to determine which items are checked.

Reference

[CheckedListBox class](#)

Describes this class and has links to all its members.

Related Sections

[Windows Forms Controls Used to List Options](#)

Provides a list of things you can do with list boxes, combo boxes, and checked list boxes.

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

CheckedListBox Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [CheckedListBox](#) control extends the [ListBox](#) control. It does almost everything that a list box does and also can display a check mark next to items in the list. Other differences between the two controls are that checked list boxes only support [DrawMode.Normal](#); and that checked list boxes can only have one item or none selected. Note that a selected item appears highlighted on the form and is not the same as a checked item.

Checked list boxes can have items added at design time using the [String Collection Editor](#) or their items can be added dynamically from a collection at run time, using the [Items](#) property. For more information, see [How to: Add and Remove Items from a Windows Forms ComboBox, ListBox, or CheckedListBox Control](#).

See Also

[CheckedListBox](#)

[CheckedListBox.Items](#)

[ListControl.DataSource](#)

[ListBox Control Overview](#)

[Windows Forms Controls Used to List Options](#)

[How to: Determine Checked Items in the Windows Forms CheckedListBox Control](#)

How to: Determine Checked Items in the Windows Forms CheckedListBox Control

5/4/2018 • 2 min to read • [Edit Online](#)

When presenting data in a Windows Forms `CheckedListBox` control, you can either iterate through the collection stored in the `CheckedItems` property, or step through the list using the `GetItemChecked` method to determine which items are checked. The `GetItemChecked` method takes an item index number as its argument and returns `true` or `false`. Contrary to what you might expect, the `SelectedItems` and `SelectedIndices` properties do not determine which items are checked; they determine which items are highlighted.

To determine checked items in a `CheckedListBox` control

1. Iterate through the `CheckedItems` collection, starting at 0 since the collection is zero-based. Note that this method will give you the item number in the list of checked items, not the overall list. So if the first item in the list is not checked and the second item is checked, the code below will display text like "Checked Item 1 = MyListItem2".

```
' Determine if there are any items checked.  
If CheckedListBox1.CheckedItems.Count <> 0 Then  
    ' If so, loop through all checked items and print results.  
    Dim x As Integer  
    Dim s As String = ""  
    For x = 0 To CheckedListBox1.CheckedItems.Count - 1  
        s = s & "Checked Item " & (x + 1).ToString & " = " & CheckedListBox1.CheckedItems(x).ToString &  
        ControlChars.CrLf  
    Next x  
    MessageBox.Show(s)  
End If
```

```
// Determine if there are any items checked.  
if(checkedListBox1.CheckedItems.Count != 0)  
{  
    // If so, loop through all checked items and print results.  
    string s = "";  
    for(int x = 0; x <= checkedListBox1.CheckedItems.Count - 1 ; x++)  
    {  
        s = s + "Checked Item " + (x+1).ToString() + " = " + checkedListBox1.CheckedItems[x].ToString() +  
        "\n";  
    }  
    MessageBox.Show (s);  
}
```

```

// Determine if there are any items checked.
if(chedTextBox1->CheckedItems->Count != 0)
{
    // If so, loop through all checked items and print results.
    String ^ s = "";
    for(int x = 0; x <= checkedListBox1->CheckedItems->Count - 1; x++)
    {
        s = String::Concat(s, "Checked Item ", (x+1).ToString(),
                           " = ", checkedListBox1->CheckedItems[x]->ToString(),
                           "\n");
    }
    MessageBox::Show(s);
}

```

● or -

- Step through the [Items](#) collection, starting at 0 since the collection is zero-based, and call the [GetItemChecked](#) method for each item. Note that this method will give you the item number in the overall list, so if the first item in the list is not checked and the second item is checked, it will display something like "Item 2 = MyListItem2".

```

Dim i As Integer
Dim s As String
s = "Checked Items:" & ControlChars.CrLf
For i = 0 To (CheckedListBox1.Items.Count - 1)
    If CheckedListBox1.GetItemChecked(i) = True Then
        s = s & "Item " & (i + 1).ToString & " = " & CheckedListBox1.Items(i).ToString & ControlChars.CrLf
    End If
Next
MessageBox.Show(s)

```

```

int i;
string s;
s = "Checked items:\n" ;
for (i = 0; i <= (checkedListBox1.Items.Count-1); i++)
{
    if (checkedListBox1.GetItemChecked(i))
    {
        s = s + "Item " + (i+1).ToString() + " = " + checkedListBox1.Items[i].ToString() + "\n";
    }
}
MessageBox.Show (s);

```

```

int i;
String ^ s;
s = "Checked items:\n" ;
for (i = 0; i <= (checkedListBox1->Items->Count-1); i++)
{
    if (checkedListBox1->GetItemChecked(i))
    {
        s = String::Concat(s, "Item ", (i+1).ToString(), " = ",
                           checkedListBox1->Item[i]->ToString(), "\n");
    }
}
MessageBox::Show(s);

```

See Also

[Windows Forms Controls Used to List Options](#)

ColorDialog Component (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [ColorDialog](#) component is a pre-configured dialog box that allows the user to select a color from a palette and to add custom colors to that palette. It is the same dialog box that you see in other Windows-based applications to select colors. Use it within your Windows-based application as a simple solution in lieu of configuring your own dialog box.

In This Section

[ColorDialog Component Overview](#)

Introduces the general concepts of the [ColorDialog](#) component, which allows you to display a pre-configured dialog box that users can use to select colors from a palette.

[How to: Change the Appearance of the Windows Forms ColorDialog Component](#)

Describes how to change the colors available to users and other properties.

[How to: Show a Color Palette with the ColorDialog Component](#)

Explains how to select a color at run time by means of an instance of the [ColorDialog](#) component.

Related Sections

[Controls You Can Use On Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

[ColorDialog](#)

Provides reference information on the [ColorDialog](#) class and its members.

[Essential Code for Windows Forms Dialog Boxes](#)

Discusses the Windows Forms dialog box controls and components and the code necessary for executing their basic functions.

[Dialog-Box Controls and Components](#)

Lists a set of controls that allow users to perform standard interactions with the application or system.

ColorDialog Component Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [ColorDialog](#) component is a pre-configured dialog box that allows the user to select a color from a palette and to add custom colors to that palette. It is the same dialog box that you see in other Windows-based applications to select colors. Use it within your Windows-based application as a simple solution in lieu of configuring your own dialog box.

The color selected in the dialog box is returned in the [Color](#) property. If the [AllowFullOpen](#) property is set to `false`, the "Define Custom Colors" button is disabled and the user is restricted to the predefined colors in the palette. If the [SolidColorOnly](#) property is set to `true`, the user cannot select dithered colors. To display the dialog box, you must call its [ShowDialog](#) method.

See Also

[ColorDialog](#)

[ColorDialog Component](#)

[Dialog-Box Controls and Components](#)

[How to: Change the Appearance of the Windows Forms ColorDialog Component](#)

How to: Change the Appearance of the Windows Forms ColorDialog Component

5/4/2018 • 1 min to read • [Edit Online](#)

You can configure the appearance of the Windows Forms [ColorDialog](#) component with a number of its properties. The dialog box has two sections — one that shows basic colors and one that allows the user to define custom colors.

Most of the properties restrict what colors the user can select from the dialog box. If the [AllowFullOpen](#) property is set to `true`, the user is allowed to define custom colors. The [FullOpen](#) property is `true` if the dialog box is expanded to define custom colors; otherwise the user must click a "Define Custom Colors" button. When the [AnyColor](#) property is set to `true`, the dialog box displays all available colors in the set of basic colors. If the [SolidColorOnly](#) property is set to `true`, the user cannot select dithered colors; only solid colors are available to select.

If the [ShowHelp](#) property is set to `true`, a Help button appears on the dialog box. When the user clicks the Help button, the [ColorDialog](#) component's [HelpRequest](#) event is raised.

To configure the appearance of the color dialog box

1. Set the [AllowFullOpen](#), [AnyColor](#), [SolidColorOnly](#), and [ShowHelp](#) properties to the desired values.

```
ColorDialog1.AllowFullOpen = True  
ColorDialog1.AnyColor = True  
ColorDialog1.SolidColorOnly = False  
ColorDialog1.ShowHelp = True
```

```
colorDialog1.AllowFullOpen = true;  
colorDialog1.AnyColor = true;  
colorDialog1.SolidColorOnly = false;  
colorDialog1.ShowHelp = true;
```

```
colorDialog1->AllowFullOpen = true;  
colorDialog1->AnyColor = true;  
colorDialog1->SolidColorOnly = false;  
colorDialog1->ShowHelp = true;
```

See Also

[ColorDialog](#)

[ColorDialog Component](#)

[ColorDialog Component Overview](#)

How to: Show a Color Palette with the ColorDialog Component

5/4/2018 • 1 min to read • [Edit Online](#)

The [ColorDialog](#) component displays a palette of colors and returns a property containing the color the user has selected.

To choose a color using the ColorDialog component

1. Display the dialog box using the [ShowDialog](#) method.
2. Use the [DialogResult](#) property to determine how the dialog box was closed.
3. Use the [Color](#) property of the [ColorDialog](#) component to set the chosen color.

In the example below, the [Button](#) control's [Click](#) event handler opens a [ColorDialog](#) component. When a color is chosen and the user clicks **OK**, the [Button](#) control's background color is set to the chosen color. The example assumes your form has a [Button](#) control and a [ColorDialog](#) component.

```
Private Sub Button1_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles Button1.Click  
    If ColorDialog1.ShowDialog() = DialogResult.OK Then  
        Button1.BackColor = ColorDialog1.Color  
    End If  
End Sub
```

```
private void button1_Click(object sender, System.EventArgs e)  
{  
    if(colorDialog1.ShowDialog() == DialogResult.OK)  
    {  
        button1.BackColor = colorDialog1.Color;  
    }  
}
```

```
private:  
    void button1_Click(System::Object ^ sender,  
        System::EventArgs ^ e)  
    {  
        if(colorDialog1->ShowDialog() == DialogResult::OK)  
        {  
            button1->BackColor = colorDialog1->Color;  
        }  
    }  
}
```

(Visual C#, Visual C++) Place the following code in the form's constructor to register the event handler.

```
this.button1.Click += new System.EventHandler(this.button1_Click);
```

```
this->button1->Click +=  
    gcnew System::EventHandler(this, &Form1::button1_Click);
```

See Also

[ColorDialog](#)

[ColorDialog Component](#)

ComboBox Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `ComboBox` control is used to display data in a drop-down combo box. By default, the `ComboBox` control appears in two parts: the top part is a text box that allows the user to type a list item. The second part is a list box that displays a list of items from which the user can select one.

In This Section

[ComboBox Control Overview](#)

Explains what this control is and its key features and properties.

[How to: Create Variable Sized Text in a ComboBox Control](#)

Demonstrates custom drawing of text in a `ComboBox` control.

Reference

[ComboBox class](#)

Describes this class and has links to all its members.

Related Sections

[Windows Forms Controls Used to List Options](#)

Provides a list of things you can do with list boxes, combo boxes, and checked list boxes.

See Also

[Controls to Use on Windows Forms](#)

ComboBox Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [ComboBox](#) control is used to display data in a drop-down combo box. By default, the [ComboBox](#) control appears in two parts: the top part is a text box that allows the user to type a list item. The second part is a list box that displays a list of items from which the user can select one. For more information on other styles of combo box, see [When to Use a Windows Forms ComboBox Instead of a ListBox](#).

The [SelectedIndex](#) property returns an integer value that corresponds to the selected list item. You can programmatically change the selected item by changing the [SelectedIndex](#) value in code; the corresponding item in the list will appear in the text box portion of the combo box. If no item is selected, the [SelectedIndex](#) value is -1. If the first item in the list is selected, then the [SelectedIndex](#) value is 0. The [SelectedItem](#) property is similar to [SelectedIndex](#), but returns the item itself, usually a string value. The [Count](#) property reflects the number of items in the list, and the value of the [Count](#) property is always one more than the largest possible [SelectedIndex](#) value because [SelectedIndex](#) is zero-based.

To add or delete items in a [ComboBox](#) control, use the [Add](#), [Insert](#), [Clear](#) or [Remove](#) method. Alternatively, you can add items to the list by using the [Items](#) property in the designer.

See Also

[ComboBox](#)

[ListBox Control Overview](#)

[When to Use a Windows Forms ComboBox Instead of a ListBox](#)

[How to: Add and Remove Items from a Windows Forms ComboBox, ListBox, or CheckedListBox Control](#)

[How to: Sort the Contents of a Windows Forms ComboBox, ListBox, or CheckedListBox Control](#)

[How to: Access Specific Items in a Windows Forms ComboBox, ListBox, or CheckedListBox Control](#)

[How to: Bind a Windows Forms ComboBox or ListBox Control to Data](#)

[Windows Forms Controls Used to List Options](#)

[How to: Create a Lookup Table for a Windows Forms ComboBox, ListBox, or CheckedListBox Control](#)

How to: Create Variable Sized Text in a ComboBox Control

5/4/2018 • 1 min to read • [Edit Online](#)

This example demonstrates custom drawing of text in a [ComboBox](#) control. When an item meets a certain criteria, it is drawn in a larger font and turned red.

Example

```
Private Sub ComboBox1_MeasureItem(ByVal sender As Object, ByVal e As _
System.Windows.Forms.MeasureItemEventArgs) Handles ComboBox1.MeasureItem
    Dim bFont As New Font("Arial", 8, FontStyle.Bold)
    Dim lFont As New Font("Arial", 12, FontStyle.Bold)
    Dim siText As SizeF

    If ComboBox1.Items().Item(e.Index) = "Two" Then
        siText = e.Graphics.MeasureString(ComboBox1.Items().Item(e.Index), _
lFont)
    Else
        siText = e.Graphics.MeasureString(ComboBox1.Items().Item(e.Index), bFont)
    End If

    e.ItemHeight = siText.Height
    e.ItemWidth = siText.Width
End Sub

Private Sub ComboBox1_DrawItem(ByVal sender As Object, ByVal e As _
System.Windows.Forms.DrawItemEventArgs) Handles ComboBox1.DrawItem
    Dim g As Graphics = e.Graphics
    Dim bFont As New Font("Arial", 8, FontStyle.Bold)
    Dim lFont As New Font("Arial", 12, FontStyle.Bold)

    If ComboBox1.Items().Item(e.Index) = "Two" Then
        g.DrawString(ComboBox1.Items.Item(e.Index), lFont, Brushes.Red, _
e.Bounds.X, e.Bounds.Y)
    Else
        g.DrawString(ComboBox1.Items.Item(e.Index), bFont, Brushes.Black, e.Bounds.X, e.Bounds.Y)
    End If
End Sub
```

Compiling the Code

This example requires:

- A Windows form.
- A [ComboBox](#) control named `ListBox1` with three items in the [Items](#) property. In this example, the three items are named `"One", "Two", and "Three"`. The [DrawMode](#) property of `ComboBox1` must be set to [OwnerDrawVariable](#).

NOTE

This technique is also applicable to the [ListBox](#) control — you can substitute a [ListBox](#) for the [ComboBox](#).

- References to the [System.Windows.Forms](#) and [System.Drawing](#) namespaces.

See Also

[DrawItem](#)

[DrawItemEventArgs](#)

[MeasureItem](#)

[Controls with Built-In Owner-Drawing Support](#)

[ListBox Control](#)

[ComboBox Control](#)

ContextMenu Component (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

NOTE

Although `MenuStrip` and `ContextMenuStrip` replace and add functionality to the `MainMenu` and `ContextMenu` controls of previous versions, `MainMenu` and `ContextMenu` are retained for both backward compatibility and future use if you choose.

The Windows Forms `ContextMenu` component is used to provide users with an easily accessible shortcut menu of frequently used commands that are associated with the selected object. The items in a shortcut menu are frequently a subset of the items from main menus that appear elsewhere in the application. Shortcut menus are usually available by right-clicking the mouse. On Windows Forms they are associated with other controls.

In This Section

[ContextMenu Component Overview](#)

Introduces the general concepts of the `ContextMenu` component, which allows users to create menus of frequently used commands associated with a selected object.

[How to: Add and Remove Menu Items with the Windows Forms ContextMenu Component](#)

Explains how to add and remove shortcut menu items in Windows Forms.

Reference

[ContextMenu](#)

Provides reference information on the `ContextMenu` class and its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

See Also

[MenuStrip](#)

[ContextMenuStrip](#)

ContextMenu Component Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

IMPORTANT

Although [MenuStrip](#) and [ContextMenuStrip](#) replace and add functionality to the [MainMenu](#) and [ContextMenu](#) controls of previous versions, [MainMenu](#) and [ContextMenu](#) are retained for both backward compatibility and future use if you choose.

With the Windows Forms [ContextMenu](#) component, you can provide users with an easily accessible shortcut menu of frequently used commands that are associated with the selected object. The items in a shortcut menu are frequently a subset of the items from main menus that appear elsewhere in the application. A user can typically access a shortcut menu by right-clicking the mouse. On Windows Forms, shortcut menus are associated with controls.

Key Properties

You can associate a shortcut menu with a control by setting the control's [ContextMenu](#) property to the [ContextMenu](#) component. A single shortcut menu can be associated with multiple controls, but each control can have only one shortcut menu.

The key property of the [ContextMenu](#) component is the [MenuItems](#) property. You can add menu items by programmatically creating [MenuItem](#) objects and adding them to the [Menu.MenuItemCollection](#) of the shortcut menu. Because the items in a shortcut menu are usually drawn from other menus, you will most frequently add items to a shortcut menu by copying them.

See Also

[ContextMenu](#)

[MenuStrip](#)

[ContextMenuStrip](#)

How to: Add and Remove Menu Items with the Windows Forms ContextMenu Component

5/4/2018 • 1 min to read • [Edit Online](#)

Explains how to add and remove shortcut menu items in Windows Forms.

The Windows Forms [ContextMenu](#) component provides a menu of frequently used commands that are relevant to the selected object. You can add items to the shortcut menu by adding [MenuItem](#) objects to the [MenuItems](#) collection.

You can remove items from a shortcut menu permanently; however, at run time it may be more appropriate to hide or disable the items instead.

IMPORTANT

Although [MenuStrip](#) and [ContextMenuStrip](#) replace and add functionality to the [MainMenu](#) and [ContextMenu](#) controls of previous versions, [MainMenu](#) and [ContextMenu](#) are retained for both backward compatibility and future use if you choose.

To remove items from a shortcut menu

1. Use the [Remove](#) or [RemoveAt](#) method of the [MenuItems](#) collection of the [ContextMenu](#) component to remove a particular menu item.

```
' Removes the first item in the shortcut menu.  
ContextMenu1.MenuItems.RemoveAt(0)  
' Removes a particular object from the shortcut menu.  
ContextMenu1.MenuItems.Remove(mnuItemNew)
```

```
// Removes the first item in the shortcut menu.  
contextMenu1.MenuItems.RemoveAt(0);  
// Removes a particular object from the shortcut menu.  
contextMenu1.MenuItems.Remove(mnuItemNew);
```

```
// Removes the first item in the shortcut menu.  
contextMenu1->MenuItems->RemoveAt(0);  
// Removes a particular object from the shortcut menu.  
contextMenu1->MenuItems->Remove(mnuItemNew);
```

-or-

2. Use the [Clear](#) method of the [MenuItems](#) collection of the [ContextMenu](#) component to remove all items from the menu.

```
ContextMenu1.MenuItems.Clear()
```

```
contextMenu1.MenuItems.Clear();
```

```
contextMenu1->MenuItems->Clear();
```

See Also

[ContextMenu](#)

[ContextMenu Component](#)

[ContextMenu Component Overview](#)

ContextMenuStrip Control

5/4/2018 • 1 min to read • [Edit Online](#)

The [ContextMenuStrip](#) control provides a shortcut menu that you associate with a control.

In This Section

[ContextMenuStrip Control Overview](#)

Explains what the control is and its key features and properties.

[How to: Associate a ContextMenuStrip with a Control](#)

Describes making a [ContextMenuStrip](#) the shortcut menu for a specific control.

[How to: Add Menu Items to a ContextMenuStrip](#)

Describes how to add selectable options to a [ContextMenuStrip](#).

[How to: Configure ContextMenuStrip Check Margins and Image Margins](#)

Describes how to customize a [ContextMenuStrip](#) by setting check and image margin properties in various ways.

[How to: Enable Check Margins and Image Margins in ContextMenuStrip Controls](#)

Describes how to turn [ContextMenuStrip](#) check margins on and off.

[How to: Handle the ContextMenuStrip Opening Event](#)

Describes how to customize the behavior of a [ContextMenuStrip](#) control by handling the [Opening](#) event.

Also see [ContextMenuStrip Tasks Dialog Box](#) or [ContextMenuStrip Items Collection Editor](#).

Reference

[MenuStrip](#)

Describes the features of the [MenuStrip](#) class, which provides a menu system for a form.

[ContextMenuStrip](#)

Describes the features of the [ContextMenuStrip](#), which represents a shortcut menu.

[ToolStripMenuItem](#)

Describes the features of the [ToolStripMenuItem](#) class, which represents a selectable option displayed on a [MenuStrip](#) or [ContextMenuStrip](#).

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

ContextMenuStrip Control Overview

5/4/2018 • 1 min to read • [Edit Online](#)

NOTE

The [ContextMenuStrip](#) control replaces and adds functionality to the [ContextMenu](#) control; however, the [ContextMenu](#) control is retained for backward compatibility and future use if you choose.

Shortcut menus, also called context menus, appear at the mouse position when the user clicks the right mouse button. Shortcut *menus* provide options for the client area or the control at the mouse pointer location.

The [ContextMenuStrip](#) control is designed to work seamlessly with the new [ToolStrip](#) and related controls, but you can associate a [ContextMenuStrip](#) with other controls just as easily.

The following table shows the important [ContextMenuStrip](#) companion classes.

CLASS	DESCRIPTION
ToolStripMenuItem	Represents a selectable option displayed on a MenuStrip or ContextMenuStrip .
ToolStripDropDown	Represents a control that enables the user to select a single item from a list that is displayed when the user clicks a ToolStripDropDownButton or a higher-level menu item.
ToolStripDropDownItem	Provides basic functionality for controls derived from ToolStripItem that display drop-down items when clicked.

See Also

[ToolStrip](#)
[MenuStrip](#)
[ContextMenuStrip](#)
[ToolStripMenuItem](#)
[ToolStripDropDown](#)

How to: Associate a ContextMenuStrip with a Control

5/4/2018 • 4 min to read • [Edit Online](#)

After creating your controls and shortcut menus, use the following procedures to display a given shortcut menu when the user right-clicks the control. These procedures associate a [ContextMenuStrip](#) with a Windows Form and with a [ToolStrip](#) control.

To associate a ContextMenuStrip with a Windows Form

1. Set the [ContextMenuStrip](#) property to the name of the associated [ContextMenuStrip](#).

To associate a ContextMenuStrip with a ToolStrip control

1. Set the control's [ContextMenuStrip](#) property to the name of the associated [ContextMenuStrip](#).

Example

The following code example creates a Windows Form and a [ToolStrip](#), and associates a different [ContextMenuStrip](#) control with each of them.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace WindowsApplication10
{
    public class Form1 : Form
    {
        private ToolStripButton toolStripButton1;
        private ToolStripButton toolStripButton2;
        private ToolStripButton toolStripButton3;
        private ContextMenuStrip contextMenuStrip1;
        private.IContainer components;
        private ToolStripMenuItem toolStripMenuItem1;
        private ToolStripMenuItem toolStripMenuItem2;
        private ContextMenuStrip contextMenuStrip2;
        private ToolStripMenuItem rearrangeButtonsToolStripMenuItem;
        private ToolStripMenuItem selectIconsToolStripMenuItem;
        private ToolStrip toolStrip1;

        public Form1()
        {
            InitializeComponent();
        }
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.Run(new Form1());
        }

        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(Form1));
            this.toolStrip1 = new System.Windows.Forms.ToolStrip();
```

```
this.toolStripButton1 = new System.Windows.Forms.ToolStripButton();
this.toolStripButton2 = new System.Windows.Forms.ToolStripButton();
this.toolStripButton3 = new System.Windows.Forms.ToolStripButton();
this.contextMenuStrip1 = new System.Windows.Forms.ContextMenuStrip(this.components);
this.contextMenuStrip2 = new System.Windows.Forms.ContextMenuStrip(this.components);
this.toolStripMenuItem1 = new System.Windows.Forms.ToolStripItem();
this.toolStripMenuItem2 = new System.Windows.Forms.ToolStripItem();
this.rearrangeButtonsToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
this.selectIconsToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
this.toolStrip1.SuspendLayout();
this.contextMenuStrip1.SuspendLayout();
this.contextMenuStrip2.SuspendLayout();
this.SuspendLayout();
//
// Associate contextMenuStrip2 with toolStrip1.
// toolStrip1 property settings follow.
//
this.toolStrip1.ContextMenuStrip = this.contextMenuStrip2;
this.toolStrip1.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
this.toolStripButton1,
this.toolStripButton2,
this.toolStripButton3});
this.toolStrip1.Location = new System.Drawing.Point(0, 0);
this.toolStrip1.Name = "toolStrip1";
this.toolStrip1.Size = new System.Drawing.Size(292, 25);
this.toolStrip1.TabIndex = 0;
this.toolStrip1.Text = "toolStrip1";
//
// toolStripButton1
//
this.toolStripButton1.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image;
this.toolStripButton1.Image = ((System.Drawing.Image)
(resources.GetObject("toolStripButton1.Image")));
this.toolStripButton1.ImageTransparentColor = System.Drawing.Color.Magenta;
this.toolStripButton1.Name = "toolStripButton1";
this.toolStripButton1.Text = "toolStripButton1";
//
// toolStripButton2
//
this.toolStripButton2.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image;
this.toolStripButton2.Image = ((System.Drawing.Image)
(resources.GetObject("toolStripButton2.Image")));
this.toolStripButton2.ImageTransparentColor = System.Drawing.Color.Magenta;
this.toolStripButton2.Name = "toolStripButton2";
this.toolStripButton2.Text = "toolStripButton2";
//
// toolStripButton3
//
this.toolStripButton3.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image;
this.toolStripButton3.Image = ((System.Drawing.Image)
(resources.GetObject("toolStripButton3.Image")));
this.toolStripButton3.ImageTransparentColor = System.Drawing.Color.Magenta;
this.toolStripButton3.Name = "toolStripButton3";
this.toolStripButton3.Text = "toolStripButton3";
//
// contextMenuStrip1
//
this.contextMenuStrip1.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
this.toolStripMenuItem1,
this.toolStripMenuItem2});
this.contextMenuStrip1.Name = "contextMenuStrip1";
this.contextMenuStrip1.RightToLeft = System.Windows.Forms.RightToLeft.No;
this.contextMenuStrip1.Size = new System.Drawing.Size(131, 48);
//
// contextMenuStrip2
//
this.contextMenuStrip2.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
this.rearrangeButtonsToolStripMenuItem,
this.selectIconsToolStripMenuItem});
```

```

        this.contextMenuStrip2.Name = "contextMenuStrip2";
        this.contextMenuStrip2.RightToLeft = System.Windows.Forms.RightToLeft.No;
        this.contextMenuStrip2.Size = new System.Drawing.Size(162, 48);
        //
        // toolStripMenuItem1
        //
        this.toolStripMenuItem1.Name = "toolStripMenuItem1";
        this.toolStripMenuItem1.Text = "&Resize";
        //
        // toolStripMenuItem2
        //
        this.toolStripMenuItem2.Name = "toolStripMenuItem2";
        this.toolStripMenuItem2.Text = "&Keep on Top";
        //
        // rearrangeButtonsToolStripMenuItem
        //
        this.rearrangeButtonsToolStripMenuItem.Name = "rearrangeButtonsToolStripMenuItem";
        this.rearrangeButtonsToolStripMenuItem.Text = "R&earrange Buttons";
        //
        // selectIconsToolStripMenuItem
        //
        this.selectIconsToolStripMenuItem.Name = "selectIconsToolStripMenuItem";
        this.selectIconsToolStripMenuItem.Text = "&Select Icons";
        //
        // Associate contextMenuStrip1 with Form1.
        // Form1 property settings follow.
        //
        this.ClientSize = new System.Drawing.Size(292, 266);
        this.ContextMenuStrip = this.contextMenuStrip1;
        this.Controls.Add(this.toolStrip1);
        this.Name = "Form1";
        this.toolStrip1.ResumeLayout(false);
        this.contextMenuStrip1.ResumeLayout(false);
        this.contextMenuStrip2.ResumeLayout(false);
        this.ResumeLayout(false);
        this.PerformLayout();
    }
}
}

```

```

Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Text
Imports System.Windows.Forms

Public Class Form1
    Inherits Form
    Private toolStripButton1 As ToolStripButton
    Private toolStripButton2 As ToolStripButton
    Private toolStripButton3 As ToolStripButton
    Private contextMenuStrip1 As ContextMenuStrip
    Private components As.IContainer
    Private toolStripMenuItem1 As ToolStripMenuItem
    Private toolStripMenuItem2 As ToolStripMenuItem
    Private contextMenuStrip2 As ContextMenuStrip
    Private rearrangeButtonsToolStripMenuItem As ToolStripMenuItem
    Private selectIconsToolStripMenuItem As ToolStripMenuItem
    Private toolStrip1 As ToolStrip

    Public Sub New()
        InitializeComponent()
    End Sub

```

```

<STAThread()> _
Public Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())
End Sub

Private Sub InitializeComponent()
    Me.components = New System.ComponentModel.Container()
    Dim resources As New System.ComponentModel.ComponentResourceManager(GetType(Form1))
    Me.toolStrip1 = New System.Windows.Forms.ToolStrip()
    Me.toolStripButton1 = New System.Windows.Forms.ToolStripButton()
    Me.toolStripButton2 = New System.Windows.Forms.ToolStripButton()
    Me.toolStripButton3 = New System.Windows.Forms.ToolStripButton()
    Me.contextMenuStrip1 = New System.Windows.Forms.ContextMenuStrip(Me.components)
    Me.contextMenuStrip2 = New System.Windows.Forms.ContextMenuStrip(Me.components)
    Me.toolStripMenuItem1 = New System.Windows.Forms.ToolStripItem()
    Me.toolStripMenuItem2 = New System.Windows.Forms.ToolStripItem()
    Me.rearrangeButtonsToolStripMenuItem = New System.Windows.Forms.ToolStripItem()
    Me.selectIconsToolStripMenuItem = New System.Windows.Forms.ToolStripItem()
    Me.toolStripButton1.SuspendLayout()
    Me.contextMenuStrip1.SuspendLayout()
    Me.contextMenuStrip2.SuspendLayout()
    Me.SuspendLayout()

    ' Associate contextMenuStrip2 with toolStrip1.
    ' toolStrip1 property settings follow.

    Me.toolStrip1.ContextMenuStrip = Me.contextMenuStrip2
    Me.toolStrip1.Items.AddRange(New System.Windows.Forms.ToolStripItem() {Me.toolStripButton1,
    Me.toolStripButton2, Me.toolStripButton3})
    Me.toolStrip1.Location = New System.Drawing.Point(0, 0)
    Me.toolStrip1.Name = "toolStrip1"
    Me.toolStrip1.Size = New System.Drawing.Size(292, 25)
    Me.toolStrip1.TabIndex = 0
    Me.toolStrip1.Text = "toolStrip1"

    ' toolStripButton1

    Me.toolStripButton1.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image
    Me.toolStripButton1.Image = CType(resources.GetObject("toolStripButton1.Image"), System.Drawing.Image)
    Me.toolStripButton1.ImageTransparentColor = System.Drawing.Color.Magenta
    Me.toolStripButton1.Name = "toolStripButton1"
    Me.toolStripButton1.Text = "toolStripButton1"

    ' toolStripButton2

    Me.toolStripButton2.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image
    Me.toolStripButton2.Image = CType(resources.GetObject("toolStripButton2.Image"), System.Drawing.Image)
    Me.toolStripButton2.ImageTransparentColor = System.Drawing.Color.Magenta
    Me.toolStripButton2.Name = "toolStripButton2"
    Me.toolStripButton2.Text = "toolStripButton2"

    ' toolStripButton3

    Me.toolStripButton3.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image
    Me.toolStripButton3.Image = CType(resources.GetObject("toolStripButton3.Image"), System.Drawing.Image)
    Me.toolStripButton3.ImageTransparentColor = System.Drawing.Color.Magenta
    Me.toolStripButton3.Name = "toolStripButton3"
    Me.toolStripButton3.Text = "toolStripButton3"

    ' contextMenuStrip1

    Me.contextMenuStrip1.Items.AddRange(New System.Windows.Forms.ToolStripItem() {Me.toolStripMenuItem1,
    Me.toolStripMenuItem2})
    Me.contextMenuStrip1.Name = "contextMenuStrip1"
    Me.contextMenuStrip1.RightToLeft = System.Windows.Forms.RightToLeft.No
    Me.contextMenuStrip1.Size = New System.Drawing.Size(131, 48)

```

```

' contextMenuStrip2
'
Me.contextMenuStrip2.Items.AddRange(New System.Windows.Forms.ToolStripItem()
{Me.rearrangeButtonsToolStripMenuItem, Me.selectIconsToolStripMenuItem})
Me.contextMenuStrip2.Name = "contextMenuStrip2"
Me.contextMenuStrip2.RightToLeft = System.Windows.Forms.RightToLeft.No
Me.contextMenuStrip2.Size = New System.Drawing.Size(162, 48)
'
' toolStripMenuItem1
'
Me.toolStripMenuItem1.Name = "toolStripMenuItem1"
Me.toolStripMenuItem1.Text = "&Resize"
'
' toolStripMenuItem2
'
Me.toolStripMenuItem2.Name = "toolStripMenuItem2"
Me.toolStripMenuItem2.Text = "&Keep on Top"
'
' rearrangeButtonsToolStripMenuItem
'
Me.rearrangeButtonsToolStripMenuItem.Name = "rearrangeButtonsToolStripMenuItem"
Me.rearrangeButtonsToolStripMenuItem.Text = "R&earrange Buttons"
'
' selectIconsToolStripMenuItem
'
Me.selectIconsToolStripMenuItem.Name = "selectIconsToolStripMenuItem"
Me.selectIconsToolStripMenuItem.Text = "&Select Icons"
'
' Associate contextMenuStrip1 with Form1.
' Form1 property settings follow.
'
Me.ClientSize = New System.Drawing.Size(292, 266)
Me.ContextMenuStrip = Me.contextMenuStrip1
Me.Controls.Add(toolStrip1)
Me.Name = "Form1"
Me.toolStrip1.ResumeLayout(False)
Me.contextMenuStrip1.ResumeLayout(False)
Me.contextMenuStrip2.ResumeLayout(False)
Me.ResumeLayout(False)
Me.PerformLayout()
Me.PerformLayout()
End Sub

```

Compiling the Code

This example requires:

- References to the System, System.Data, System.Drawing and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[ContextMenuStrip](#)

[ContextMenuStrip](#)

[ToolStrip](#)

[How to: Add Menu Items to a ContextMenuStrip](#)

[ContextMenuStrip Control](#)

How to: Add Menu Items to a ContextMenuStrip

5/4/2018 • 1 min to read • [Edit Online](#)

You can add just one menu item or several items at a time to a [ContextMenuStrip](#).

To add a single menu item to a ContextMenuStrip

- Use the [Add](#) method to add one menu item to a [ContextMenuStrip](#).

```
Me.contextMenuStrip1.Items.Add(Me.toolStripMenuItem1)
```

```
this.contextMenuStrip1.Items.Add(toolStripMenuItem1);
```

To add several menu items to a ContextMenuStrip

- Use the [AddRange](#) method to add several menu items to a [ContextMenuStrip](#).

```
Me.contextMenuStrip1.Items.AddRange(New _  
System.Windows.Forms.ToolStripItem() {Me.toolStripMenuItem1, _  
Me.toolStripMenuItem2})
```

```
this.contextMenuStrip1.Items.AddRange(new  
System.Windows.Forms.ToolStripItem[] {  
this.toolStripMenuItem1, this.toolStripMenuItem2});
```

See Also

[ContextMenuStrip Control](#)

How to: Configure ContextMenuStrip Check Margins and Image Margins

5/4/2018 • 5 min to read • [Edit Online](#)

You can customize a [ContextMenuStrip](#) by setting the [ShowImageMargin](#) and [ShowCheckMargin](#) properties in various combinations.

Example

The following code example demonstrates how to set and customize the [ContextMenuStrip](#) check and image margins.

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Drawing;
```

```
Imports System
Imports System.Collections.Generic
Imports System.Windows.Forms
Imports System.Drawing
```

```
// This code example demonstrates how to set the check
// and image margins for a ToolStripMenuItem.
class Form5 : Form
{
    public Form5()
    {
        // Size the form to show three wide menu items.
        this.Width = 500;
        this.Text = "ToolStripContextMenuStrip: Image and Check Margins";

        // Create a new ToolStrip control.
        ToolStrip ms = new ToolStrip();

        // Create the ToolStripMenuItems for the ToolStrip control.
        ToolStripMenuItem bothMargins = new ToolStripMenuItem("BothMargins");
        ToolStripMenuItem imageMarginOnly = new ToolStripMenuItem("ImageMargin");
        ToolStripMenuItem checkMarginOnly = new ToolStripMenuItem("CheckMargin");
        ToolStripMenuItem noMargins = new ToolStripMenuItem("NoMargins");

        // Customize the DropDowns menus.
        // This ToolStripMenuItem has an image margin
        // and a check margin.
        bothMargins.DropDown = CreateCheckImageContextMenuStrip();
        ((ContextMenuStrip)bothMargins.DropDown).ShowImageMargin = true;
        ((ContextMenuStrip)bothMargins.DropDown).ShowCheckMargin = true;

        // This ToolStripMenuItem has only an image margin.
        imageMarginOnly.DropDown = CreateCheckImageContextMenuStrip();
        ((ContextMenuStrip)imageMarginOnly.DropDown).ShowImageMargin = true;
        ((ContextMenuStrip)imageMarginOnly.DropDown).ShowCheckMargin = false;

        // This ToolStripMenuItem has only a check margin.
        checkMarginOnly.DropDown = CreateCheckImageContextMenuStrip();
        ((ContextMenuStrip)checkMarginOnly.DropDown).ShowImageMargin = false;
```

```

    ((ContextMenuStrip)checkMarginOnly.DropDown).ShowCheckMargin = true;

    // This ToolStripMenuItem has no image and no check margin.
    noMargins.DropDown = CreateCheckImageContextMenuStrip();
    ((ContextMenuStrip)noMargins.DropDown).ShowImageMargin = false;
    ((ContextMenuStrip)noMargins.DropDown).ShowCheckMargin = false;

    // Populate the MenuStrip control with the ToolStripMenuItems.
    ms.Items.Add(bothMargins);
    ms.Items.Add(imageMarginOnly);
    ms.Items.Add(checkMarginOnly);
    ms.Items.Add(noMargins);

    // Dock the MenuStrip control to the top of the form.
    ms.Dock = DockStyle.Top;

    // Add the MenuStrip control to the controls collection last.
    // This is important for correct placement in the z-order.
    this.Controls.Add(ms);
}

// This utility method creates a Bitmap for use in
// a ToolStripMenuItem's image margin.
internal Bitmap CreateSampleBitmap()
{
    // The Bitmap is a smiley face.
    Bitmap sampleBitmap = new Bitmap(32, 32);
    Graphics g = Graphics.FromImage(sampleBitmap);

    using (Pen p = new Pen(ProfessionalColors.ButtonPressedBorder))
    {
        // Set the Pen width.
        p.Width = 4;

        // Set up the mouth geometry.
        Point[] curvePoints = new Point[]{
            new Point(4,14),
            new Point(16,24),
            new Point(28,14)};

        // Draw the mouth.
        g.DrawCurve(p, curvePoints);

        // Draw the eyes.
        g.DrawEllipse(p, new Rectangle(new Point(7, 4), new Size(3, 3)));
        g.DrawEllipse(p, new Rectangle(new Point(22, 4), new Size(3, 3)));
    }

    return sampleBitmap;
}

// This utility method creates a ContextMenuStrip control
// that has four ToolStripMenuItems showing the four
// possible combinations of image and check margins.
internal ContextMenuStrip CreateCheckImageContextMenuStrip()
{
    // Create a new ContextMenuStrip control.
    ContextMenuStrip checkImageContextMenuStrip = new ContextMenuStrip();

    // Create a ToolStripMenuItem with a
    // check margin and an image margin.
    ToolStripMenuItem yesCheckYesImage =
        new ToolStripMenuItem("Check, Image");
    yesCheckYesImage.Checked = true;
    yesCheckYesImage.Image = CreateSampleBitmap();

    // Create a ToolStripMenuItem with no
    // check margin and with an image margin.
    ToolStripMenuItem noCheckYesImage =

```

```

    ToolStripMenuItem noCheckYesImage =
        new ToolStripMenuItem("No Check, Image");
    noCheckYesImage.Checked = false;
    noCheckYesImage.Image = CreateSampleBitmap();

    // Create a ToolStripMenuItem with a
    // check margin and without an image margin.
    ToolStripMenuItem yesCheckNoImage =
        new ToolStripMenuItem("Check, No Image");
    yesCheckNoImage.Checked = true;

    // Create a ToolStripMenuItem with no
    // check margin and no image margin.
    ToolStripMenuItem noCheckNoImage =
        new ToolStripMenuItem("No Check, No Image");
    noCheckNoImage.Checked = false;

    // Add the ToolStripMenuItems to the ContextMenuStrip control.
    checkImageContextMenuStrip.Items.Add(yesCheckYesImage);
    checkImageContextMenuStrip.Items.Add(noCheckYesImage);
    checkImageContextMenuStrip.Items.Add(yesCheckNoImage);
    checkImageContextMenuStrip.Items.Add(noCheckNoImage);

    return checkImageContextMenuStrip;
}
}

```

```

' This code example demonstrates how to set the check
' and image margins for a ToolStripMenuItem.

Class Form5
    Inherits Form

    Public Sub New()
        ' Size the form to show three wide menu items.
        Me.Width = 500
        Me.Text = "ToolStripContextMenuStrip: Image and Check Margins"

        ' Create a new MenuStrip control.
        Dim ms As New MenuStrip()

        ' Create the ToolStripMenuItems for the MenuStrip control.
        Dim bothMargins As New ToolStripMenuItem("BothMargins")
        Dim imageMarginOnly As New ToolStripMenuItem("ImageMargin")
        Dim checkMarginOnly As New ToolStripMenuItem("CheckMargin")
        Dim noMargins As New ToolStripMenuItem("NoMargins")

        ' Customize the DropDowns menus.
        ' This ToolStripMenuItem has an image margin
        ' and a check margin.
        bothMargins.DropDown = CreateCheckImageContextMenuStrip()
        CType(bothMargins.DropDown, ContextMenuStrip).ShowImageMargin = True
        CType(bothMargins.DropDown, ContextMenuStrip).ShowCheckMargin = True

        ' This ToolStripMenuItem has only an image margin.
        imageMarginOnly.DropDown = CreateCheckImageContextMenuStrip()
        CType(imageMarginOnly.DropDown, ContextMenuStrip).ShowImageMargin = True
        CType(imageMarginOnly.DropDown, ContextMenuStrip).ShowCheckMargin = False

        ' This ToolStripMenuItem has only a check margin.
        checkMarginOnly.DropDown = CreateCheckImageContextMenuStrip()
        CType(checkMarginOnly.DropDown, ContextMenuStrip).ShowImageMargin = False
        CType(checkMarginOnly.DropDown, ContextMenuStrip).ShowCheckMargin = True

        ' This ToolStripMenuItem has no image and no check margin.
        noMargins.DropDown = CreateCheckImageContextMenuStrip()
        CType(noMargins.DropDown, ContextMenuStrip).ShowImageMargin = False
        CType(noMargins.DropDown, ContextMenuStrip).ShowCheckMargin = False
    End Sub

```

```

' Populate the MenuStrip control with the ToolStripMenuItems.
ms.Items.Add(bothMargins)
ms.Items.Add(imageMarginOnly)
ms.Items.Add(checkMarginOnly)
ms.Items.Add(noMargins)

' Dock the MenuStrip control to the top of the form.
ms.Dock = DockStyle.Top

' Add the MenuStrip control to the controls collection last.
' This is important for correct placement in the z-order.
Me.Controls.Add(ms)
End Sub

' This utility method creates a Bitmap for use in
' a ToolStripMenuItem's image margin.
Friend Function CreateSampleBitmap() As Bitmap

    ' The Bitmap is a smiley face.
    Dim sampleBitmap As New Bitmap(32, 32)
    Dim g As Graphics = Graphics.FromImage(sampleBitmap)

    Dim p As New Pen(ProfessionalColors.ButtonPressedBorder)
    Try
        ' Set the Pen width.
        p.Width = 4

        ' Set up the mouth geometry.
        Dim curvePoints() As Point = _
        {New Point(4, 14), New Point(16, 24), New Point(28, 14)}

        ' Draw the mouth.
        g.DrawCurve(p, curvePoints)

        ' Draw the eyes.
        g.DrawEllipse(p, New Rectangle(New Point(7, 4), New Size(3, 3)))
        g.DrawEllipse(p, New Rectangle(New Point(22, 4), New Size(3, 3)))
    Finally
        p.Dispose()
    End Try

    Return sampleBitmap
End Function

' This utility method creates a ContextMenuStrip control
' that has four ToolStripMenuItems showing the four
' possible combinations of image and check margins.
Friend Function CreateCheckImageContextMenuStrip() As ContextMenuStrip
    ' Create a new ContextMenuStrip control.
    Dim checkImageContextMenuStrip As New ContextMenuStrip()

    ' Create a ToolStripMenuItem with a
    ' check margin and an image margin.
    Dim yesCheckYesImage As New ToolStripMenuItem("Check, Image")
    yesCheckYesImage.Checked = True
    yesCheckYesImage.Image = CreateSampleBitmap()

    ' Create a ToolStripMenuItem with no
    ' check margin and with an image margin.
    Dim noCheckYesImage As New ToolStripMenuItem("No Check, Image")
    noCheckYesImage.Checked = False
    noCheckYesImage.Image = CreateSampleBitmap()

    ' Create a ToolStripMenuItem with a
    ' check margin and without an image margin.
    Dim yesCheckNoImage As New ToolStripMenuItem("Check, No Image")
    yesCheckNoImage.Checked = True
    yesCheckNoImage.Image = CreateSampleBitmap()

    ' Create a ToolStripMenuItem with no
    ' check margin and without an image margin.
    Dim noCheckNoImage As New ToolStripMenuItem("No Check, No Image")
    noCheckNoImage.Checked = False
    noCheckNoImage.Image = CreateSampleBitmap()

```

```
    ' Create a ToolStripMenuItem with no
    ' check margin and no image margin.
    Dim noCheckNoImage As New ToolStripMenuItem("No Check, No Image")
    noCheckNoImage.Checked = False

    ' Add the ToolStripMenuItems to the ContextMenuStrip control.
    checkImageContextMenuStrip.Items.Add(yesCheckYesImage)
    checkImageContextMenuStrip.Items.Add(noCheckYesImage)
    checkImageContextMenuStrip.Items.Add(yesCheckNoImage)
    checkImageContextMenuStrip.Items.Add(noCheckNoImage)

    Return checkImageContextMenuStrip
End Function
End Class
```

Compiling the Code

This example requires:

- References to the System.Design, System.Drawing, and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[ContextMenuStrip](#)

[ToolStripDropDown](#)

[ToolStrip Control](#)

[How to: Enable Check Margins and Image Margins in ContextMenuStrip Controls](#)

How to: Enable Check Margins and Image Margins in ContextMenuStrip Controls

5/4/2018 • 5 min to read • [Edit Online](#)

You can customize the [ToolStripMenuItem](#) objects in your [MenuStrip](#) control with check marks and custom images.

Example

The following code example demonstrates how to create menu items that have check marks and custom images.

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Drawing;
```

```
Imports System
Imports System.Collections.Generic
Imports System.Windows.Forms
Imports System.Drawing
```

```
// This code example demonstrates how to set the check
// and image margins for a ToolStripMenuItem.
class Form5 : Form
{
    public Form5()
    {
        // Size the form to show three wide menu items.
        this.Width = 500;
        this.Text = "ToolStripContextMenuStrip: Image and Check Margins";

        // Create a new MenuStrip control.
        MenuStrip ms = new MenuStrip();

        // Create the ToolStripMenuItems for the MenuStrip control.
        ToolStripMenuItem bothMargins = new ToolStripMenuItem("BothMargins");
        ToolStripMenuItem imageMarginOnly = new ToolStripMenuItem("ImageMargin");
        ToolStripMenuItem checkMarginOnly = new ToolStripMenuItem("CheckMargin");
        ToolStripMenuItem noMargins = new ToolStripMenuItem("NoMargins");

        // Customize the DropDowns menus.
        // This ToolStripMenuItem has an image margin
        // and a check margin.
        bothMargins.DropDown = CreateCheckImageContextMenuStrip();
        ((ContextMenuStrip)bothMargins.DropDown).ShowImageMargin = true;
        ((ContextMenuStrip)bothMargins.DropDown).ShowCheckMargin = true;

        // This ToolStripMenuItem has only an image margin.
        imageMarginOnly.DropDown = CreateCheckImageContextMenuStrip();
        ((ContextMenuStrip)imageMarginOnly.DropDown).ShowImageMargin = true;
        ((ContextMenuStrip)imageMarginOnly.DropDown).ShowCheckMargin = false;

        // This ToolStripMenuItem has only a check margin.
        checkMarginOnly.DropDown = CreateCheckImageContextMenuStrip();
        ((ContextMenuStrip)checkMarginOnly.DropDown).ShowImageMargin = false;
        ((ContextMenuStrip)checkMarginOnly.DropDown).ShowCheckMargin = true;
```

```

// This ToolStripMenuItem has no image and no check margin.
noMargins.DropDown = CreateCheckImageContextMenuStrip();
((ContextMenuStrip)noMargins.DropDown).ShowImageMargin = false;
((ContextMenuStrip)noMargins.DropDown).ShowCheckMargin = false;

// Populate the MenuStrip control with the ToolStripMenuItems.
ms.Items.Add(bothMargins);
ms.Items.Add(imageMarginOnly);
ms.Items.Add(checkMarginOnly);
ms.Items.Add(noMargins);

// Dock the MenuStrip control to the top of the form.
ms.Dock = DockStyle.Top;

// Add the MenuStrip control to the controls collection last.
// This is important for correct placement in the z-order.
this.Controls.Add(ms);
}

// This utility method creates a Bitmap for use in
// a ToolStripMenuItem's image margin.
internal Bitmap CreateSampleBitmap()
{
    // The Bitmap is a smiley face.
    Bitmap sampleBitmap = new Bitmap(32, 32);
    Graphics g = Graphics.FromImage(sampleBitmap);

    using (Pen p = new Pen(ProfessionalColors.ButtonPressedBorder))
    {
        // Set the Pen width.
        p.Width = 4;

        // Set up the mouth geometry.
        Point[] curvePoints = new Point[]{
            new Point(4,14),
            new Point(16,24),
            new Point(28,14)};

        // Draw the mouth.
        g.DrawCurve(p, curvePoints);

        // Draw the eyes.
        g.DrawEllipse(p, new Rectangle(new Point(7, 4), new Size(3, 3)));
        g.DrawEllipse(p, new Rectangle(new Point(22, 4), new Size(3, 3)));
    }

    return sampleBitmap;
}

// This utility method creates a ContextMenuStrip control
// that has four ToolStripMenuItems showing the four
// possible combinations of image and check margins.
internal ContextMenuStrip CreateCheckImageContextMenuStrip()
{
    // Create a new ContextMenuStrip control.
    ContextMenuStrip checkImageContextMenuStrip = new ContextMenuStrip();

    // Create a ToolStripMenuItem with a
    // check margin and an image margin.
    ToolStripMenuItem yesCheckYesImage =
        new ToolStripMenuItem("Check, Image");
    yesCheckYesImage.Checked = true;
    yesCheckYesImage.Image = CreateSampleBitmap();

    // Create a ToolStripMenuItem with no
    // check margin and with an image margin.
    ToolStripMenuItem noCheckYesImage =
        new ToolStripMenuItem("No Check, Image");
}

```

```

noCheckYesImage.Checked = false;
noCheckYesImage.Image = CreateSampleBitmap();

// Create a ToolStripMenuItem with a
// check margin and without an image margin.
ToolStripMenuItem yesCheckNoImage =
    new ToolStripMenuItem("Check, No Image");
yesCheckNoImage.Checked = true;

// Create a ToolStripMenuItem with no
// check margin and no image margin.
ToolStripMenuItem noCheckNoImage =
    new ToolStripMenuItem("No Check, No Image");
noCheckNoImage.Checked = false;

// Add the ToolStripMenuItems to the ContextMenuStrip control.
checkImageContextMenuStrip.Items.Add(yesCheckYesImage);
checkImageContextMenuStrip.Items.Add(noCheckYesImage);
checkImageContextMenuStrip.Items.Add(yesCheckNoImage);
checkImageContextMenuStrip.Items.Add(noCheckNoImage);

return checkImageContextMenuStrip;
}
}

```

```

' This code example demonstrates how to set the check
' and image margins for a ToolStripMenuItem.

Class Form5
    Inherits Form

    Public Sub New()
        ' Size the form to show three wide menu items.
        Me.Width = 500
        Me.Text = "ToolStripContextMenuStrip: Image and Check Margins"

        ' Create a new MenuStrip control.
        Dim ms As New MenuStrip()

        ' Create the ToolStripMenuItems for the MenuStrip control.
        Dim bothMargins As New ToolStripMenuItem("BothMargins")
        Dim imageMarginOnly As New ToolStripMenuItem("ImageMargin")
        Dim checkMarginOnly As New ToolStripMenuItem("CheckMargin")
        Dim noMargins As New ToolStripMenuItem("NoMargins")

        ' Customize the DropDowns menus.
        ' This ToolStripMenuItem has an image margin
        ' and a check margin.
        bothMargins.DropDown = CreateCheckImageContextMenuStrip()
        CType(bothMargins.DropDown, ContextMenuStrip).ShowImageMargin = True
        CType(bothMargins.DropDown, ContextMenuStrip).ShowCheckMargin = True

        ' This ToolStripMenuItem has only an image margin.
        imageMarginOnly.DropDown = CreateCheckImageContextMenuStrip()
        CType(imageMarginOnly.DropDown, ContextMenuStrip).ShowImageMargin = True
        CType(imageMarginOnly.DropDown, ContextMenuStrip).ShowCheckMargin = False

        ' This ToolStripMenuItem has only a check margin.
        checkMarginOnly.DropDown = CreateCheckImageContextMenuStrip()
        CType(checkMarginOnly.DropDown, ContextMenuStrip).ShowImageMargin = False
        CType(checkMarginOnly.DropDown, ContextMenuStrip).ShowCheckMargin = True

        ' This ToolStripMenuItem has no image and no check margin.
        noMargins.DropDown = CreateCheckImageContextMenuStrip()
        CType(noMargins.DropDown, ContextMenuStrip).ShowImageMargin = False
        CType(noMargins.DropDown, ContextMenuStrip).ShowCheckMargin = False

        ' Populate the MenuStrip control with the ToolStripMenuItems.
    End Sub

```

```

        ms.Items.Add(bothMargins)
        ms.Items.Add(imageMarginOnly)
        ms.Items.Add(checkMarginOnly)
        ms.Items.Add(noMargins)

        ' Dock the MenuStrip control to the top of the form.
        ms.Dock = DockStyle.Top

        ' Add the MenuStrip control to the controls collection last.
        ' This is important for correct placement in the z-order.
        Me.Controls.Add(ms)
End Sub

' This utility method creates a Bitmap for use in
' a ToolStripMenuItem's image margin.
Friend Function CreateSampleBitmap() As Bitmap

    ' The Bitmap is a smiley face.
    Dim sampleBitmap As New Bitmap(32, 32)
    Dim g As Graphics = Graphics.FromImage(sampleBitmap)

    Dim p As New Pen(ProfessionalColors.ButtonPressedBorder)
    Try
        ' Set the Pen width.
        p.Width = 4

        ' Set up the mouth geometry.
        Dim curvePoints() As Point = _
        {New Point(4, 14), New Point(16, 24), New Point(28, 14)}

        ' Draw the mouth.
        g.DrawCurve(p, curvePoints)

        ' Draw the eyes.
        g.DrawEllipse(p, New Rectangle(New Point(7, 4), New Size(3, 3)))
        g.DrawEllipse(p, New Rectangle(New Point(22, 4), New Size(3, 3)))
    Finally
        p.Dispose()
    End Try

    Return sampleBitmap
End Function

' This utility method creates a ContextMenuStrip control
' that has four ToolStripMenuItems showing the four
' possible combinations of image and check margins.
Friend Function CreateCheckImageContextMenuStrip() As ContextMenuStrip
    ' Create a new ContextMenuStrip control.
    Dim checkImageContextMenuStrip As New ContextMenuStrip()

    ' Create a ToolStripMenuItem with a
    ' check margin and an image margin.
    Dim yesCheckYesImage As New ToolStripMenuItem("Check, Image")
    yesCheckYesImage.Checked = True
    yesCheckYesImage.Image = CreateSampleBitmap()

    ' Create a ToolStripMenuItem with no
    ' check margin and with an image margin.
    Dim noCheckYesImage As New ToolStripMenuItem("No Check, Image")
    noCheckYesImage.Checked = False
    noCheckYesImage.Image = CreateSampleBitmap()

    ' Create a ToolStripMenuItem with a
    ' check margin and without an image margin.
    Dim yesCheckNoImage As New ToolStripMenuItem("Check, No Image")
    yesCheckNoImage.Checked = True

    ' Create a ToolStripMenuItem with no
    ' check margin and no image margin.

```

```
Dim noCheckNoImage As New ToolStripMenuItem("No Check, No Image")
noCheckNoImage.Checked = False

' Add the ToolStripMenuItems to the ContextMenuStrip control.
checkImageContextMenuStrip.Items.Add(yesCheckYesImage)
checkImageContextMenuStrip.Items.Add(noCheckYesImage)
checkImageContextMenuStrip.Items.Add(yesCheckNoImage)
checkImageContextMenuStrip.Items.Add(noCheckNoImage)

Return checkImageContextMenuStrip
End Function
End Class
```

Set the [ToolStripDropDownMenu.ShowCheckMargin](#) and [ToolStripDropDownMenu.ShowImageMargin](#) properties to specify when check marks and custom images appear in your menu items.

Compiling the Code

This example requires:

- References to the System.Design, System.Drawing, and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[ToolStripMenuItem](#)

[ToolStripDropDownMenu](#)

[MenuStrip](#)

[ToolStrip](#)

[ToolStrip Control](#)

How to: Handle the ContextMenuStrip Opening Event

5/4/2018 • 1 min to read • [Edit Online](#)

You can customize the behavior of your [ContextMenuStrip](#) control by handling the [Opening](#) event.

Example

The following code example demonstrates how to handle the [Opening](#) event. The event handler adds items dynamically to a [ContextMenuStrip](#) control. For the complete code example, see [How to: Add ToolStrip Items Dynamically](#).

```
// This event handler is invoked when the ContextMenuStrip
// control's Opening event is raised. It demonstrates
// dynamic item addition and dynamic SourceControl
// determination with reuse.
void cms_Opening(object sender, System.ComponentModel.CancelEventArgs e)
{
    // Acquire references to the owning control and item.
    Control c = fruitContextMenuStrip.SourceControl as Control;
    ToolStripDropDownItem tsi = fruitContextMenuStrip.OwnerItem as ToolStripDropDownItem;

    // Clear the ContextMenuStrip control's Items collection.
    fruitContextMenuStrip.Items.Clear();

    // Check the source control first.
    if (c != null)
    {
        // Add custom item (Form)
        fruitContextMenuStrip.Items.Add("Source: " + c.GetType().ToString());
    }
    else if (tsi != null)
    {
        // Add custom item (ToolStripDropDownButton or ToolStripMenuItem)
        fruitContextMenuStrip.Items.Add("Source: " + tsi.GetType().ToString());
    }

    // Populate the ContextMenuStrip control with its default items.
    fruitContextMenuStrip.Items.Add("-");
    fruitContextMenuStrip.Items.Add("Apples");
    fruitContextMenuStrip.Items.Add("Oranges");
    fruitContextMenuStrip.Items.Add("Pears");

    // Set Cancel to false.
    // It is optimized to true based on empty entry.
    e.Cancel = false;
}
```

```

' This event handler is invoked when the ContextMenuStrip
' control's Opening event is raised. It demonstrates
' dynamic item addition and dynamic SourceControl
' determination with reuse.
Sub cms_Opening(ByVal sender As Object, ByVal e As System.ComponentModel.CancelEventArgs)

    ' Acquire references to the owning control and item.
    Dim c As Control = fruitContextMenuStrip.SourceControl
    Dim tsi As ToolStripDropDownItem = fruitContextMenuStrip.OwnerItem

    ' Clear the ContextMenuStrip control's
    ' Items collection.
    fruitContextMenuStrip.Items.Clear()

    ' Check the source control first.
    If (c IsNot Nothing) Then
        ' Add custom item (Form)
        fruitContextMenuStrip.Items.Add(("Source: " + c.GetType().ToString()))
    ElseIf (tsi IsNot Nothing) Then
        ' Add custom item (ToolStripDropDownButton or ToolStripMenuItem)
        fruitContextMenuStrip.Items.Add(("Source: " + tsi.GetType().ToString()))
    End If

    ' Populate the ContextMenuStrip control with its default items.
    fruitContextMenuStrip.Items.Add("-")
    fruitContextMenuStrip.Items.Add("Apples")
    fruitContextMenuStrip.Items.Add("Oranges")
    fruitContextMenuStrip.Items.Add("Pears")

    ' Set Cancel to false.
    ' It is optimized to true based on empty entry.
    e.Cancel = False
End Sub

```

Set the `CancelEventArgs.Cancel` property to `true` to prevent the menu from opening.

See Also

[ContextMenuStrip](#)

[Cancel](#)

[ToolStripDropDown](#)

[ToolStrip Control](#)

DataGrid Control (Windows Forms)

5/4/2018 • 2 min to read • [Edit Online](#)

NOTE

The [DataGridView](#) control replaces and adds functionality to the [DataGrid](#) control; however, the [DataGrid](#) control is retained for both backward compatibility and future use, if you choose. For more information, see [Differences Between the Windows Forms DataGridView and DataGrid Controls](#).

The Windows Forms [DataGrid](#) control provides a user interface to ADO.NET datasets, displaying tabular data and enabling updates to the data source.

When the [DataGrid](#) control is set to a valid data source, the control is automatically populated, creating columns and rows based on the shape of the data. The [DataGrid](#) control can be used to display either a single table or the hierarchical relationships between a set of tables.

In This Section

[DataGrid Control Overview](#)

Describes the basic features of the [DataGrid](#) control.

[How to: Add Tables and Columns to the Windows Forms DataGrid Control Using the Designer](#)

Describes how to add tables and columns to the [DataGrid](#) control using the designer.

[How to: Add Tables and Columns to the Windows Forms DataGrid Control](#)

Describes how to add tables and columns to the [DataGrid](#) control programmatically.

[How to: Bind the Windows Forms DataGrid Control to a Data Source Using the Designer](#)

Describes how to bind an ADO.NET dataset to the [DataGrid](#) control using the designer.

[How to: Bind the Windows Forms DataGrid Control to a Data Source](#)

Describes how to bind an ADO.NET dataset to the [DataGrid](#) control.

[How to: Change Displayed Data at Run Time in the Windows Forms DataGrid Control](#)

Describes how to change data programmatically in the [DataGrid](#) control.

[How to: Create Master-Details Lists with the Windows Forms DataGrid Control Using the Designer](#)

Describes how to display two tables, tied together with a parent/child relationship, in two separate [DataGrid](#) controls using the designer.

[How to: Create Master-Details Lists with the Windows Forms DataGrid Control](#)

Describes how to display two tables, tied together with a parent/child relationship, in two separate [DataGrid](#) controls.

[How to: Delete or Hide Columns in the Windows Forms DataGrid Control](#)

Describes how to remove columns in the [DataGrid](#) control.

[How to: Format the Windows Forms DataGrid Control Using the Designer](#)

Describes how to change the appearance-related properties of the [DataGrid](#) control using the designer.

[How to: Format the Windows Forms DataGrid Control](#)

Describes how to change the appearance-related properties of the [DataGrid](#) control.

[Keyboard Shortcuts for the Windows Forms DataGrid Control](#)

Lists shortcuts for navigating through the `DataGrid` control.

[How to: Respond to Clicks in the Windows Forms DataGrid Control](#)

Describes how to determine which cell a user has clicked in the `DataGrid` control.

[How to: Validate Input with the Windows Forms DataGrid Control](#)

Describes how to validate input in the dataset bound to the `DataGrid` control.

Reference

[DataGrid](#)

Provides an overview of the `DataGrid` class.

[DataSource](#)

Provides details about using this property to bind the `DataGrid` control to data.

Related Sections

[Windows Forms Data Binding](#)

Provides links to topics on data binding in Windows Forms.

See Also

[DataGridView Control](#)

[Differences Between the Windows Forms DataGridView and DataGrid Controls](#)

DataGrid Control Overview (Windows Forms)

5/4/2018 • 9 min to read • [Edit Online](#)

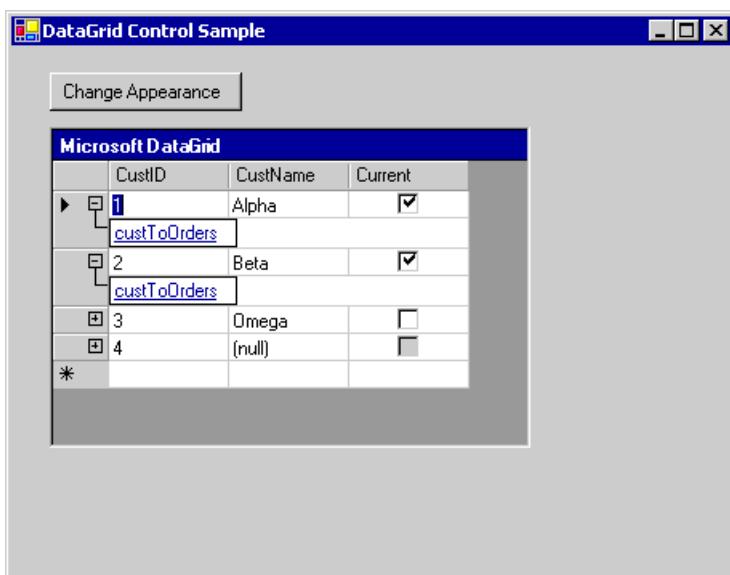
NOTE

The [DataGridView](#) control replaces and adds functionality to the [DataGrid](#) control; however, the [DataGrid](#) control is retained for both backward compatibility and future use, if you choose. For more information, see [Differences Between the Windows Forms DataGridView and DataGrid Controls](#).

The Windows Forms [DataGrid](#) control displays data in a series of rows and columns. The simplest case is when the grid is bound to a data source with a single table that contains no relationships. In that case, the data appears in simple rows and columns, as in a spreadsheet. For more information about binding data to other controls, see [Data Binding and Windows Forms](#).

If the [DataGrid](#) is bound to data with multiple related tables, and if navigation is enabled on the grid, the grid will display expanders in each row. With an expander, the user can move from a parent table to a child table. Clicking a node displays the child table, and clicking a back button displays the original parent table. In this manner, the grid displays the hierarchical relationships between tables.

The following screen shot shows a DataGrid bound to data with multiple tables.



A DataGrid bound to data with multiple tables

The [DataGrid](#) can provide a user interface for a dataset, navigation between related tables, and rich formatting and editing capabilities.

The display and manipulation of data are separate functions: The control handles the user interface, whereas data updates are handled by the Windows Forms data-binding architecture and by .NET Framework data providers. Therefore, multiple controls bound to the same data source will stay in sync.

NOTE

If you are familiar with the DataGrid control in Visual Basic 6.0, you will find some significant differences in the Windows Forms [DataGrid](#) control.

When the grid is bound to a [DataSet](#), the columns and rows are automatically created, formatted, and filled. For

more information, see [Data Binding and Windows Forms](#). Following the generation of the **DataGrid** control, you can add, delete, rearrange, and format columns and rows depending on your needs.

Binding Data to the Control

For the **DataGrid** control to work, it should be bound to a data source using the **DataSource** and **DataMember** properties at design time or the **SetDataBinding** method at run time. This binding points the **DataGrid** to an instantiated data-source object, such as a [DataSet](#) or [DataTable](#)). The **DataGrid** control shows the results of actions that are performed on the data. Most data-specific actions are not performed through the **DataGrid**, but instead through the data source.

If the data in the bound dataset is updated through any mechanism, the **DataGrid** control reflects the changes. If the data grid and its table styles and column styles have the **ReadOnly** property set to **false**, the data in the dataset can be updated through the **DataGrid** control.

Only one table can be shown in the **DataGrid** at a time. If a parent-child relationship is defined between tables, the user can move between the related tables to select the table to be displayed in the **DataGrid** control. For information about binding a **DataGrid** control to an ADO.NET data source at either design time or run time, see [How to: Bind the Windows Forms DataGrid Control to a Data Source](#).

Valid data sources for the **DataGrid** include:

- [DataTable](#) class
- [DataView](#) class
- [DataSet](#) class
- [DataViewManager](#) class

If your source is a dataset, the dataset might be an object in the form or an object passed to the form by an XML Web service. You can bind to either typed or untyped datasets.

You can also bind a **DataGrid** control to additional structures if the objects in the structure, such as the elements in an array, expose public properties. The grid will display all the public properties of the elements in the structure. For example, if you bind the **DataGrid** control to an array of customer objects, the grid will display all the public properties of those customer objects. In some instances, this means that although you can bind to the structure, the resulting bound structure might not have practical application. For example, you can bind to an array of integers, but because the **Integer** data type does not support a public property, the grid cannot display any data.

You can bind to the following structures if their elements expose public properties:

- Any component that implements the [IList](#) interface. This includes single-dimension arrays.
- Any component that implements the [IListSource](#) interface.
- Any component that implements the [IBindingList](#) interface.

For more information about possible data sources, see [Data Sources Supported by Windows Forms](#).

Grid Display

A common use of the **DataGrid** control is to display a single table of data from a dataset. However, the control can also be used to display multiple tables, including related tables. The display of the grid is adjusted automatically according to the data source. The following table shows what is displayed for various configurations.

CONTENTS OF DATA SET	WHAT IS DISPLAYED
Single table.	Table is displayed in a grid.
Multiple tables.	The grid can display a tree view that users can navigate to locate the table they want to display.
Multiple related tables.	The grid can display a tree view to select tables with, or you can specify that the grid display the parent table. Records in the parent table let users navigate to related child rows.

NOTE

Tables in a dataset are related using a [DataRelation](#). Also see [HYPERLINK "http://msdn.microsoft.com/library/dbwcse3d\(v=vs.110\)" Relationships in Datasets](#) or [Relationships in Datasets](#).

When the [DataGrid](#) control is displaying a table and the [AllowSorting](#) property is set to `true`, data can be resorted by clicking the column headers. The user can also add rows and edit cells.

The relationships between a set of tables are displayed to users by using a parent/child structure of navigation. Parent tables are the highest level of data, and child tables are those data tables that are derived from the individual listings in the parent tables. Expanders are displayed in each parent row that contains a child table. Clicking an expander generates a list of Web-like links to the child tables. When the user selects a link, the child table is displayed. Clicking the show/hide parent rows icon () will hide the information about the parent table or cause it to reappear if the user has previously hidden it. The user can click a back button to move back to the previously viewed table.

Columns and Rows

The [DataGrid](#) consists of a collection of [DataGridTableStyle](#) objects that are contained in the [DataGrid](#) control's [TableStyles](#) property. A table style may contain a collection of [DataGridColumnStyle](#) objects that are contained in the [GridColumnStyles](#) property of the [DataGridTableStyle](#). You can edit the [TableStyles](#) and [GridColumnStyles](#) properties by using collection editors accessed through the [Properties](#) window.

Any [DataGridTableStyle](#) associated with the [DataGrid](#) control can be accessed through the [GridTableStylesCollection](#). The [GridTableStylesCollection](#) can be edited in the designer with the [DataGridTableStyle](#) collection editor, or programmatically through the [DataGrid](#) control's [TableStyles](#) property.



The following illustration shows the objects included in the DataGrid control.

Table styles and column styles are synchronized with [DataTable](#) objects and [DataColumn](#) objects by setting their [MappingName](#) properties to the appropriate [TableName](#) and [ColumnName](#) properties. When a [DataGridTableStyle](#) that has no column styles is added to a [DataGrid](#) control bound to a valid data source, and the [MappingName](#) property of that table style is set to a valid [TableName](#) property, a collection of [DataGridColumnStyle](#) objects is created for that table style. For each [DataColumn](#) found in the [Columns](#) collection of the [DataTable](#), a corresponding [DataGridColumnStyle](#) is added to the [GridColumnStylesCollection](#). [GridColumnStylesCollection](#) is accessed through the [GridColumnStyles](#) property of the [DataGridTableStyle](#). Columns can be added or deleted from the grid using the [Add](#) or [Remove](#) method on the [GridColumnStylesCollection](#). For more information, see [How to: Add Tables and Columns to the Windows Forms DataGrid Control](#) and [How to: Delete or Hide Columns in the Windows Forms DataGrid Control](#).

A collection of column types extends the [DataGridColumnStyle](#) class with rich formatting and editing capabilities. All column types inherit from the [DataGridColumnStyle](#) base class. The class that is created depends on the [DataType](#) property of the [DataTable](#) from which the [DataColumn](#) is based. For example, a [DataColumn](#) that has its [DataType](#) property set to [Boolean](#) will be associated with the [DataGridBoolColumn](#). The following table describes each of these column types.

COLUMN TYPE	DESCRIPTION
DataGridTextBoxColumn	Accepts and displays data as formatted or unformatted strings. Editing capabilities are the same as they are for editing data in a simple TextBox . Inherits from DataGridColumnStyle .
DataGridBoolColumn	Accepts and displays <code>true</code> , <code>false</code> , and null values. Inherits from DataGridColumnStyle .

Double-clicking the right edge of a column resizes the column to display its full caption and widest entry.

Table Styles and Column Styles

As soon as you have established the default format of the [DataGrid](#) control, you can customize the colors that will be used when certain tables are displayed within the data grid.

This is achieved by creating instances of the [DataGridTableStyle](#) class. Table styles specify the formatting of specific tables, distinct from the default formatting of the [DataGrid](#) control itself. Each table may have only one table style defined for it at a time.

Sometimes, you will want to have a specific column look different from the rest of the columns of a particular data table. You can create a customized set of column styles by using the [GridColumnStyles](#) property.

Column styles are related to columns in a dataset just like table styles are related to data tables. Just as each table may only have one table style defined for it at a time, so too can each column only have one column style defined for it, in a particular table style. This relationship is defined in the column's [MappingName](#) property.

If you have created a table style without column styles added to it, Visual Studio will add default column styles when the form and grid are created at run time. However, if you have created a table style and added any column styles to it, Visual Studio will not create any column styles. Also, you will need to define column styles and assign them with the mapping name to have the columns that you want appear in the grid.

Because you specify which columns are included in the data grid by assigning them a column style and no column style has been assigned to the columns, you can include columns of data in the dataset that are not displayed in the grid. However, because the data column is included in the dataset, you can programmatically edit the data that is not displayed.

NOTE

In general, create column styles and add them to the column styles collection before adding table styles to the table styles collection. When you add an empty table style to the collection, column styles are automatically generated for you.

Consequently, an exception will be thrown if you try to add new column styles with duplicate [MappingName](#) values to the column styles collection.

Sometimes, you will want to just tweak one column among many columns; for example, the dataset contains 50 columns and you only want 49 of them. In this case, it is easier to import all 50 columns and programmatically remove one, rather than programmatically adding each of the 49 individual columns you want.

Formatting

Formatting that can be applied to the [DataGrid](#) control includes border styles, gridline styles, fonts, caption properties, data alignment, and alternating background colors between rows. For more information, see [How to: Format the Windows Forms DataGrid Control](#).

Events

Besides the common control events such as [MouseDown](#), [Enter](#), and [Scroll](#), the [DataGrid](#) control supports events associated with editing and navigation within the grid. The [CurrentCell](#) property determines which cell is selected. The [CurrentCellChanged](#) event is raised when the user navigates to a new cell. When the user navigates to a new table through parent/child relations, the [Navigate](#) event is raised. The [BackButtonClick](#) event is raised when the user clicks the back button when the user is viewing a child table, and the [ShowParentDetailsButtonClick](#) event is raised when the show/hide parent rows icon is clicked.

See Also

[DataGrid Control](#)

[How to: Bind the Windows Forms DataGrid Control to a Data Source](#)

[How to: Add Tables and Columns to the Windows Forms DataGrid Control](#)

[How to: Delete or Hide Columns in the Windows Forms DataGrid Control](#)

[How to: Format the Windows Forms DataGrid Control](#)

How to: Add Tables and Columns to the Windows Forms DataGrid Control

5/4/2018 • 1 min to read • [Edit Online](#)

NOTE

The [DataGridView](#) control replaces and adds functionality to the [DataGrid](#) control; however, the [DataGrid](#) control is retained for both backward compatibility and future use, if you choose. For more information, see [Differences Between the Windows Forms DataGridView and DataGrid Controls](#).

You can display data in the Windows Forms [DataGrid](#) control in tables and columns by creating **DataGridTableStyle** objects and adding them to the **GridTableStylesCollection** object, which is accessed through the [DataGrid](#) control's **TableStyles** property. Each table style displays the contents of whatever data table is specified in the **DataGridTableStyle** object's **MappingName** property. By default, a table style with no column styles specified will display all the columns within that data table. You can restrict which columns from the table appear by adding **DataGridColumnStyle** objects to the **GridColumnStylesCollection** object, which is accessed through the **GridColumnStyles** property of each **DataGridTableStyle** object.

To add a table and column to a DataGrid programmatically

1. In order to display data in the table, you must first bind the [DataGrid](#) control to a dataset. For more information, see [How to: Bind the Windows Forms DataGrid Control to a Data Source](#).

Caution

When programmatically specifying column styles, always create **DataGridColumnStyle** objects and add them to the **GridColumnStylesCollection** object before adding **DataGridTableStyle** objects to the **GridTableStylesCollection** object. When you add an empty **DataGridTableStyle** object to the collection, **DataGridColumnStyle** objects are automatically generated for you. Consequently, an exception will be thrown if you try to add new **DataGridColumnStyle** objects with duplicate **MappingName** values to the **GridColumnStylesCollection** object.

2. Declare a new table style and set its mapping name.

```
Dim ts1 As New DataGridTableStyle()
ts1.MappingName = "Customers"
```

```
DataGridTableStyle ts1 = new DataGridTableStyle();
ts1.MappingName = "Customers";
```

```
DataGridTableStyle* ts1 = new DataGridTableStyle();
ts1->MappingName = S"Customers";
```

3. Declare a new column style and set its mapping name and other properties.

```
Dim myDataCol As New DataGridBoolColumn()
myDataCol.HeaderText = "My New Column"
myDataCol.MappingName = "Current"
```

```
DataGridBoolColumn myDataCol = new DataGridBoolColumn();
myDataCol.HeaderText = "My New Column";
myDataCol.MappingName = "Current";
```

```
DataGridBoolColumn^ myDataCol = gcnew DataGridBoolColumn();
myDataCol->HeaderText = "My New Column";
myDataCol->MappingName = "Current";
```

4. Call the **Add** method of the **GridColumnStylesCollection** object to add the column to the table style

```
ts1.GridColumnStyles.Add(myDataCol)
```

```
ts1.GridColumnStyles.Add(myDataCol);
```

```
ts1->GridColumnStyles->Add(myDataCol);
```

5. Call the **Add** method of the **GridTableStylesCollection** object to add the table style to the data grid.

```
DataGrid1.TableStyles.Add(ts1)
```

```
dataGrid1.TableStyles.Add(ts1);
```

```
dataGrid1->TableStyles->Add(ts1);
```

See Also

[DataGridView Control](#)

[How to: Delete or Hide Columns in the Windows Forms DataGridView Control](#)

How to: Add Tables and Columns to the Windows Forms DataGrid Control Using the Designer

5/4/2018 • 2 min to read • [Edit Online](#)

NOTE

The [DataGridView](#) control replaces and adds functionality to the [DataGrid](#) control; however, the [DataGrid](#) control is retained for both backward compatibility and future use, if you choose. For more information, see [Differences Between the Windows Forms DataGridView and DataGrid Controls](#).

You can display data in the Windows Forms [DataGrid](#) control in tables and columns by creating [DataGridTableStyle](#) objects and adding them to the [GridTableStylesCollection](#) object, which is accessed through the [DataGrid](#) control's [TableStyles](#) property. Each table style displays the contents of whatever data table is specified in the [MappingName](#) property of the [DataGridTableStyle](#). By default, a table style without column styles specified will display all the columns within that data table. You can restrict which columns from the table appear by adding [DataGridColumnStyle](#) objects to the [GridColumnStylesCollection](#), which is accessed through the [GridColumnStyles](#) property of each [DataGridTableStyle](#).

The following procedures require a **Windows Application** project with a form that contains a [DataGrid](#) control. For information about how to set up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#). By default in Visual Studio 2005, the [DataGrid](#) control is not in the [Toolbox](#). For information about adding it, see [How to: Add Items to the Toolbox](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To add a table to the DataGrid control in the designer

1. In order to display data in the table, you must first bind the [DataGrid](#) control to a dataset. For more information, see [How to: Bind the Windows Forms DataGrid Control to a Data Source Using the Designer](#).
2. Select the [DataGrid](#) control's [TableStyles](#) property in the Properties window, and then click the ellipsis button (next to the property to display the **DataGridTableStyle Collection Editor**.
3. In the collection editor, click **Add** to insert a table style.
4. Click **OK** to close the collection editor, and then reopen it by clicking the ellipsis button next to the [TableStyles](#) property.

When you reopen the collection editor, any data tables bound to the control will appear in the drop-down list for the [MappingName](#) property of the table style.

5. In the **Members** box of the collection editor, click the table style.
6. In the **Properties** box of the collection editor, select the [MappingName](#) value for the table you want to display.

To add a column to the DataGrid control in the designer

1. In the **Members** box of the **DataGridTableStyle Collection Editor**, select the appropriate table style. In the **Properties** box of the collection editor, select the **GridColumnStyles** collection, and then click the ellipsis button (**...**) next to the property to display the **DataGridColumnStyle Collection Editor**.
2. In the collection editor, click **Add** to insert a column style or click the down arrow next to **Add** to specify a column type.

In the drop-down box, you can select either the **DataGridTextBoxColumn** or **DataGridBoolColumn** type.

3. Click **OK** to close the **DataGridColumnStyle Collection Editor**, and then reopen it by clicking the ellipsis button next to the **GridColumnStyles** property.

When you reopen the collection editor, any data columns in the bound data table will appear in the drop-down list for the **MappingName** property of the column style.

4. In the **Members** box of the collection editor, click the column style.
5. In the **Properties** box of the collection editor, select the **MappingName** value for the column you want to display.

See Also

[DataGridView Control](#)

[How to: Delete or Hide Columns in the Windows Forms DataGridView Control](#)

How to: Bind the Windows Forms DataGrid Control to a Data Source

5/4/2018 • 2 min to read • [Edit Online](#)

NOTE

The [DataGridView](#) control replaces and adds functionality to the [DataGrid](#) control; however, the [DataGrid](#) control is retained for both backward compatibility and future use, if you choose. For more information, see [Differences Between the Windows Forms DataGridView and DataGrid Controls](#).

The Windows Forms [DataGrid](#) control is specifically designed to display information from a data source. You bind the control at run time by calling the [SetDataBinding](#) method. Although you can display data from a variety of data sources, the most typical sources are datasets and data views.

To data-bind the DataGrid control programmatically

1. Write code to fill the dataset.

If the data source is a dataset or a data view based on a dataset table, add code to the form to fill the dataset.

The exact code you use depends on where the dataset is getting data. If the dataset is being populated directly from a database, you typically call the [Fill](#) method of a data adapter, as in the following example, which populates a dataset called `DsCategories1`:

```
sqlDataAdapter1.Fill(DsCategories1)
```

```
sqlDataAdapter1.Fill(DsCategories1);
```

```
sqlDataAdapter1->Fill(dsCategories1);
```

If the dataset is being filled from an XML Web service, you typically create an instance of the service in your code and then call one of its methods to return a dataset. You then merge the dataset from the XML Web service into your local dataset. The following example shows how you can create an instance of an XML Web service called `CategoriesService`, call its `GetCategories` method, and merge the resulting dataset into a local dataset called `DsCategories1`:

```
Dim ws As New MyProject.localhost.CategoriesService()
ws.Credentials = System.Net.CredentialCache.DefaultCredentials
DsCategories1.Merge(ws.GetCategories())
```

```
MyProject.localhost.CategoriesService ws = new MyProject.localhost.CategoriesService();
ws.Credentials = System.Net.CredentialCache.DefaultCredentials;
DsCategories1.Merge(ws.GetCategories());
```

```
MyProject::localhost::CategoriesService^ ws =
    new MyProject::localhost::CategoriesService();
ws->Credentials = System::Net::CredentialCache::DefaultCredentials;
dsCategories1->Merge(ws->GetCategories());
```

2. Call the [DataGrid](#) control's [SetDataBinding](#) method, passing it the data source and a data member. If you do not need to explicitly pass a data member, pass an empty string.

NOTE

If you are binding the grid for the first time, you can set the control's [DataSource](#) and [DataMember](#) properties. However, you cannot reset these properties once they have been set. Therefore, it is recommended that you always use the [SetDataBinding](#) method.

The following example shows how you can programmatically bind to the Customers table in a dataset called `DsCustomers1`:

```
DataGrid1.SetDataBinding(DsCustomers1, "Customers")
```

```
DataGrid1.SetDataBinding(DsCustomers1, "Customers");
```

```
dataGrid1->SetDataBinding(dsCustomers1, "Customers");
```

If the Customers table is the only table in the dataset, you could alternatively bind the grid this way:

```
DataGrid1.SetDataBinding(DsCustomers1, "")
```

```
DataGrid1.SetDataBinding(DsCustomers1, "");
```

```
dataGrid1->SetDataBinding(dsCustomers1, "");
```

3. (Optional) Add the appropriate table styles and column styles to the grid. If there are no table styles, you will see the table, but with minimal formatting and with all columns visible.

See Also

[DataGrid Control Overview](#)

[How to: Add Tables and Columns to the Windows Forms DataGrid Control](#)

[DataGrid Control](#)

[Windows Forms Data Binding](#)

How to: Bind the Windows Forms DataGrid Control to a Data Source Using the Designer

5/4/2018 • 2 min to read • [Edit Online](#)

NOTE

The [DataGridView](#) control replaces and adds functionality to the [DataGrid](#) control; however, the [DataGrid](#) control is retained for both backward compatibility and future use, if you choose. For more information, see [Differences Between the Windows Forms DataGridView and DataGrid Controls](#).

The Windows Forms [DataGrid](#) control is specifically designed to display information from a data source. You bind the control at design time by setting the [DataSource](#) and [DataMember](#) properties, or at run time by calling the [SetDataBinding](#) method. Although you can display data from a variety of data sources, the most typical sources are datasets and data views.

If the data source is available at design time—for example, if the form contains an instance of a dataset or a data view—you can bind the grid to the data source at design time. You can then preview what the data will look like in the grid.

You can also bind the grid programmatically, at run time. This is useful when you want to set a data source based on information you get at run time. For example, the application might let the user specify the name of a table to view. It is also necessary in situations where the data source does not exist at design time. This includes data sources such as arrays, collections, untyped datasets, and data readers.

The following procedure requires a **Windows Application** project with a form containing a [DataGrid](#) control. For information about setting up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#). In Visual Studio 2005, the [DataGrid](#) control is not in the [Toolbox](#) by default. For information about adding it, see [How to: Add Items to the Toolbox](#). Additionally in Visual Studio 2005, you can use the [Data Sources](#) window for design-time data binding. For more information see [Bind controls to data in Visual Studio](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To data-bind the DataGrid control to a single table in the designer

1. Set the control's [DataSource](#) property to the object containing the data items you want to bind to.
2. If the data source is a dataset, set the [DataMember](#) property to the name of the table to bind to.
3. If the data source is a dataset or a data view based on a dataset table, add code to the form to fill the dataset.

The exact code you use depends on where the dataset is getting data. If the dataset is being populated directly from a database, you typically call the [Fill](#) method of a data adapter, as in the following code example, which populates a dataset called [DsCategories1](#):

```
sqlDataAdapter1.Fill(DsCategories1)
```

```
sqlDataAdapter1.Fill(DsCategories1);
```

```
sqlDataAdapter1->Fill(dsCategories1);
```

4. (Optional) Add the appropriate table styles and column styles to the grid.

If there are no table styles, you will see the table, but with minimal formatting and with all columns visible.

To data-bind the DataGrid control to multiple tables in a dataset in the designer

1. Set the control's [DataSource](#) property to the object containing the data items you want to bind to.
2. If the dataset contains related tables (that is, if it contains a relation object), set the [DataMember](#) property to the name of the parent table.
3. Write code to fill the dataset.

See Also

[DataGrid Control Overview](#)

[How to: Add Tables and Columns to the Windows Forms DataGrid Control](#)

[DataGrid Control](#)

[Windows Forms Data Binding](#)

[Accessing data in Visual Studio](#)

How to: Change Displayed Data at Run Time in the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

NOTE

The [DataGridView](#) control replaces and adds functionality to the [DataGrid](#) control; however, the [DataGrid](#) control is retained for both backward compatibility and future use, if you choose. For more information, see [Differences Between the Windows Forms DataGridView and DataGrid Controls](#).

After you have created a Windows Forms [DataGrid](#) using the design-time features, you may also wish to dynamically change elements of the [DataSet](#) object of the grid at run time. This can include changes to either individual values of the table or changing which data source is bound to the [DataGrid](#) control. Changes to individual values are done through the [DataSet](#) object, not the [DataGrid](#) control.

To change data programmatically

1. Specify the desired table from the [DataSet](#) object and the desired row and field from the table and set the cell equal to the new value.

NOTE

To specify the first table of the [DataSet](#) or the first row of the table, use 0.

The following example shows how to change the second entry of the first row of the first table of a dataset by clicking `Button1`. The [DataSet](#) (`ds`) and Tables (`0` and `1`) were previously created.

```
Protected Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    ds.Tables(0).Rows(0)(1) = "NewEntry"
End Sub
```

```
private void button1_Click(object sender, System.EventArgs e)
{
    ds.Tables[0].Rows[0][1] = "NewEntry";
}
```

```
private:
void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    dataSet1->Tables[0]->Rows[0][1] = "NewEntry";
}
```

(Visual C#, Visual C++) Place the following code in the form's constructor to register the event handler.

```
this.button1.Click += new System.EventHandler(this.button1_Click);
```

```
this->button1->Click +=  
    gcnew System::EventHandler(this, &Form1::button1_Click);
```

At run time you can use the [SetDataBinding](#) method to bind the **DataGrid** control to a different data source. For example, you may have several ADO.NET data controls, each connected to a different database.

To change the DataSource programmatically

1. Set the [SetDataBinding](#) method to the name of the data source and table you want to bind to.

The following example shows how to change the date source using the [SetDataBinding](#) method to an ADO.NET data control (`adoPubsAuthors`) that is connected to the Authors table in the Pubs database.

```
Private Sub ResetSource()  
    DataGrid1.SetDataBinding(adoPubsAuthors, "Authors")  
End Sub
```

```
private void ResetSource()  
{  
    DataGrid1.SetDataBinding(adoPubsAuthors, "Authors");  
}
```

```
private:  
    void ResetSource()  
    {  
        dataGrid1->SetDataBinding(adoPubsAuthors, "Authors");  
    }
```

See Also

[ADO.NET DataSets](#)

[How to: Delete or Hide Columns in the Windows Forms DataGrid Control](#)

[How to: Add Tables and Columns to the Windows Forms DataGrid Control](#)

[How to: Bind the Windows Forms DataGrid Control to a Data Source](#)

How to: Create Master/Detail Lists with the Windows Forms DataGrid Control

5/4/2018 • 1 min to read • [Edit Online](#)

NOTE

The [DataGridView](#) control replaces and adds functionality to the [DataGrid](#) control; however, the [DataGrid](#) control is retained for both backward compatibility and future use, if you choose. For more information, see [Differences Between the Windows Forms DataGridView and DataGrid Controls](#).

If your [DataSet](#) contains a series of related tables, you can use two [DataGrid](#) controls to display the data in a master/detail format. One [DataGrid](#) is designated to be the master grid, and the second is designated to be the details grid. When you select an entry in the master list, all of the related child entries are shown in the details list. For example, if your [DataSet](#) contains a Customers table and a related Orders table, you would specify the Customers table to be the master grid and the Orders table to be the details grid. When a customer is selected from the master grid, all of the orders associated with that customer in the Orders table would be displayed in the details grid.

To set a master/detail relationship programmatically

1. Create two new [DataGrid](#) controls and set their properties.
2. Add tables to the dataset.
3. Declare a variable of type [DataRelation](#) to represent the relation you want to create.
4. Instantiate the relationship by specifying a name for the relationship and specifying the table, column, and item that will tie the two tables.
5. Add the relationship to the [DataSet](#) object's [Relations](#) collection.
6. Use the [SetDataBinding](#) method of the [DataGrid](#) to bind each of the grids to the [DataSet](#).

The following example shows how to set a master/detail relationship between the Customers and Orders tables in a previously generated [DataSet](#) (`ds`).

```
Dim myDataRelation As DataRelation
myDataRelation = New DataRelation _
    ("CustOrd", ds.Tables("Customers").Columns("CustomerID"), _
     ds.Tables("Orders").Columns("CustomerID"))
' Add the relation to the DataSet.
ds.Relations.Add(myDataRelation)
GridOrders.SetDataBinding(ds, "Customers")
GridDetails.SetDataBinding(ds, "Customers.CustOrd")
```

```
DataRelation myDataRelation;
myDataRelation = new DataRelation("CustOrd", ds.Tables["Customers"].Columns["CustomerID"],
ds.Tables["Orders"].Columns["CustomerID"]);
// Add the relation to the DataSet.
ds.Relations.Add(myDataRelation);
GridOrders.SetDataBinding(ds,"Customers");
GridDetails.SetDataBinding(ds,"Customers.CustOrd");
```

```
DataRelation^ myDataRelation;
myDataRelation = gcnew DataRelation("CustOrd",
    ds->Tables["Customers"]->Columns["CustomerID"],
    ds->Tables["Orders"]->Columns["CustomerID"]);
// Add the relation to the DataSet.
ds->Relations->Add(myDataRelation);
GridOrders->SetDataBinding(ds, "Customers");
GridDetails->SetDataBinding(ds, "Customers.CustOrd");
```

See Also

[DataGridView Control](#)

[DataGridView Control Overview](#)

[How to: Bind the Windows Forms DataGridView Control to a Data Source](#)

How to: Create Master-Details Lists with the Windows Forms DataGrid Control Using the Designer

5/4/2018 • 2 min to read • [Edit Online](#)

NOTE

The [DataGridView](#) control replaces and adds functionality to the [DataGrid](#) control; however, the [DataGrid](#) control is retained for both backward compatibility and future use, if you choose. For more information, see [Differences Between the Windows Forms DataGridView and DataGrid Controls](#).

If your [DataSet](#) contains a series of related tables, you can use two [DataGrid](#) controls to display the data in a master-detail format. One [DataGrid](#) is designated to be the master grid, and the second is designated to be the details grid. When you select an entry in the master list, all of the related child entries are shown in the details list. For example, if your [DataSet](#) contains a Customers table and a related Orders table, you would specify the Customers table to be the master grid and the Orders table to be the details grid. When a customer is selected from the master grid, all of the orders associated with that customer in the Orders table would be displayed in the details grid.

The following procedure requires a **Windows Application** project. For information about setting up such a project, see [How to: Create a Windows Application Project](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To create a master-details list in the designer

1. Add two [DataGrid](#) controls to the form. For more information, see [How to: Add Controls to Windows Forms](#). In Visual Studio 2005, the [DataGrid](#) control is not in the **Toolbox** by default. For more information, see [How to: Add Items to the Toolbox](#).

NOTE

The following steps are not applicable to Visual Studio 2005, which uses the **Data Sources** window for design-time data binding. For more information, see [Bind controls to data in Visual Studio](#) and [How to: Display Related Data in a Windows Forms Application](#).

2. Drag two or more tables from **Server Explorer** to the form.
3. From the **Data** menu, select **Generate DataSet**.
4. Set the relationships between the tables using the XML Designer. For details, see "How to: Create One-to-Many Relationships in XML Schemas and Datasets" on MSDN.
5. Save the relationships by selecting **Save All** from the **File** menu.
6. Configure the [DataGrid](#) control that you want to designate the master grid, as follows:
 - a. Select the [DataSet](#) from the drop-down list in the [DataSource](#) property.

- b. Select the master table (for example, "Customers") from the drop-down list in the **DataMember** property.
7. Configure the **DataGrid** control that you want to designate the details grid, as follows:
 - a. Select the **DataSet** from the drop-down list in the **DataSource** property.
 - b. Select the relationship (for example, "Customers.CustOrd") between the master and detail tables from the drop-down list in the **DataMember** property. In order to see the relationship, expand the node by clicking on the plus (+) sign next to the master table in the drop-down list.

See Also

[DataGrid Control](#)

[DataGrid Control Overview](#)

[How to: Bind the Windows Forms DataGrid Control to a Data Source](#)

[Bind controls to data in Visual Studio](#)

How to: Delete or Hide Columns in the Windows Forms DataGrid Control

5/4/2018 • 2 min to read • [Edit Online](#)

NOTE

The [DataGridView](#) control replaces and adds functionality to the [DataGrid](#) control; however, the [DataGrid](#) control is retained for both backward compatibility and future use, if you choose. For more information, see [Differences Between the Windows Forms DataGridView and DataGrid Controls](#).

You can programmatically delete or hide columns in the Windows Forms [DataGrid](#) control by using the properties and methods of the [GridColumnStylesCollection](#) and [DataGridColumnStyle](#) objects (which are members of the [DataGridTableStyle](#) class).

The deleted or hidden columns still exist in the data source the grid is bound to, and can still be accessed programmatically. They are just no longer visible in the datagrid.

NOTE

If your application does not access certain columns of data, and you do not want them displayed in the datagrid, then it is probably not necessary to include them in the data source in the first place.

To delete a column from the DataGrid programmatically

1. In the declarations area of your form, declare a new instance of the [DataGridTableStyle](#) class.
2. Set the [DataGridTableStyle.MappingName](#) property to the table in your data source that you want to apply the style to. The following example uses the [DataGrid.DataMember](#) property, which it assumes is already set.
3. Add the new [DataGridTableStyle](#) object to the datagrid's table styles collection.
4. Call the [RemoveAt](#) method of the [DataGrid](#)'s [GridColumnStyles](#) collection, specifying the column index of the column to delete.

```
' Declare a new DataGridTableStyle in the
' declarations area of your form.
Dim ts As DataGridTableStyle = New DataGridTableStyle()

Sub DeleteColumn()
    ' Set the DataGridTableStyle.MappingName property
    ' to the table in the data source to map to.
    ts.MappingName = DataGrid1.DataMember

    ' Add it to the datagrid's TableStyles collection
    DataGrid1.TableStyles.Add(ts)

    ' Delete the first column (index 0)
    DataGrid1.TableStyles(0).GridColumnStyles.RemoveAt(0)
End Sub
```

```

// Declare a new DataGridTableStyle in the
// declarations area of your form.
DataGridTableStyle ts = new DataGridTableStyle();

private void deleteColumn()
{
    // Set the DataGridTableStyle.MappingName property
    // to the table in the data source to map to.
    ts.MappingName = dataGrid1.DataMember;

    // Add it to the datagrid's TableStyles collection
    dataGrid1.TableStyles.Add(ts);

    // Delete the first column (index 0)
    dataGrid1.TableStyles[0].GridColumnStyles.RemoveAt(0);
}

```

To hide a column in the DataGrid programmatically

1. In the declarations area of your form, declare a new instance of the [DataGridTableStyle](#) class.
2. Set the [MappingName](#) property of the [DataGridTableStyle](#) to the table in your data source that you want to apply the style to. The following code example uses the [DataGrid.DataMember](#) property, which it assumes is already set.
3. Add the new [DataGridTableStyle](#) object to the datagrid's table styles collection.
4. Hide the column by setting its [Width](#) property to 0, specifying the column index of the column to hide.

```

' Declare a new DataGridTableStyle in the
' declarations area of your form.
Dim ts As DataGridTableStyle = New DataGridTableStyle()

Sub HideColumn()
    ' Set the DataGridTableStyle.MappingName property
    ' to the table in the data source to map to.
    ts.MappingName = DataGrid1.DataMember

    ' Add it to the datagrid's TableStyles collection
    DataGrid1.TableStyles.Add(ts)

    ' Hide the first column (index 0)
    DataGrid1.TableStyles(0).GridColumnStyles(0).Width = 0
End Sub

```

```

// Declare a new DataGridTableStyle in the
// declarations area of your form.
DataGridTableStyle ts = new DataGridTableStyle();

private void hideColumn()
{
    // Set the DataGridTableStyle.MappingName property
    // to the table in the data source to map to.
    ts.MappingName = dataGrid1.DataMember;

    // Add it to the datagrid's TableStyles collection
    dataGrid1.TableStyles.Add(ts);

    // Hide the first column (index 0)
    dataGrid1.TableStyles[0].GridColumnStyles[0].Width = 0;
}

```

See Also

[How to: Change Displayed Data at Run Time in the Windows Forms DataGridView Control](#)

[How to: Add Tables and Columns to the Windows Forms DataGridView Control](#)

How to: Format the Windows Forms DataGridView Control

5/4/2018 • 5 min to read • [Edit Online](#)

NOTE

The [DataGridView](#) control replaces and adds functionality to the [DataGrid](#) control; however, the [DataGrid](#) control is retained for both backward compatibility and future use, if you choose. For more information, see [Differences Between the Windows Forms DataGridView and DataGrid Controls](#).

Applying different colors to various parts of a [DataGrid](#) control can help to make the information in it easier to read and interpret. Color can be applied to rows and columns. Rows and columns can also be hidden or shown at your discretion.

There are three basic aspects of formatting the [DataGrid](#) control. You can set properties to establish a default style in which data is displayed. From that base, you can then customize the way certain tables are displayed at run time. Finally, you can modify which columns are displayed in the data grid as well as the colors and other formatting that is shown.

As an initial step in formatting a data grid, you can set the properties of the [DataGrid](#) itself. These color and format choices form a base from which you can then make changes depending on the data tables and columns displayed.

To establish a default style for the DataGrid control

1. Set the following properties as appropriate:

PROPERTY	DESCRIPTION
AlternatingBackColor	The BackColor property defines the color of the even-numbered rows of the grid. When you set the AlternatingBackColor property to a different color, every other row is set to this new color (rows 1, 3, 5, and so on).
BackColor	The background color of the even-numbered rows of the grid (rows 0, 2, 4, 6, and so on).
BackgroundColor	Whereas the BackColor and AlternatingBackColor properties determines the color of rows in the grid, the BackgroundColor property determines the color of the nonrow area, which is only visible when the grid is scrolled to the bottom, or if only a few rows are contained in the grid.
BorderStyle	The grid's border style, one of the BorderStyle enumeration values.
CaptionBackColor	The background color of the grid's window caption which appears immediately above the grid.
CaptionFont	The font of the caption at the top of the grid.

PROPERTY	DESCRIPTION
CaptionForeColor	The background color of the grid's window caption.
Font	The font used to display the text in the grid.
ForeColor	The color of the font displayed by the data in the rows of the data grid.
GridLineColor	The color of the grid lines of the data grid.
GridLineStyle	The style of the lines separating the cells of the grid, one of the DataGridLineStyle enumeration values.
HeaderBackColor	The background color of row and column headers.
HeaderFont	The font used for the column headers.
HeaderForeColor	The foreground color of the grid's column headers, including the column header text and the plus/minus glyphs (to expand rows when multiple related tables are displayed).
LinkColor	The color of text of all the links in the data grid, including links to child tables, the relation name, and so on.
ParentRowsBackColor	In a child table, this is the background color of the parent rows.
ParentRowsForeColor	In a child table, this is the foreground color of the parent rows.
ParentRowsLabelStyle	Determines whether the table and column names are displayed in the parent row, by means of the DataGridParentRowsLabelStyle enumeration.
PreferredColumnWidth	<p>The default width (in pixels) of columns in the grid. Set this property before resetting the DataSource and DataMember properties (either separately, or through the SetDataBinding method), or the property will have no effect.</p> <p>The property cannot be set to a value less than 0.</p>
PreferredRowHeight	<p>The row height (in pixels) of rows in the grid. Set this property before resetting the DataSource and DataMember properties (either separately, or through the SetDataBinding method), or the property will have no effect.</p> <p>The property cannot be set to a value less than 0.</p>
RowHeaderWidth	The width of the row headers of the grid.
SelectionBackColor	When a row or cell is selected, this is the background color.

PROPERTY	DESCRIPTION
SelectionForeColor	When a row or cell is selected, this is the foreground color.

NOTE

Keep in mind, when customizing the colors of controls, that it is possible to make the control inaccessible, due to poor color choice (for example, red and green). Use the colors available on the **System Colors** palette to avoid this issue.

The following procedures assume your form has a [DataGrid](#) control bound to a data table. For more information, see [Binding the Windows Forms DataGrid Control to a Data Source](#).

To set the table and column style of a data table programmatically

1. Create a new table style and set its properties.
2. Create a column style and set its properties.
3. Add the column style to the table style's column styles collection.
4. Add the table style to the data grid's table styles collection.
5. In the example below, create an instance of a new [DataGridTableStyle](#) and set its [MappingName](#) property.
6. Create a new instance of a [GridColumnStyle](#) and set its [MappingName](#) (and some other layout and display properties).
7. Repeat steps 2 through 6 for each column style you want to create.

The following example illustrates how a [DataGridTextBoxColumn](#) is created, because a name is to be displayed in the column. Additionally, you add the column style to the [GridColumnStylesCollection](#) of the table style, and you add the table style to the [GridTableStylesCollection](#) of the data grid.

```

Private Sub CreateAuthorFirstNameColumn()
    ' Add a GridTableStyle and set the MappingName
    ' to the name of the DataTable.
    Dim TSAuthors As New DataGridTableStyle()
    TSAuthors.MappingName = "Authors"

    ' Add a GridColumnStyle and set the MappingName
    ' to the name of a DataColumn in the DataTable.
    ' Set the HeaderText and Width properties.
    Dim TCFfirstName As New DataGridTextBoxColumn()
    TCFfirstName.MappingName = "AV_FName"
    TCFfirstName.HeaderText = "First Name"
    TCFfirstName.Width = 75
    TSAuthors.GridColumnStyles.Add(TCFfirstName)

    ' Add the DataGridTableStyle instance to
    ' the GridTableStylesCollection.
    myDataGrid.TableStyles.Add(TSAuthors)
End Sub

```

```

private void addCustomDataTableStyle()
{
    // Add a GridTableStyle and set the MappingName
    // to the name of the DataTable.
    DataGridTableStyle TSAuthors = new DataGridTableStyle();
    TSAuthors.MappingName = "Authors";

    // Add a GridColumnStyle and set the MappingName
    // to the name of a DataColumn in the DataTable.
    // Set the HeaderText and Width properties.
    DataGridColumnStyle TCFfirstName = new DataGridTextBoxColumn();
    TCFfirstName.MappingName = " AV_FName";
    TCFfirstName.HeaderText = "First Name";
    TCFfirstName.Width = 75;
    TSAuthors.GridColumnStyles.Add(TCFfirstName);

    // Add the DataGridTableStyle instance to
    // the GridTableStylesCollection.
    dataGrid1.TableStyles.Add(TSAuthors);
}

```

```

private:
void addCustomDataTableStyle()
{
    // Add a GridTableStyle and set the MappingName
    // to the name of the DataTable.
    DataGridTableStyle^ TSAuthors = gcnew DataGridTableStyle();
    TSAuthors->MappingName = "Authors";

    // Add a GridColumnStyle and set the MappingName
    // to the name of a DataColumn in the DataTable.
    // Set the HeaderText and Width properties.
    DataGridColumnStyle^ TCFfirstName = gcnew DataGridTextBoxColumn();
    TCFfirstName->MappingName = "AV_FName";
    TCFfirstName->HeaderText = "First Name";
    TCFfirstName->Width = 75;
    TSAuthors->GridColumnStyles->Add(TCFfirstName);

    // Add the DataGridTableStyle instance to
    // the GridTableStylesCollection.
    dataGrid1->TableStyles->Add(TSAuthors);
}

```

See Also

[GridTableStylesCollection](#)

[GridColumnStylesCollection](#)

[DataGrid](#)

[How to: Delete or Hide Columns in the Windows Forms DataGrid Control](#)

[DataGrid Control](#)

How to: Format the Windows Forms DataGrid Control Using the Designer

5/4/2018 • 4 min to read • [Edit Online](#)

NOTE

The [DataGridView](#) control replaces and adds functionality to the [DataGrid](#) control; however, the [DataGrid](#) control is retained for both backward compatibility and future use, if you choose. For more information, see [Differences Between the Windows Forms DataGridView and DataGrid Controls](#).

Applying different colors to various parts of a [DataGrid](#) control can help to make the information in it easier to read and interpret. Color can be applied to rows and columns. Rows and columns can also be hidden or shown at your discretion.

There are three basic aspects of formatting the [DataGrid](#) control:

- You can set properties to establish a default style in which data is displayed.
- From that base, you can then customize the way certain tables are displayed at run time.
- Finally, you can modify which columns are displayed in the data grid as well as the colors and other formatting that is shown.

As an initial step in formatting a data grid, you can set the properties of the [DataGrid](#) itself. These color and format choices form a base from which you can then make changes depending on the data tables and columns displayed.

The following procedure requires a **Windows Application** project with a form containing a [DataGrid](#) control. For information about setting up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#). In Visual Studio 2005, the [DataGrid](#) control is not in the [Toolbox](#) by default. For more information, see [How to: Add Items to the Toolbox](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To establish a default style for the DataGrid control

1. Select the [DataGrid](#) control.
2. In the **Properties** window, set the following properties, as appropriate.

PROPERTY	DESCRIPTION
AlternatingBackColor	The BackColor property defines the color of the even-numbered rows of the grid. When you set the AlternatingBackColor property to a different color, every other row is set to this new color (rows 1, 3, 5, and so on).

PROPERTY	DESCRIPTION
BackColor	The background color of the even-numbered rows of the grid (rows 0, 2, 4, 6, and so on).
BackgroundColor	Whereas the BackColor and AlternatingBackColor properties determines the color of rows in the grid, the BackgroundColor property determines the color of the area outside the row area, which is only visible when the grid is scrolled to the bottom, or if only a few rows are contained in the grid.
BorderStyle	The grid's border style, one of the BorderStyle enumeration values.
CaptionBackColor	The background color of the grid's window caption which appears immediately above the grid.
CaptionFont	The font of the caption at the top of the grid.
CaptionForeColor	The background color of the grid's window caption.
Font	The font used to display the text in the grid.
ForeColor	The color of the font displayed by the data in the rows of the data grid.
GridLineColor	The color of the grid lines of the data grid.
GridLineStyle	The style of the lines separating the cells of the grid, one of the DataGridLineStyle enumeration values.
HeaderBackColor	The background color of row and column headers.
HeaderFont	The font used for the column headers.
HeaderForeColor	The foreground color of the grid's column headers, including the column header text and the plus sign (+) and minus sign (-) glyphs that expand and collapse rows when multiple related tables are displayed.
LinkColor	The color of text of all the links in the data grid, including links to child tables, the relation name, and so on.
ParentRowsBackColor	In a child table, this is the background color of the parent rows.
ParentRowsForeColor	In a child table, this is the foreground color of the parent rows.
ParentRowsLabelStyle	Determines whether the table and column names are displayed in the parent row, by means of the DataGridParentRowsLabelStyle enumeration.

PROPERTY	DESCRIPTION
PreferredColumnWidth	The default width (in pixels) of columns in the grid. Set this property before resetting the DataSource and DataMember properties (either separately, or through the SetDataBinding method), or the property will have no effect. The property cannot be set to a value less than 0.
PreferredRowHeight	The row height (in pixels) of rows in the grid. Set this property before resetting the DataSource and DataMember properties (either separately, or through the SetDataBinding method), or the property will have no effect. The property cannot be set to a value less than 0.
RowHeaderWidth	The width of the row headers of the grid.
SelectionBackColor	When a row or cell is selected, this is the background color.
SelectionForeColor	When a row or cell is selected, this is the foreground color.

NOTE

When you are customizing the colors of controls, it is possible to make the control inaccessible due to poor color choice (for example, red and green). Use the colors available on the **System Colors** palette to avoid this issue.

The following procedure requires a [DataGrid](#) control bound to a data table. For more information, see [How to: Bind the Windows Forms DataGrid Control to a Data Source](#).

To set the table and column style of a data table at design time

1. Select the [DataGrid](#) control on your form.
2. In the **Properties** window, select the [TableStyles](#) property and click the **Ellipsis** (...) button.
3. In the **DataGridTableStyle Collection Editor** dialog box, click **Add** to add a table style to the collection.

With the **DataGridTableStyle Collection Editor**, you can add and remove table styles, set display and layout properties, and set the mapping name for the table styles.

4. Set the [MappingName](#) property to the mapping name for each table style.

The mapping name is used to specify which table style should be used with which table.

5. In the **DataGridTableStyle Collection Editor**, select the [GridColumnStyles](#) property and click the ellipsis button (...).
6. In the **DataGridColumnStyle Collection Editor** dialog box, add column styles to the table style you created.

With the **DataGridColumnStyle Collection Editor**, you can add and remove column styles, set display and layout properties, and set the mapping name and formatting strings for the data columns.

NOTE

For more information about formatting strings, see [Formatting Types](#).

See Also

[GridTableStylesCollection](#)

[GridColumnStylesCollection](#)

[DataGrid](#)

[How to: Delete or Hide Columns in the Windows Forms DataGrid Control](#)

[DataGrid Control](#)

How to: Respond to Clicks in the Windows Forms DataGrid Control

5/4/2018 • 2 min to read • [Edit Online](#)

NOTE

The [DataGridView](#) control replaces and adds functionality to the [DataGrid](#) control; however, the [DataGrid](#) control is retained for both backward compatibility and future use, if you choose. For more information, see [Differences Between the Windows Forms DataGridView and DataGrid Controls](#).

After the Windows Forms [DataGrid](#) is connected to a database, you can monitor which cell the user clicked.

To detect when the user of the DataGrid selects a different cell

- In the [CurrentCellChanged](#) event handler, write code to respond appropriately.

```
Private Sub myDataGrid_CurrentCellChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles myDataGrid.CurrentCellChanged
    MessageBox.Show("Col is " & myDataGrid.CurrentCell.ColumnNumber _
        & ", Row is " & myDataGrid.CurrentCell.RowNumber _
        & ", Value is " & myDataGrid.Item(myDataGrid.CurrentCell))
End Sub
```

```
private void myDataGrid_CurrentCellChanged(object sender,
System.EventArgs e)
{
    MessageBox.Show ("Col is " + myDataGrid.CurrentCell.ColumnNumber
        + ", Row is " + myDataGrid.CurrentCell.RowNumber
        + ", Value is " + myDataGrid[myDataGrid.CurrentCell] );
}
```

(Visual C#) Place the following code in the form's constructor to register the event handler.

```
this.myDataGrid.CurrentCellChanged += new
    System.EventHandler(this.myDataGrid_CurrentCellChanged);
```

To determine which part of the DataGrid the user clicked

- Call the [HitTest](#) method in an appropriate event handler, such as for the [MouseDown](#) or [Click](#) event.

The [HitTest](#) method returns a [DataGrid.HitTestInfo](#) object that contains the row and column of a clicked area.

```
Private Sub myDataGrid_MouseDown(ByVal sender As Object, _
ByVal e As MouseEventArgs) Handles myDataGrid.MouseDown
    Dim myGrid As DataGrid = CType(sender, DataGrid)
    Dim hti As System.Windows.Forms.DataGrid.HitTestInfo
    hti = myGrid.HitTest(e.X, e.Y)
    Dim message As String = "You clicked "

    Select Case hti.Type
        Case System.Windows.Forms.DataGrid.HitTestType.None
            message &= "the background."
        Case System.Windows.Forms.DataGrid.HitTestType.Cell
            message &= "cell at row " & hti.Row & ", col " & hti.Column
        Case System.Windows.Forms.DataGrid.HitTestType.ColumnHeader
            message &= "the column header for column " & hti.Column
        Case System.Windows.Forms.DataGrid.HitTestType.RowHeaders
            message &= "the row header for row " & hti.Row
        Case System.Windows.Forms.DataGrid.HitTestType.ColumnResize
            message &= "the column resizer for column " & hti.Column
        Case System.Windows.Forms.DataGrid.HitTestType.RowHeaders
            message &= "the row resizer for row " & hti.Row
        Case System.Windows.Forms.DataGrid.HitTestType.Caption
            message &= "the caption"
        Case System.Windows.Forms.DataGrid.HitTestType.ParentRows
            message &= "the parent row"
    End Select

    Console.WriteLine(message)
End Sub
```

```

private void myDataGrid_MouseDown(object sender,
System.Windows.Forms.MouseEventArgs e)
{
    DataGrid myGrid = (DataGrid) sender;
    System.Windows.Forms.DataGrid.HitTestInfo hti;
    hti = myGrid.HitTest(e.X, e.Y);
    string message = "You clicked ";

    switch (hti.Type)
    {
        case System.Windows.Forms.DataGrid.HitTestType.None :
            message += "the background.";
            break;
        case System.Windows.Forms.DataGrid.HitTestType.Cell :
            message += "cell at row " + hti.Row + ", col " + hti.Column;
            break;
        case System.Windows.Forms.DataGrid.HitTestType.ColumnHeader :
            message += "the column header for column " + hti.Column;
            break;
        case System.Windows.Forms.DataGrid.HitTestType.RowHeader :
            message += "the row header for row " + hti.Row;
            break;
        case System.Windows.Forms.DataGrid.HitTestType.ColumnResize :
            message += "the column resizer for column " + hti.Column;
            break;
        case System.Windows.Forms.DataGrid.HitTestType.RowResize :
            message += "the row resizer for row " + hti.Row;
            break;
        case System.Windows.Forms.DataGrid.HitTestType.Caption :
            message += "the caption";
            break;
        case System.Windows.Forms.DataGrid.HitTestType.ParentRows :
            message += "the parent row";
            break;
    }

    Console.WriteLine(message);
}

```

(Visual C#) Place the following code in the form's constructor to register the event handler.

```

this.myDataGrid.MouseDown += new
    System.Windows.Forms.MouseEventHandler
    (this.myDataGrid_MouseDown);

```

See Also

[DataGrid Control](#)

[How to: Change Displayed Data at Run Time in the Windows Forms DataGrid Control](#)

How to: Validate Input with the Windows Forms DataGrid Control

5/4/2018 • 2 min to read • [Edit Online](#)

NOTE

The [DataGridView](#) control replaces and adds functionality to the [DataGrid](#) control; however, the [DataGrid](#) control is retained for both backward compatibility and future use, if you choose. For more information, see [Differences Between the Windows Forms DataGridView and DataGrid Controls](#).

There are two types of input validation available for the Windows Forms [DataGrid](#) control. If the user attempts to enter a value that is of an unacceptable data type for the cell, for example a string into an integer, the new invalid value is replaced with the old value. This kind of input validation is done automatically and cannot be customized.

The other type of input validation can be used to reject any unacceptable data, for example a 0 value in a field that must be greater than or equal to 1, or an inappropriate string. This is done in the dataset by writing an event handler for the [ColumnChanging](#) or [RowChanging](#) event. The example below uses the [ColumnChanging](#) event because the unacceptable value is disallowed for the "Product" column in particular. You might use the [RowChanging](#) event for checking that the value of an "End Date" column is later than the "Start Date" column in the same row.

To validate user input

1. Write code to handle the [ColumnChanging](#) event for the appropriate table. When inappropriate input is detected, call the [SetColumnError](#) method of the [DataRow](#) object.

```
Private Sub Customers_ColumnChanging(ByVal sender As Object, _
ByVal e As System.Data.DataColumnChangeEventArgs)
    ' Only check for errors in the Product column
    If (e.Column.ColumnName.Equals("Product")) Then
        ' Do not allow "Automobile" as a product.
        If CType(e.ProposedValue, String) = "Automobile" Then
            Dim badValue As Object = e.ProposedValue
            e.ProposedValue = "Bad Data"
            e.Row.RowError = "The Product column contains an error"
            e.Row.SetColumnError(e.Column, "Product cannot be " & _
                CType(badValue, String))
        End If
    End If
End Sub
```

```
//Handle column changing events on the Customers table
private void Customers_ColumnChanging(object sender, System.Data.DataColumnChangeEventArgs e) {

    //Only check for errors in the Product column
    if (e.Column.ColumnName.Equals("Product")) {

        //Do not allow "Automobile" as a product
        if (e.ProposedValue.Equals("Automobile")) {
            object badValue = e.ProposedValue;
            e.ProposedValue = "Bad Data";
            e.Row.RowError = "The Product column contains an error";
            e.Row.SetColumnError(e.Column, "Product cannot be " + badValue);
        }
    }
}
```

2. Connect the event handler to the event.

Place the following code within either the form's [Load](#) event or its constructor.

```
' Assumes the grid is bound to a dataset called customersDataSet1
' with a table called Customers.
' Put this code in the form's Load event or its constructor.
AddHandler customersDataSet1.Tables("Customers").ColumnChanging, AddressOf Customers_ColumnChanging
```

```
// Assumes the grid is bound to a dataset called customersDataSet1
// with a table called Customers.
// Put this code in the form's Load event or its constructor.
customersDataSet1.Tables["Customers"].ColumnChanging += new
DataColumnChangeEventHandler(this.Customers_ColumnChanging);
```

See Also

[DataGridView](#)
[ColumnChanging](#)
[SetColumnError](#)
[DataGridView Control](#)

Keyboard Shortcuts for the Windows Forms DataGrid Control

5/4/2018 • 2 min to read • [Edit Online](#)

NOTE

The [DataGridView](#) control replaces and adds functionality to the [DataGrid](#) control; however, the [DataGrid](#) control is retained for both backward compatibility and future use, if you choose. For more information, see [Differences Between the Windows Forms DataGridView and DataGrid Controls](#).

The following table lists the keyboard shortcuts that can be used for navigation within the Windows Forms [DataGrid](#) control:

ACTION	SHORTCUT
Complete a cell entry and move down to the next cell. If focus is on a child table link, navigate to that table.	ENTER
Cancel cell editing if in cell edit mode. If in marquee selection, cancel editing on the row.	ESC
Delete the character before the insertion point when editing a cell.	BACKSPACE
Delete the character after the insertion point when editing a cell.	DELETE
Move to the first cell in the current row.	HOME
Move to the last cell in the current row.	END
Highlight characters in the current cell and position the insertion point at the end of the line. Same behavior as double-clicking a cell.	F2
If focus is on a cell, move to the next cell in the row. If focus is on the last cell in a row, move to the first child table link of the row and expand it. If focus is on a child link, move to the next child link. If focus is on the last child link, move to the first cell of the next row.	TAB

ACTION	SHORTCUT
If focus is on a cell, move to the previous cell in the row.	SHIFT+TAB
If focus is on the first cell in a row, move to the last expanded child table link of the previous row, or move to the last cell of the previous row.	
If focus is on a child link, move to the previous child link.	
If focus is on the first child link, move to the last cell of the previous row.	
Move to the next control in the tab order.	CTRL+ TAB
Move to the previous control in the tab order.	CTRL+SHIFT+ TAB
Move up to the parent table if in a child table. Same behavior as clicking the Back button.	ALT+LEFT ARROW
Expand child table links. ALT+DOWN ARROW expands all links, not just the ones selected.	ALT+DOWN ARROW or CTRL+PLUS SIGN
Collapse child table links. ALT+UP ARROW collapses all links, not just the ones selected.	ALT+UP ARROW or CTRL-MINUS SIGN
Move to the farthest nonblank cell in the direction of the arrow.	CTRL+ARROW
Extend the selection one row in the direction of the arrow (excluding child table links).	SHIFT+UP/DOWN ARROW
Extend the selection to farthest nonblank row in the direction of the arrow (excluding child table links).	CTRL+SHIFT+ UP/DOWN ARROW
Move to the upper-left cell.	CTRL+HOME
Move to the lower-right cell.	CTRL+END
Extend the selection to the top row.	CTRL+SHIFT+HOME
Extend the selection to the bottom row.	CTRL+SHIFT+END
Select the current row (excluding child table links).	SHIFT+SPACEBAR
Select the entire grid (excluding child table links).	CTRL+A
Display the parent row when in a child table.	CTRL+PAGE DOWN
Hide the parent row when in a child table.	CTRL+PAGE UP
Extend the selection down one screen (excluding child table links).	SHIFT+PAGE DOWN
Extend the selection up one screen (excluding child table links).	SHIFT+PAGE UP

ACTION	SHORTCUT
Call the EndEdit method for the current row.	CTRL+ENTER
Enter a DBNull.Value value into a cell when in edit mode.	CTRL+0

See Also

[DataGrid Control Overview](#)

[DataGrid Control](#)

DataGridView Control (Windows Forms)

5/4/2018 • 2 min to read • [Edit Online](#)

The `DataGridView` control provides a powerful and flexible way to display data in a tabular format. You can use the `DataGridView` control to show read-only views of a small amount of data, or you can scale it to show editable views of very large sets of data.

You can extend the `DataGridView` control in a number of ways to build custom behaviors into your applications. For example, you can programmatically specify your own sorting algorithms, and you can create your own types of cells. You can easily customize the appearance of the `DataGridView` control by choosing among several properties. Many types of data stores can be used as a data source, or the `DataGridView` control can operate with no data source bound to it.

The topics in this section describe the concepts and techniques that you can use to build `DataGridView` features into your applications.

In This Section

[DataGridView Control Overview](#)

Provides topics that describe the architecture and core concepts of the Windows Forms `DataGridView` control.

[Default Functionality in the Windows Forms DataGridView Control](#)

Describes the default appearance and behavior of the Windows Forms `DataGridView` control when it is bound to a data source.

[Column Types in the Windows Forms DataGridView Control](#)

Describes the column types in the Windows Forms `DataGridView` control used to display data and allow users to modify or add data.

[Basic Column, Row, and Cell Features in the Windows Forms DataGridView Control](#)

Provides topics that describe commonly-used cell, row, and column properties.

[Basic Formatting and Styling in the Windows Forms DataGridView Control](#)

Provides topics that describe how to modify the basic appearance of the control and the display formatting of cell data.

[Displaying Data in the Windows Forms DataGridView Control](#)

Provides topics that describe how to populate the control with data either manually, or from an external data source.

[Resizing Columns and Rows in the Windows Forms DataGridView Control](#)

Provides topics that describe how the size of rows and columns can be adjusted automatically to fit cell content or to fit the available width of the control.

[Sorting Data in the Windows Forms DataGridView Control](#)

Provides topics that describe the sorting features in the control.

[Data Entry in the Windows Forms DataGridView Control](#)

Provides topics that describe how to change the way users add and modify data in the control.

[Selection and Clipboard Use with the Windows Forms DataGridView Control](#)

Provides topics that describe the cell, row, and column selection features in the control.

[Programming with Cells, Rows, and Columns in the Windows Forms DataGridView Control](#)

Provides topics that describe how to program with cell, row, and column objects.

[Customizing the Windows Forms DataGridView Control](#)

Provides topics that describe custom painting `DataGridView` cells and rows, and creating derived cell, column, and row types.

[Performance Tuning in the Windows Forms DataGridView Control](#)

Provides topics that describe how to use the control efficiently to avoid performance problems when working with large amounts of data.

[Default Keyboard and Mouse Handling in the Windows Forms DataGridView Control](#)

Describes how users can interact with the `DataGridView` control through a keyboard and a mouse.

[Differences Between the Windows Forms DataGridView and DataGrid Controls](#)

Describes how the `DataGridView` control improves upon and replaces the `DataGrid` control.

Also see [Using the Designer with the Windows Forms DataGridView Control](#).

Reference

[DataGridView](#)

Provides reference documentation for the `DataGridView` control.

[BindingSource](#)

Provides reference documentation for the `BindingSource` component. The `DataGridView` control and the `BindingSource` component are designed to work closely together.

See Also

[Controls to Use on Windows Forms](#)

Using the Designer with the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

Visual Studio provides designer support for the `DataGridView` control that enables you to perform many setup tasks without writing code. These tasks include binding the control to a data source, modifying the columns used to display data, and adjusting the appearance and basic behavior of the control.

In This Section

[How to: Add and Remove Columns in the Windows Forms DataGridView Control Using the Designer](#)

Describes how to use the **Add Columns** and **Edit Columns** dialog boxes to populate and modify the columns collection.

[How to: Bind Data to the Windows Forms DataGridView Control Using the Designer](#)

Describes how to use the **Choose Data Source** option on the control's smart tag to connect to data.

[How to: Change the Order of Columns in the Windows Forms DataGridView Control Using the Designer](#)

Describes how to use the **Edit Columns** dialog box to rearrange columns.

[How to: Change the Type of a Windows Forms DataGridView Column Using the Designer](#)

Describes how to use the **Edit Columns** dialog box to change column types.

[How to: Enable Column Reordering in the Windows Forms DataGridView Control Using the Designer](#)

Describes how to use the control's smart tag to enable users to rearrange columns.

[How to: Freeze Columns in the Windows Forms DataGridView Control Using the Designer](#)

Describes how to use the **Edit Columns** dialog box to prevent specific columns from scrolling.

[How to: Hide Columns in the Windows Forms DataGridView Control Using the Designer](#)

Describes how to use the **Edit Columns** dialog box to hide specific columns.

[How to: Make Columns Read-Only in the Windows Forms DataGridView Control Using the Designer](#)

Describes how to use the **Edit Columns** dialog box to prevent users from editing values in specific columns.

[How to: Prevent Row Addition and Deletion in the Windows Forms DataGridView Control Using the Designer](#)

Describes how to use the control's smart tag to prevent users from adding or deleting rows.

[How to: Set Alternating Row Styles for the Windows Forms DataGridView Control Using the Designer](#)

Describes how to use the **CellStyle Builder** dialog box to create a ledger-like appearance in the control.

[How to: Set Default Cell Styles and Data Formats for the Windows Forms DataGridView Control Using the Designer](#)

Describes how to use the **CellStyle Builder** dialog box to set up the basic appearance and data-display formats for the control.

Reference

[DataGridView](#)

Provides reference documentation for the `DataGridView` control.

See Also

DataGridView Control

How to: Add and Remove Columns in the Windows Forms DataGridView Control Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [DataGridView](#) control must contain columns in order to display data. If you plan to populate the control manually, you must add the columns yourself. Alternately, you can bind the control to a data source, which generates and populates the columns automatically. If the data source contains more columns than you want to display, you can remove the unwanted columns.

The following procedures require a **Windows Application** project with a form containing a [DataGridView](#) control. For information about setting up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To add a column using the designer

1. Click the smart tag glyph (□) on the upper-right corner of the [DataGridView](#) control, and then select **Add Column**.
2. In the **Add Column** dialog box, choose the **Databound Column** option and select a column from the data source, or choose the **Unbound Column** option and define the column using the fields provided.
3. Click the **Add** button to add the column, causing it to appear in the designer if the existing columns do not already fill the control display area.

NOTE

You can modify column properties in the **Edit Columns** dialog box, which you can access from the control's smart tag.

To remove a column using the designer

1. Choose **Edit Columns** from the control's smart tag.
2. Select a column from the **Selected Columns** list.
3. Click the **Remove** button to delete the column, causing it to disappear from the designer.

See Also

[DataGridView](#)

[How to: Create a Windows Application Project](#)

[How to: Add Controls to Windows Forms](#)

How to: Bind Data to the Windows Forms DataGridView Control Using the Designer

5/4/2018 • 3 min to read • [Edit Online](#)

You can use the designer to connect a [DataGridView](#) control to data sources of several different varieties, including databases, business objects, or Web services. When you bind the control to a data source using the designer, the control is automatically bound to a [BindingSource](#) component that represents the data source. Additionally, columns are automatically generated in the control to match the schema information provided by the data source.

After columns have been generated, you can modify them to meet your needs. For example, you can remove or hide columns you are not interested in displaying, you can rearrange the columns, or you can modify the column types. For more information about modifying columns, see the topics listed in the See Also section.

You can also bind multiple [DataGridView](#) controls to related tables to create master/detail relationships. In this configuration, one control displays a parent table and another control displays only those rows from a child table that are related to the current row in the parent table. For more information, see [How to: Display Related Data in a Windows Forms Application](#).

The following procedure requires a **Windows Application** project with a form that contains a [DataGridView](#) control or two controls for a master/detail relationship. For information about starting such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To bind the control to a data source

1. Click the smart tag glyph (ⓘ) on the upper-right corner of the [DataGridView](#) control.
2. Click the drop-down arrow for the **Choose Data Source** option.
3. If your project does not already have a data source, click **Add Project Data Source** and follow the steps indicated by the wizard.

For more information, see [Data Source Configuration Wizard](#). Your new data source will appear in the **Choose Data Source** drop-down window. If your new data source contains only one member, such as a single database table, the control will automatically bind to that member. Otherwise, continue to the next step.

4. Expand the **Other Data Sources** and **Project Data Sources** nodes if they are not already expanded, and then select the data source to bind the control to.
5. If your data source contains more than one member, such as if you have created a [System.Data.DataSet](#) that contains multiple tables, expand the data source, and then select the specific member to bind to.
6. To create a master/detail relationship, in the **Choose Data Source** drop-down window for a second [DataGridView](#) control, expand the [BindingSource](#) created for the parent table, and then select the related child table from the list shown.

NOTE

If your project already has a data source, you can also use the **Data Sources** window to create a data form. For more information, see [Data Sources Window](#).

See Also

[DataGridView](#)

[BindingSource](#)

[DataGridViewDataMember](#)

[DataGridView.DataSource](#)

[How to: Connect to Data in a Database](#)

[How to: Add and Remove Columns in the Windows Forms DataGridView Control Using the Designer](#)

[How to: Change the Order of Columns in the Windows Forms DataGridView Control Using the Designer](#)

[How to: Change the Type of a Windows Forms DataGridView Column Using the Designer](#)

[How to: Freeze Columns in the Windows Forms DataGridView Control Using the Designer](#)

[How to: Hide Columns in the Windows Forms DataGridView Control Using the Designer](#)

[How to: Make Columns Read-Only in the Windows Forms DataGridView Control Using the Designer](#)

[How to: Create a Windows Application Project](#)

[How to: Add Controls to Windows Forms](#)

[Data Sources Window](#)

[How to: Display Related Data in a Windows Forms Application](#)

How to: Change the Order of Columns in the Windows Forms DataGridView Control Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

When you bind a Windows Forms [DataGridView](#) control to a data source, the display order of the automatically generated columns is dictated by the data source. If this order is not what you prefer, you can change the order of the columns using the designer. You may also want to add unbound columns to the control and change their display order. For information about how to change the column order programmatically, see [How to: Change the Order of Columns in the Windows Forms DataGridView Control](#).

The following procedure requires a **Windows Application** project with a form containing a [DataGridView](#) control. For information about setting up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#)

To change the column order using the designer

1. Click the smart tag glyph (ⓘ) on the upper-right corner of the [DataGridView](#) control, and then select **Edit Columns**.
2. Select a column from the **Selected Columns** list.
3. Click the up or down arrow to the right of the **Selected Columns** list until the selected column is in the position you want.

See Also

[DataGridView](#)

[How to: Add and Remove Columns in the Windows Forms DataGridView Control Using the Designer](#)

[How to: Create a Windows Application Project](#)

[How to: Add Controls to Windows Forms](#)

How to: Change the Type of a Windows Forms DataGridView Column Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

Sometimes you will want to change the type of a column that has already been added to a Windows Forms [DataGridView](#) control. For example, you may want to modify the types of some of the columns that are generated automatically when you bind the control to a data source. This is useful when the table you display has columns containing foreign keys to rows in a related table. In this case, you may want to replace the text box columns that display these foreign keys with combo box columns that display more meaningful values from the related table.

The following procedure requires a **Windows Application** project with a form containing a [DataGridView](#) control. For information about setting up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To change the type of a column using the designer

1. Click the smart tag glyph (ⓘ) on the upper-right corner of the [DataGridView](#) control, and then select **Edit Columns**.
2. Select a column from the **Selected Columns** list.
3. In the **Column Properties** grid, set the `ColumnType` property to the new column type.

NOTE

The `ColumnType` property is a design-time-only property that indicates the class representing the column type. It is not an actual property defined in a column class.

See Also

- [DataGridView](#)
- [DataGridViewColumn](#)
- [How to: Create a Windows Application Project](#)
- [How to: Add Controls to Windows Forms](#)

How to: Enable Column Reordering in the Windows Forms DataGridView Control Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

When viewing data displayed in a Windows Forms [DataGridView](#) control, users sometimes want to compare the values in specific columns. This can be inconvenient if the columns are widely separated in the control, especially if users must scroll back and forth horizontally in order to see all the columns they are interested in. You can make the task of comparing column values easier by enabling your users to reorder the columns. When you enable column reordering, users can move a column to a new position by dragging the column header with the mouse.

The following procedure requires a **Windows Application** project with a form containing a [DataGridView](#) control. For information about setting up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To enable column reordering

- Click the smart tag glyph (ⓘ) on the upper-right corner of the [DataGridView](#) control, and then select **Enable Column Reordering**.

See Also

[DataGridView](#)

[DataGridView.AllowUserToOrderColumns](#)

[How to: Freeze Columns in the Windows Forms DataGridView Control Using the Designer](#)

[How to: Create a Windows Application Project](#)

[How to: Add Controls to Windows Forms](#)

How to: Freeze Columns in the Windows Forms DataGridView Control Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

When users view data displayed in a Windows Forms [DataGridView](#) control, they sometimes need to refer to a single column or set of columns frequently. For example, when you display a table of customer information that contains many columns, it is useful for you to display the customer name at all times while enabling other columns to scroll outside the visible region.

To achieve this behavior, you can freeze columns in the control. When you freeze a column, all the columns to its left (or to its right in right-to-left language scripts) are frozen as well. Frozen columns remain in place while all other columns can scroll. If column reordering is enabled, the frozen columns are treated as a group distinct from the unfrozen columns. Users can reposition columns in either group, but they cannot move a column from one group to the other.

The following procedure requires a **Windows Application** project with a form containing a [DataGridView](#) control. For information about setting up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To freeze a column using the designer

1. Click the smart tag glyph (ⓘ) on the upper-right corner of the [DataGridView](#) control, and then select **Edit Columns**.
2. Select a column from the **Selected Columns** list.
3. In the **Column Properties** grid, set the [Frozen](#) property to `true`.

NOTE

You can also freeze a column when adding it by selecting the **Frozen** box in the **Add Column** dialog box.

See Also

[DataGridView](#)

[DataGridViewColumn.Frozen](#)

[How to: Add and Remove Columns in the Windows Forms DataGridView Control Using the Designer](#)

[How to: Enable Column Reordering in the Windows Forms DataGridView Control Using the Designer](#)

[How to: Display Right-to-Left Text in Windows Forms for Globalization](#)

[How to: Create a Windows Application Project](#)

[How to: Add Controls to Windows Forms](#)

How to: Hide Columns in the Windows Forms DataGridView Control Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

Sometimes you will want to display only some of the columns that are available in a Windows Forms [DataGridView](#) control. For example, you may want to show an employee salary column to users with management credentials while hiding it from other users. Alternately, you may want to bind the control to a data source that contains many columns, only some of which you want to display. In this case, you will typically remove the columns you are not interested in displaying rather than hiding them. For more information, see [How to: Add and Remove Columns in the Windows Forms DataGridView Control Using the Designer](#).

The following procedure requires a **Windows Application** project with a form containing a [DataGridView](#) control. For information about setting up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To hide a column using the designer

1. Click the smart tag glyph (ⓘ) on the upper-right corner of the [DataGridView](#) control, and then select **Edit Columns**.
2. Select a column from the **Selected Columns** list.
3. In the **Column Properties** grid, set the **Visible** property to `false`.

NOTE

You can also hide a column when adding it by clearing the **Visible** check box in the **Add Column** dialog box.

See Also

[DataGridView](#)

[DataGridViewColumn.Visible](#)

[How to: Add and Remove Columns in the Windows Forms DataGridView Control Using the Designer](#)

[How to: Create a Windows Application Project](#)

[How to: Add Controls to Windows Forms](#)

How to: Make Columns Read-Only in the Windows Forms DataGridView Control Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

By default, users can modify text and numerical data displayed in the Windows Forms `DataGridView` control. If you want to display data that is not meant for modification, you must make the columns that contain the data read-only. For information about how to make the control entirely read-only, see [How to: Prevent Row Addition and Deletion in the Windows Forms DataGridView Control Using the Designer](#).

The following procedure requires a **Windows Application** project with a form containing a `DataGridView` control. For information about setting up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To make a column read-only by using the designer

1. Click the smart tag glyph (ⓘ) on the upper-right corner of the `DataGridView` control, and then select **Edit Columns**.
2. Select a column from the **Selected Columns** list.
3. In the **Column Properties** grid, set the `ReadOnly` property to `true`.

NOTE

You can also make a column read-only when you add it by selecting the **Read Only** check box in the **Add Column** dialog box.

See Also

[DataGridView](#)

[DataGridViewColumn.ReadOnly](#)

[How to: Add and Remove Columns in the Windows Forms DataGridView Control Using the Designer](#)

[How to: Prevent Row Addition and Deletion in the Windows Forms DataGridView Control Using the Designer](#)

[How to: Create a Windows Application Project](#)

[How to: Add Controls to Windows Forms](#)

How to: Prevent Row Addition and Deletion in the Windows Forms DataGridView Control Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

Sometimes you will want to prevent users from entering new rows of data or deleting existing rows in your [DataGridView](#) control. New rows are entered in the special row for new records at the bottom of the control. When you disable row addition, the row for new records is not displayed. You can then make the control entirely read-only by disabling row deletion and cell editing.

The following procedure requires a **Windows Application** project with a form containing a [DataGridView](#) control. For information about setting up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To prevent row addition and deletion

- Click the smart tag glyph (ⓘ) on the upper-right corner of the [DataGridView](#) control, and then clear the **Enable Adding** and **Enable Deleting** check boxes.

NOTE

To make the control entirely read-only, clear the **Enable Editing** check box as well.

See Also

[DataGridView](#)

[DataGridView.AllowUserToAddRows](#)

[How to: Create a Windows Application Project](#)

[How to: Add Controls to Windows Forms](#)

How to: Set Alternating Row Styles for the Windows Forms DataGridView Control Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

Tabular data is often presented in a ledger-like format where alternating rows have different background colors. This format makes it easier for users to tell which cells are in each row, especially with wide tables that have many columns.

With the [DataGridView](#) control, you can specify complete style information for alternating rows. You can use style characteristics like foreground color and font, in addition to background color, to differentiate alternating rows. For more information, see [Cell Styles in the Windows Forms DataGridView Control](#).

The following procedure requires a **Windows Application** project with a form containing a [DataGridView](#) control. For information about setting up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

Define styles for alternating rows

1. Select the [DataGridView](#) control in the designer.
2. In the **Properties** window, click the ellipsis button (next to the [AlternatingRowsDefaultCellStyle](#) property.
3. In the **CellStyle Builder** dialog box, define the style by setting the properties, and use the **Preview** pane to confirm your choices. The styles you specify are used for every other row displayed in the control, starting with the second one.
4. To define styles for the remaining rows, repeat steps 2 and 3 using the [RowsDefaultCellStyle](#) property.

NOTE

Cells are displayed using styles inherited from multiple properties. For more information about style inheritance, see [Cell Styles in the Windows Forms DataGridView Control](#).

See Also

[DataGridView](#)

[Cell Styles in the Windows Forms DataGridView Control](#)

[Basic Formatting and Styling in the Windows Forms DataGridView Control](#)

[Using the Designer with the Windows Forms DataGridView Control](#)

[How to: Create a Windows Application Project](#)

[How to: Add Controls to Windows Forms](#)

How to: Set Default Cell Styles and Data Formats for the Windows Forms DataGridView Control Using the Designer

5/4/2018 • 2 min to read • [Edit Online](#)

The [DataGridView](#) control lets you specify default cell styles and cell data formats for the entire control, for specific columns, for row and column headers, and for alternating rows to create a ledger effect. Default styles set for the entire control are overridden by default styles set for columns and alternating rows. Additionally, styles that you set in code for individual rows and cells override the default styles.

For more information about cell styles, see [Cell Styles in the Windows Forms DataGridView Control](#). To set styles for alternating rows, see [How to: Set Alternating Row Styles for the Windows Forms DataGridView Control Using the Designer](#).

You can also set styles using the [RowTemplate](#) property to affect all rows that will be added to the control. For more information about the row template, see [How to: Use the Row Template to Customize Rows in the Windows Forms DataGridView Control](#).

The following procedures require a **Windows Application** project with a form containing a [DataGridView](#) control. For information about setting up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To set default styles for all cells in the control

1. Select the [DataGridView](#) control in the designer.
2. In the **Properties** window, click the ellipsis button (...) next to the [DefaultCellStyle](#), [ColumnHeadersDefaultCellStyle](#), or [RowHeadersDefaultCellStyle](#) property. The **CellStyle Builder** dialog box appears.
3. Define the style by setting the properties, using the **Preview** pane to confirm your choices.

NOTE

If visual styles are enabled, the row and column headers (except for the [TopLeftHeaderCell](#)) are automatically styled by the current theme, overriding the [ColumnHeadersDefaultCellStyle](#) and [RowHeadersDefaultCellStyle](#) property values.

You can set cell styles for multiple selected [DataGridView](#) controls using the designer, but only if they have identical values for the cell style property you want to modify. If any cell styles differ for that property, the **Properties** windows of the **CellStyle Builder** dialog box will be blank.

To set default styles for cells in individual columns

1. Right-click the [DataGridView](#) control in the designer and choose **Edit Columns**.
2. Select a column from the **Selected Columns** list.

3. In the **Column Properties** grid, click the ellipsis button (...) next to the **DefaultCellStyle** property. The **CellStyle Builder** dialog box appears.
4. Define the style by setting the properties, using the **Preview** pane to confirm your choices.

To format data in cells

1. Use one of the preceding procedures to display a **CellStyle Builder** dialog box related to a default cell style property.
2. In the **CellStyle Builder** dialog box, click the ellipsis button (...) next to the **Format** property. The **Format String** dialog box appears.
3. Select a format type, then modify the details of the type (such as the number of decimal places to display), using the **Sample** box to confirm your choices.
4. If you are binding the **DataGridView** control to a data source that is likely to contain null values, fill in the **Null Value** text box. This value is displayed when the cell value is equal to a null reference (Nothing in Visual Basic) or **DBNull.Value**.

See Also

[DataGridView](#)

[DataGridViewCellStyle](#)

[DataGridView.DefaultCellStyle](#)

[DataGridView.RowsDefaultCellStyle](#)

[DataGridViewColumn.DefaultCellStyle](#)

[DataGridViewCellStyle.Format](#)

[Cell Styles in the Windows Forms DataGridView Control](#)

[How to: Set Alternating Row Styles for the Windows Forms DataGridView Control Using the Designer](#)

[How to: Create a Windows Application Project](#)

[How to: Add Controls to Windows Forms](#)

DataGridView Control Overview (Windows Forms)

5/4/2018 • 2 min to read • [Edit Online](#)

NOTE

The [DataGridView](#) control replaces and adds functionality to the [DataGrid](#) control; however, the [DataGrid](#) control is retained for both backward compatibility and future use, if you choose. For more information, see [Differences Between the Windows Forms DataGridView and DataGrid Controls](#).

With the [DataGridView](#) control, you can display and edit tabular data from many different kinds of data sources.

Binding data to the [DataGridView](#) control is straightforward and intuitive, and in many cases it is as simple as setting the [DataSource](#) property. When you bind to a data source that contains multiple lists or tables, set the [DataMember](#) property to a string that specifies the list or table to bind to.

The [DataGridView](#) control supports the standard Windows Forms data binding model, so it will bind to instances of classes described in the following list:

- Any class that implements the [IList](#) interface, including one-dimensional arrays.
- Any class that implements the [IListSource](#) interface, such as the [DataTable](#) and [DataSet](#) classes.
- Any class that implements the [IBindingList](#) interface, such as the [BindingList<T>](#) class.
- Any class that implements the [IBindingListView](#) interface, such as the [BindingSource](#) class.

The [DataGridView](#) control supports data binding to the public properties of the objects returned by these interfaces or to the properties collection returned by an [ICustomTypeDescriptor](#) interface, if implemented on the returned objects.

Typically, you will bind to a [BindingSource](#) component and bind the [BindingSource](#) component to another data source or populate it with business objects. The [BindingSource](#) component is the preferred data source because it can bind to a wide variety of data sources and can resolve many data binding issues automatically. For more information, see [BindingSource Component](#).

The [DataGridView](#) control can also be used in *unbound* mode, with no underlying data store. For a code example that uses an unbound [DataGridView](#) control, see [Walkthrough: Creating an Unbound Windows Forms DataGridView Control](#).

The [DataGridView](#) control is highly configurable and extensible, and it provides many properties, methods, and events to customize its appearance and behavior. When you want your Windows Forms application to display tabular data, consider using the [DataGridView](#) control before others (for example, [DataGrid](#)). If you are displaying a small grid of read-only values, or if you are enabling a user to edit a table with millions of records, the [DataGridView](#) control will provide you with a readily programmable, memory-efficient solution.

In This Section

[DataGridView Control Technology Summary](#)

Summarizes [DataGridView](#) control concepts and the use of related classes.

[DataGridView Control Architecture](#)

Describes the architecture of the [DataGridView](#) control, explaining its type hierarchy and inheritance structure.

[DataGridView Control Scenarios](#)

Describes the most common scenarios in which [DataGridView](#) controls are used.

[DataGridView Control Code Directory](#)

Provides links to code examples in the documentation for various [DataGridView](#) tasks. These examples are categorized by task type.

Related Sections

[Column Types in the Windows Forms DataGridView Control](#)

Discusses the column types in the Windows Forms [DataGridView](#) control used to display information and allow users to modify or add information.

[Displaying Data in the Windows Forms DataGridView Control](#)

Provides topics that describe how to populate the control with data either manually, or from an external data source.

[Customizing the Windows Forms DataGridView Control](#)

Provides topics that describe custom painting [DataGridView](#) cells and rows, and creating derived cell, column, and row types.

[Performance Tuning in the Windows Forms DataGridView Control](#)

Provides topics that describe how to use the control efficiently to avoid performance problems when working with large amounts of data.

See Also

[DataGridView](#)

[BindingSource](#)

[DataGridView Control](#)

[Default Functionality in the Windows Forms DataGridView Control](#)

[Default Keyboard and Mouse Handling in the Windows Forms DataGridView Control](#)

DataGridView Control Technology Summary (Windows Forms)

5/4/2018 • 2 min to read • [Edit Online](#)

This topic summarizes information about the `DataGridView` control and the classes that support its use.

Displaying data in a tabular format is a task you are likely to perform frequently. The `DataGridView` control is designed to be a complete solution for presenting data in a grid.

Keywords

`DataGridView`, `BindingSource`, table, cell, data binding, virtual mode

Namespaces

`System.Windows.Forms`

`System.Data`

Related Technologies

`BindingSource`

Background

User interface (UI) designers frequently find it necessary to display tabular data to users. The .NET Framework provides several ways to show data in a table or grid. The `DataGridView` control represents the latest evolution of this technology for Windows Forms applications.

The `DataGridView` control can display rows of data from a data store. Many types of data stores are supported. The data store can hold simple, untyped data, such as a one-dimensional array, or it can hold typed data, such as a `DataSet`. For more information, see [How to: Bind Data to the Windows Forms DataGridView Control](#).

The `DataGridView` control provides a powerful and flexible way to display data in a tabular format. You can use the control to show read-only or editable views of small to very large sets of data.

You can extend the `DataGridView` control in several ways to build custom behavior into your applications. For example, you can programmatically specify your own sorting algorithms, and you can create your own types of cells. You can easily customize the appearance of the `DataGridView` control by choosing among several properties. Many types of data stores can be used as a data source, or the `DataGridView` control can operate without a data source bound to it.

Implementing DataGridView Classes

There are several ways for you to take advantage of the `DataGridView` control's extensibility features. You can customize many aspects of the control through events and properties, but some customizations require you to create new classes derived from existing `DataGridView` classes.

The most typically used base classes are `DataGridViewCell` and `DataGridViewColumn`. You can derive your own cell class from `DataGridViewCell` or any of its child classes. Although you can add any cell type to any column, you will typically also derive a companion column class from `DataGridViewColumn` that hosts cells of your custom cell type.

by default.

You can implement the `IDataGridViewEditingCell` interface in your derived cell class to create a cell type that has editing functionality but does not host a control in editing mode. To create a control that you can host in a cell in editing mode, you can implement the `IDataGridViewEditingControl` interface in a class derived from [Control](#).

For more information, see [How to: Customize Cells and Columns in the Windows Forms DataGridView Control by Extending Their Behavior and Appearance](#) and [How to: Host Controls in Windows Forms DataGridView Cells](#).

DataGridView Classes at a Glance

[System.Windows.Forms](#)

TECHNOLOGY AREA	CLASSES/INTERFACES/CONFIGURATION ELEMENTS
Data Binding	BindingSource
Data Presentation	DataGridView DataGridViewCell and derived classes DataGridViewRow and derived classes DataGridViewColumn and derived classes DataGridViewCellStyle
DataGridView Extensibility	DataGridViewCell and derived classes DataGridViewColumn and derived classes IDataGridViewEditingCell IDataGridViewEditingControl

What's New

The [DataGridView](#) control is designed to be a complete solution for displaying tabular data with Windows Forms. You should consider using the [DataGridView](#) control before other solutions, such as [DataGrid](#), when you are authoring a new application. For more information, see [Differences Between the Windows Forms DataGridView and DataGrid Controls](#).

The [DataGridView](#) control can work in close conjunction with the [BindingSource](#) component. This component is designed to be the primary data source of a form. It can manage the interaction between a [DataGridView](#) control and its data source, regardless of the data source type.

See Also

[DataGridView Control Overview](#)
[DataGridView Control Architecture](#)
[Protecting Connection Information](#)

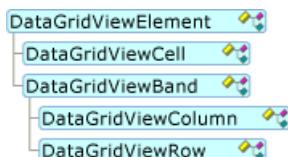
DataGridView Control Architecture (Windows Forms)

5/4/2018 • 3 min to read • [Edit Online](#)

The [DataGridView](#) control and its related classes are designed to be a flexible, extensible system for displaying and editing tabular data. These classes are all contained in the [System.Windows.Forms](#) namespace, and they are all named with the "DataGridView" prefix.

Architecture Elements

The primary [DataGridView](#) companion classes derive from [DataGridViewElement](#). The following object model illustrates the [DataGridViewElement](#) inheritance hierarchy.



DataGridViewElement object model

The [DataGridViewElement](#) class provides a reference to the parent [DataGridView](#) control and has a [State](#) property, which holds a value that represents a combination of values from the [DataGridViewElementStates](#) enumeration.

The following sections describe the [DataGridView](#) companion classes in more detail.

DataGridViewElementStates

The [DataGridViewElementStates](#) enumeration contains the following values:

- [None](#)
- [Frozen](#)
- [ReadOnly](#)
- [Resizable](#)
- [ResizableSet](#)
- [Selected](#)
- [Visible](#)

The values of this enumeration can be combined with the bitwise logical operators, so the [State](#) property can express more than one state at once. For example, a [DataGridViewElement](#) can be simultaneously [Frozen](#), [Selected](#), and [Visible](#).

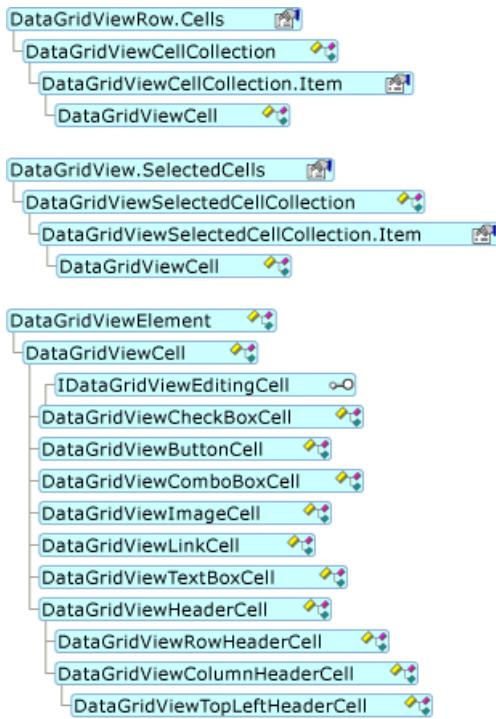
Cells and Bands

The [DataGridView](#) control comprises two fundamental kinds of objects: cells and bands. All cells derive from the [DataGridViewCell](#) base class. The two kinds of bands, [DataGridViewColumn](#) and [DataGridViewRow](#), both derive from the [DataGridViewBand](#) base class.

The [DataGridView](#) control interoperates with several classes, but the most commonly encountered are [DataGridViewCell](#), [DataGridViewColumn](#), and [DataGridViewRow](#).

DataGridViewCell

The cell is the fundamental unit of interaction for the [DataGridView](#). Display is centered on cells, and data entry is often performed through cells. You can access cells by using the [Cells](#) collection of the [DataGridViewRow](#) class, and you can access the selected cells by using the [SelectedCells](#) collection of the [DataGridView](#) control. The following object model illustrates this usage and shows the [DataGridViewCell](#) inheritance hierarchy.



DataGridViewCell object model

The [DataGridViewCell](#) type is an abstract base class, from which all cell types derive. [DataGridViewCell](#) and its derived types are not Windows Forms controls, but some host Windows Forms controls. Any editing functionality supported by a cell is typically handled by a hosted control.

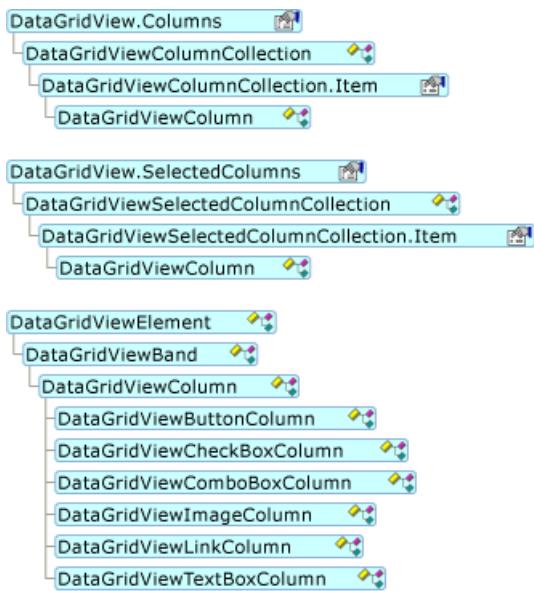
[DataGridViewCell](#) objects do not control their own appearance and painting features in the same way as Windows Forms controls. Instead, the [DataGridView](#) is responsible for the appearance of its [DataGridViewCell](#) objects. You can significantly affect the appearance and behavior of cells by interacting with the [DataGridView](#) control's properties and events. When you have special requirements for customizations that are beyond the capabilities of the [DataGridView](#) control, you can implement your own class that derives from [DataGridViewCell](#) or one of its child classes.

The following list shows the classes derived from [DataGridViewCell](#):

- [DataGridViewTextBoxCell](#)
- [DataGridViewButtonCell](#)
- [DataGridViewLinkCell](#)
- [DataGridViewCheckBoxCell](#)
- [DataGridViewComboBoxCell](#)
- [DataGridViewImageCell](#)
- [DataGridViewHeaderCell](#)
- [DataGridViewRowHeaderCell](#)
- [DataGridViewColumnHeaderCell](#)
- [DataGridViewTopLeftHeaderCell](#)
- Your custom cell types

DataGridViewColumn

The schema of the [DataGridView](#) control's attached data store is expressed in the [DataGridView](#) control's columns. You can access the [DataGridView](#) control's columns by using the [Columns](#) collection. You can access the selected columns by using the [SelectedColumns](#) collection. The following object model illustrates this usage and shows the [DataGridViewColumn](#) inheritance hierarchy.



DataGridViewColumn object model

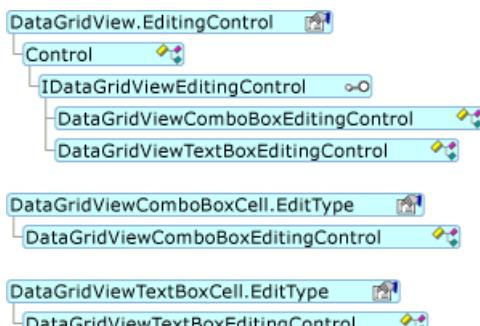
Some of the key cell types have corresponding column types. These are derived from the [DataGridViewColumn](#) base class.

The following list shows the classes derived from [DataGridViewColumn](#):

- [DataGridViewButtonColumn](#)
- [DataGridViewCheckBoxColumn](#)
- [DataGridViewComboBoxColumn](#)
- [DataGridViewImageColumn](#)
- [DataGridViewTextBoxColumn](#)
- [DataGridViewLinkColumn](#)
- Your custom column types

DataGridView Editing Controls

Cells that support advanced editing functionality typically use a hosted control that is derived from a Windows Forms control. These controls also implement the [IDataGridViewEditingControl](#) interface. The following object model illustrates the usage of these controls.



DataGridView editing control object model

The following editing controls are provided with the [DataGridView](#) control:

- [DataGridViewComboBoxEditingControl](#)
- [DataGridViewTextBoxEditingControl](#)

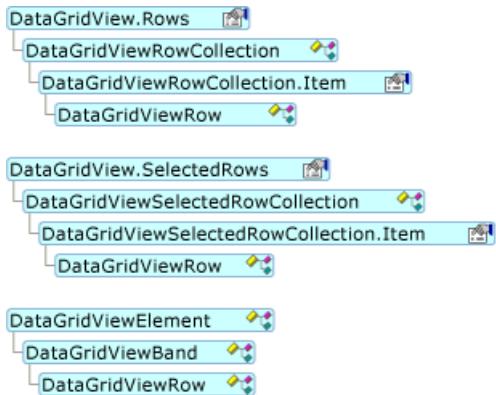
For information about creating your own editing controls, see [How to: Host Controls in Windows Forms DataGridView Cells](#).

The following table illustrates the relationship among cell types, column types, and editing controls.

CELL TYPE	HOSTED CONTROL	COLUMN TYPE
DataGridViewButtonCell	n/a	DataGridViewButtonColumn
DataGridViewCheckBoxCell	n/a	DataGridViewCheckBoxColumn
DataGridViewComboBoxCell	DataGridViewComboBoxEditingControl	DataGridViewComboBoxColumn
DataGridViewImageCell	n/a	DataGridViewImageColumn
DataGridViewLinkCell	n/a	DataGridViewLinkColumn
DataGridViewTextBoxCell	DataGridViewTextBoxEditingControl	DataGridViewTextBoxColumn

DataGridViewRow

The [DataGridViewRow](#) class displays a record's data fields from the data store to which the [DataGridView](#) control is attached. You can access the [DataGridView](#) control's rows by using the [Rows](#) collection. You can access the selected rows by using the [SelectedRows](#) collection. The following object model illustrates this usage and shows the [DataGridViewRow](#) inheritance hierarchy.



DataGridViewRow object model

You can derive your own types from the [DataGridViewRow](#) class, although this will typically not be necessary. The [DataGridView](#) control has several row-related events and properties for customizing the behavior of its [DataGridViewRow](#) objects.

If you enable the [DataGridView](#) control's [AllowUserToAddRows](#) property, a special row for adding new rows appears as the last row. This row is part of the [Rows](#) collection, but it has special functionality that may require your attention. For more information, see [Using the Row for New Records in the Windows Forms DataGridView Control](#).

See Also

[DataGridView Control Overview](#)

[Customizing the Windows Forms DataGridView Control](#)

[Using the Row for New Records in the Windows Forms DataGridView Control](#)

DataGridView Control Scenarios (Windows Forms)

5/4/2018 • 4 min to read • [Edit Online](#)

With the [DataGridView](#) control, you can display tabular data from a variety of data sources. For simple uses, you can manually populate a [DataGridView](#) and manipulate the data directly through the control. Typically, however, you will store your data in an external data source and bind the control to it through a [BindingSource](#) component.

This topic describes some of the common scenarios that involve the [DataGridView](#) control.

Scenario 1: Displaying Small Amounts of Data

You do not have to store your data in an external data source to display it in the [DataGridView](#) control. If you are working with a small amount of data, you can populate the control yourself and manipulate the data through the control. This is called *unbound mode*. For more information, see [How to: Create an Unbound Windows Forms DataGridView Control](#).

Scenario Key Points

- In unbound mode, you populate the control manually.
- Unbound mode is particularly suited for small amounts of read-only data.
- Unbound mode is also suited for spreadsheet-like or sparsely populated tables.

Scenario 2: Viewing and Updating Data Stored in an External Data Source

You can use the [DataGridView](#) control as a user interface (UI) through which users can access data kept in a data source such as a database table or a collection of business objects. For more information, see [How to: Bind Data to the Windows Forms DataGridView Control](#).

Scenario Key Points

- Bound mode lets you connect to a data source, automatically generate columns based on the data source properties or database columns, and automatically populate the control.
- Bound mode is suited for heavy user interaction with data. Data can be formatted for display, and user-specified data can be parsed into the format expected by the data source. Data entry formatting errors and database constraint errors can be detected so that users can be warned and erroneous cells can be corrected.
- Additional functionality such as column sorting, freezing, and reordering enable users to view data in the way most convenient for their workflow.
- Clipboard support enables users to copy data from your application into other applications.

Scenario 3: Advanced Data

If you have special needs that the standard data binding model does not address, you can manage the interaction between the control and your data by implementing *virtual mode*. Implementing virtual mode means implementing one or more event handlers that let the control request information about cells as the information is needed.

For example, if you work with large amounts of data, you may want to implement virtual mode to ensure optimal efficiency. Virtual mode is also useful for maintaining the values of unbound columns that you display along with

columns retrieved from another data source.

For more information about virtual mode, see [Walkthrough: Implementing Virtual Mode in the Windows Forms DataGridView Control](#).

Scenario Key Points

- Virtual mode is suited for displaying very large amounts of data when you need to fine-tune performance.

Scenario 4: Automatically Resizing Rows and Columns

When you display data that is regularly updated, you can automatically resize rows and columns to ensure that all content is visible. The [DataGridView](#) control provides several options that let you enable or disable manual resizing, resize programmatically at specific times, or resize automatically whenever content changes. For more information, see [Sizing Options in the Windows Forms DataGridView Control](#).

Scenario Key Points

- Manual resizing enables users to adjust cell heights and widths.
- Automatic resizing enables you to maintain cell sizes so that cell content is never clipped.
- Programmatic resizing enables you to resize cells at specific times to avoid the performance penalty of continuous automatic resizing.

Scenario 5: Simple Customization

The [DataGridView](#) control provides many ways for you to alter its basic appearance and behavior. For more information, see [Cell Styles in the Windows Forms DataGridView Control](#).

Scenario Key Points

- [DataGridViewCellStyle](#) objects let you provide color, font, formatting, and positioning information at multiple levels and for individual elements of the control.
- Cell styles can be layered and shared by multiple elements, letting you reuse code.

Scenario 6: Advanced Customization

The [DataGridView](#) control provides many ways for you to customize its appearance and behavior.

Scenario Key Points

- You can provide your own cell painting code. For more information, see [How to: Customize the Appearance of Cells in the Windows Forms DataGridView Control](#).
- You can provide your own row painting. This is useful, for example, to create rows with content that spans multiple columns. For more information, see [How to: Customize the Appearance of Rows in the Windows Forms DataGridView Control](#).
- You can implement your own cell and column classes to customize cell appearance. For more information, see [How to: Customize Cells and Columns in the Windows Forms DataGridView Control by Extending Their Behavior and Appearance](#).
- You can implement your own cell and column classes to host controls other than the ones provided by the built-in column types. For more information, see [How to: Host Controls in Windows Forms DataGridView Cells](#).

See Also

[DataGridView](#)

DataGridView Control Overview

DataGridView Control Code Directory (Windows Forms)

5/4/2018 • 3 min to read • [Edit Online](#)

This topic provides links to [DataGridView](#)-related code examples available in the documentation.

NOTE

A link always jumps to the top of the topic in which the code example is found.

Additional code examples are available in the class library reference documentation. For a list of the principal classes and interfaces associated with the [DataGridView](#) control, see the table in [DataGridView Control Technology Summary](#).

CodeList

Unbound Data Examples

- [How to: Add an Unbound Column to a Data-Bound Windows Forms DataGridView Control](#)
- [How to: Create an Unbound Windows Forms DataGridView Control](#)
- [Walkthrough: Creating an Unbound Windows Forms DataGridView Control](#)

Data Binding Examples

- [How to: Bind Data to the Windows Forms DataGridView Control](#)
- [How to: Autogenerate Columns in a Data-Bound Windows Forms DataGridView Control](#)
- [How to: Remove Autogenerated Columns from a Windows Forms DataGridView Control](#)
- [How to: Bind Objects to Windows Forms DataGridView Controls](#)
- [How to: Access Objects Bound to Windows Forms DataGridView Rows](#)
- [How to: Create a Master/Detail Form Using Two Windows Forms DataGridView Controls](#)
- [Walkthrough: Creating a Master/Detail Form Using Two Windows Forms DataGridView Controls](#)

Data Formatting Examples

- [How to: Format Data in the Windows Forms DataGridView Control](#)
- [How to: Customize Data Formatting in the Windows Forms DataGridView Control](#)

Data Validation Examples

- [How to: Validate Data in the Windows Forms DataGridView Control](#)
- [Walkthrough: Validating Data in the Windows Forms DataGridView Control](#)
- [How to: Handle Errors That Occur During Data Entry in the Windows Forms DataGridView Control](#)

- Walkthrough: Handling Errors that Occur During Data Entry in the Windows Forms DataGridView Control

Appearance Customization Examples

- How to: Change the Border and Gridline Styles in the Windows Forms DataGridView Control
- How to: Set Font and Color Styles in the Windows Forms DataGridView Control
- How to: Set Default Cell Styles for the Windows Forms DataGridView Control
- How to: Use the Row Template to Customize Rows in the Windows Forms DataGridView Control
- How to: Set Alternating Row Styles for the Windows Forms DataGridView Control

Behavior Customization Examples

- How to: Specify the Edit Mode for the Windows Forms DataGridView Control
- How to: Specify Default Values for New Rows in the Windows Forms DataGridView Control
- How to: Prevent Row Addition and Deletion in the Windows Forms DataGridView Control
- How to: Perform a Custom Action Based on Changes in a Cell of a Windows Forms DataGridView Control
- How to: Enable Users to Copy Multiple Cells to the Clipboard from the Windows Forms DataGridView Control
- How to: Add ToolTips to Individual Cells in a Windows Forms DataGridView Control
- How to: Display Images in Cells of the Windows Forms DataGridView Control
- How to: Customize Sorting in the Windows Forms DataGridView Control

Column Manipulation Examples

- How to: Freeze Columns in the Windows Forms DataGridView Control
- How to: Enable Column Reordering in the Windows Forms DataGridView Control
- How to: Change the Order of Columns in the Windows Forms DataGridView Control
- How to: Hide Columns in the Windows Forms DataGridView Control
- How to: Hide Column Headers in the Windows Forms DataGridView Control
- How to: Make Columns Read-Only in the Windows Forms DataGridView Control
- How to: Set the Sort Modes for Columns in the Windows Forms DataGridView Control
- How to: Work with Image Columns in the Windows Forms DataGridView Control
- How to: Manipulate Columns in the Windows Forms DataGridView Control

Row and Column Sizing Examples

- Column Fill Mode in the Windows Forms DataGridView Control
- How to: Set the Sizing Modes of the Windows Forms DataGridView Control
- How to: Programmatically Resize Cells to Fit Content in the Windows Forms DataGridView Control
- How to: Automatically Resize Cells When Content Changes in the Windows Forms DataGridView Control

Selection Examples

- [How to: Set the Selection Mode of the Windows Forms DataGridView Control](#)
- [How to: Get the Selected Cells, Rows, and Columns in the Windows Forms DataGridView Control](#)
- [How to: Get and Set the Current Cell in the Windows Forms DataGridView Control](#)

Advanced Customization Examples

- [How to: Customize the Appearance of Cells in the Windows Forms DataGridView Control](#)
- [How to: Customize the Appearance of Rows in the Windows Forms DataGridView Control](#)
- [How to: Customize Cells and Columns in the Windows Forms DataGridView Control by Extending Their Behavior and Appearance](#)
- [How to: Disable Buttons in a Button Column in the Windows Forms DataGridView Control](#)
- [How to: Host Controls in Windows Forms DataGridView Cells](#)

Advanced Data Examples

- [How to: Implement Virtual Mode in the Windows Forms DataGridView Control](#)
- [Walkthrough: Implementing Virtual Mode in the Windows Forms DataGridView Control](#)
- [Implementing Virtual Mode with Just-In-Time Data Loading in the Windows Forms DataGridView Control](#)

See Also

[DataGridView](#)

[DataGridView Control Overview](#)

Default Functionality in the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [DataGridView](#) control provides users with a significant amount of default functionality.

Default Functionality

By default, a [DataGridView](#) control:

- Automatically displays column headers and row headers that remain visible as the table scrolls vertically.
- Has a row header that contains a selection indicator for the current row.
- Has a selection rectangle in the first cell.
- Has columns that can be automatically resized when the user double-clicks the column dividers.
- Automatically supports visual styles on Windows XP and the Windows Server 2003 family when the [EnableVisualStyles](#) method is called from the application's [Main](#) method.

Additionally, the contents of a [DataGridView](#) control can be edited by default:

- If the user double-clicks or presses F2 in a cell, the control automatically puts the cell into edit mode and updates the contents of the cell as the user types.
- If the user scrolls to the end of the grid, the user will see that a row for adding new records is present. When the user clicks this row, a new row is added to the [DataGridView](#) control, with default values. When the user presses ESC, this new row disappears.
- If the user clicks a row header, the whole row is selected.

When you bind a [DataGridView](#) control to a data source by setting its [DataSource](#) property, the control:

- Automatically uses the names of the data source's columns as the column header text.
- Is populated with the contents of the data source. [DataGridView](#) columns are automatically created for each column in the data source.
- Creates a row for each visible row in the table.
- Automatically sorts the rows based on the underlying data when the user clicks a column header.

See Also

[DataGridView](#)

[DataGridView Control](#)

Column Types in the Windows Forms DataGridView Control

5/4/2018 • 5 min to read • [Edit Online](#)

The [DataGridView](#) control uses several column types to display its information and enable users to modify or add information.

When you bind a [DataGridView](#) control and set the [AutoGenerateColumns](#) property to `true`, columns are automatically generated using default column types appropriate for the data types contained in the bound data source.

You can also create instances of any of the column classes yourself and add them to the collection returned by the [Columns](#) property. You can create these instances for use as unbound columns, or you can manually bind them. Manually bound columns are useful, for example, when you want to replace an automatically generated column of one type with a column of another type.

The following table describes the various column classes available for use in the [DataGridView](#) control.

CLASS	DESCRIPTION
DataGridViewTextBoxColumn	Used with text-based values. Generated automatically when binding to numbers and strings.
DataGridViewCheckBoxColumn	Used with Boolean and CheckState values. Generated automatically when binding to values of these types.
DataGridViewImageColumn	Used to display images. Generated automatically when binding to byte arrays, Image objects, or Icon objects.
DataGridViewButtonColumn	Used to display buttons in cells. Not automatically generated when binding. Typically used as unbound columns.
DataGridViewComboBoxColumn	Used to display drop-down lists in cells. Not automatically generated when binding. Typically data-bound manually.
DataGridViewLinkColumn	Used to display links in cells. Not automatically generated when binding. Typically data-bound manually.
Your custom column type	You can create your own column class by inheriting the DataGridViewColumn class or any of its derived classes to provide custom appearance, behavior, or hosted controls. For more information, see How to: Customize Cells and Columns in the Windows Forms DataGridView Control by Extending Their Behavior and Appearance

These column types are described in more detail in the following sections.

[DataGridViewTextBoxColumn](#)

The [DataGridViewTextBoxColumn](#) is a general-purpose column type for use with text-based values such as numbers and strings. In editing mode, a [TextBox](#) control is displayed in the active cell, enabling users to modify the cell value.

Cell values are automatically converted to strings for display. Values entered or modified by the user are automatically parsed to create a cell value of the appropriate data type. You can customize these conversions by handling the [CellFormatting](#) and [CellParsing](#) events of the [DataGridView](#) control.

The cell value data type of a column is specified in the [ValueType](#) property of the column.

DataGridViewCheckBoxColumn

The [DataGridViewCheckBoxColumn](#) is used with [Boolean](#) and [CheckState](#) values. [Boolean](#) values display as two-state or three-state check boxes, depending on the value of the [ThreeState](#) property. When the column is bound to [CheckState](#) values, the [ThreeState](#) property value is `true` by default.

Typically, check box cell values are intended either for storage, like any other data, or for performing bulk operations. If you want to respond immediately when users click a check box cell, you can handle the [CellClick](#) event, but this event occurs before the cell value is updated. If you need the new value at the time of the click, one option is to calculate what the expected value will be based on the current value. Another approach is to commit the change immediately, and handle the [CellValueChanged](#) event to respond to it. To commit the change when the cell is clicked, you must handle the [CurrentCellDirtyStateChanged](#) event. In the handler, if the current cell is a check box cell, call the [CommitEdit](#) method and pass in the [Commit](#) value.

DataGridViewImageColumn

The [DataGridViewImageColumn](#) is used to display images. Image columns can be populated automatically from a data source, populated manually for unbound columns, or populated dynamically in a handler for the [CellFormatting](#) event.

The automatic population of an image column from a data source works with byte arrays in a variety of image formats, including all formats supported by the [Image](#) class and the OLE Picture format used by Microsoft® Access and the Northwind sample database.

Populating an image column manually is useful when you want to provide the functionality of a [DataGridViewButtonColumn](#), but with a customized appearance. You can handle the [DataGridView.CellClick](#) event to respond to clicks within an image cell.

Populating the cells of an image column in a handler for the [CellFormatting](#) event is useful when you want to provide images for calculated values or values in non-image formats. For example, you may have a "Risk" column with string values such as `"high"`, `"middle"`, and `"low"` that you want to display as icons. Alternately, you may have an "Image" column that contains the locations of images that must be loaded rather than the binary content of the images.

DataGridViewButtonColumn

With the [DataGridViewButtonColumn](#), you can display a column of cells that contain buttons. This is useful when you want to provide an easy way for your users to perform actions on particular records, such as placing an order or displaying child records in a separate window.

Button columns are not generated automatically when data-binding a [DataGridView](#) control. To use button columns, you must create them manually and add them to the collection returned by the [DataGridView.Columns](#) property.

You can respond to user clicks in button cells by handling the [DataGridView.CellClick](#) event.

DataGridViewComboBoxColumn

With the [DataGridViewComboBoxColumn](#), you can display a column of cells that contain drop-down list boxes. This is useful for data entry in fields that can only contain particular values, such as the Category column of the

Products table in the Northwind sample database.

You can populate the drop-down list used for all cells the same way you would populate a [ComboBox](#) drop-down list, either manually through the collection returned by the [Items](#) property, or by binding it to a data source through the [DataSource](#), [DisplayMember](#), and [ValueMember](#) properties. For more information, see [ComboBox Control](#).

You can bind the actual cell values to the data source used by the [DataGridView](#) control by setting the [DataPropertyName](#) property of the [System.Windows.Forms.DataGridViewColumn](#).

Combo box columns are not generated automatically when data-binding a [DataGridView](#) control. To use combo box columns, you must create them manually and add them to the collection returned by the [Columns](#) property.

DataGridViewLinkColumn

With the [DataGridViewLinkColumn](#), you can display a column of cells that contain hyperlinks. This is useful for URL values in the data source or as an alternative to the button column for special behaviors such as opening a window with child records.

Link columns are not generated automatically when data-binding a [DataGridView](#) control. To use link columns, you must create them manually and add them to the collection returned by the [Columns](#) property.

You can respond to user clicks on links by handling the [CellContentClick](#) event. This event is distinct from the [CellClick](#) and [CellMouseClick](#) events, which occur when a user clicks anywhere in a cell.

The [DataGridViewLinkColumn](#) class provides several properties for modifying the appearance of links before, during, and after they are clicked.

See Also

[DataGridView](#)

[DataGridViewColumn](#)

[DataGridViewButtonColumn](#)

[DataGridViewCheckBoxColumn](#)

[DataGridViewComboBoxColumn](#)

[DataGridViewImageColumn](#)

[DataGridViewTextBoxColumn](#)

[DataGridViewLinkColumn](#)

[DataGridView Control](#)

[How to: Display Images in Cells of the Windows Forms DataGridView Control](#)

[How to: Work with Image Columns in the Windows Forms DataGridView Control](#)

[Customizing the Windows Forms DataGridView Control](#)

Basic Column, Row, and Cell Features in the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

Many basic behaviors of `DataGridView` cells, rows, and columns can be modified by setting single properties. The topics in this section describe several of the most commonly used of these features.

In This Section

[How to: Hide Columns in the Windows Forms DataGridView Control](#)

Describes how to prevent specific columns from appearing in the control.

[How to: Hide Column Headers in the Windows Forms DataGridView Control](#)

Describes how to prevent the column headers from appearing in the control.

[How to: Enable Column Reordering in the Windows Forms DataGridView Control](#)

Describes how to enable users to rearrange columns in the control.

[How to: Freeze Columns in the Windows Forms DataGridView Control](#)

Describes how prevent one or more adjacent columns from scrolling.

[How to: Make Columns Read-Only in the Windows Forms DataGridView Control](#)

Describes how to prevent users from editing specific columns in the control.

[How to: Prevent Row Addition and Deletion in the Windows Forms DataGridView Control](#)

Describes how to remove the row for new records at the bottom of the control to prevent users from adding rows. Also describes how to prevent users from deleting rows.

[How to: Get and Set the Current Cell in the Windows Forms DataGridView Control](#)

Describes how to access the cell that currently has focus in the control.

[How to: Display Images in Cells of the Windows Forms DataGridView Control](#)

Describes how to create an image column that displays an icon in every cell.

Reference

[DataGridView](#)

Provides reference documentation for the control.

Related Sections

[Basic Formatting and Styling in the Windows Forms DataGridView Control](#)

Provides topics that describe how to modify the basic appearance of the control and the display formatting of cell data.

[Programming with Cells, Rows, and Columns in the Windows Forms DataGridView Control](#)

Provides topics that describe how to program with cell, row, and column objects.

See Also

[DataGridView Control](#)

[Column Types in the Windows Forms DataGridView Control](#)

How to: Hide Columns in the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

Sometimes you will want to display only some of the columns that are available in a Windows Forms [DataGridView](#) control. For example, you might want to show an employee salary column to users with management credentials while hiding it from other users. Alternately, you might want to bind the control to a data source that contains many columns, only some of which you want to display. In this case, you will typically remove the columns you are not interested in displaying rather than hide them.

In the [DataGridView](#) control, the [Visible](#) property value of a column determines whether that column is displayed.

There is support for this task in Visual Studio. Also see [How to: Hide Columns in the Windows Forms DataGridView Control Using the Designer](#).

To hide a column programmatically

- Set the [DataGridViewColumn.Visible](#) property to `false`. To hide a `CustomerID` column that is automatically generated during data binding, place the following code example in a [DataBindingComplete](#) event handler.

```
this.dataGridView1.Columns["CustomerID"].Visible = false;
```

```
Me.dataGridView1.Columns("CustomerID").Visible = False
```

Compiling the Code

This example requires:

- A [DataGridView](#) control named `dataGridView1` that contains a column named `CustomerID`.
- References to the [System](#) and [System.Windows.Forms](#) assemblies.

See Also

[DataGridView](#)

[DataGridViewColumn.Visible](#)

[Basic Column, Row, and Cell Features in the Windows Forms DataGridView Control](#)

[How to: Remove Autogenerated Columns from a Windows Forms DataGridView Control](#)

[How to: Change the Order of Columns in the Windows Forms DataGridView Control](#)

How to: Hide Column Headers in the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

Sometimes you will want to display a [DataGridView](#) without column headers. In the [DataGridView](#) control, the [ColumnHeadersVisible](#) property value determines whether the column headers are displayed.

To hide the column headers

- Set the [DataGridView.ColumnHeadersVisible](#) property to `false`.

```
dataGridView1.ColumnHeadersVisible = false;
```

```
dataGridView1.ColumnHeadersVisible = False
```

Compiling the Code

This example requires:

- A [DataGridView](#) control named `dataGridView1`.
- References to the [System](#) and [System.Windows.Forms](#) assemblies.

See Also

[DataGridView](#)

[DataGridView.ColumnHeadersVisible](#)

[Basic Column, Row, and Cell Features in the Windows Forms DataGridView Control](#)

How to: Enable Column Reordering in the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

When you enable column reordering in the [DataGridView](#) control, users can move a column to a new position by dragging the column header with the mouse. In the [DataGridView](#) control, the [DataGridView.AllowUserToOrderColumns](#) property value determines whether users can move columns to different positions.

There is support for this task in Visual Studio. Also see [How to: Enable Column Reordering in the Windows Forms DataGridView Control Using the Designer](#)

To enable column reordering programmatically

- Set the [DataGridView.AllowUserToOrderColumns](#) property to `true`.

```
dataGridView1.AllowUserToOrderColumns = true;
```

```
dataGridView1.AllowUserToOrderColumns = True
```

Compiling the Code

This example requires:

- A [DataGridView](#) control named `dataGridView1`.
- References to the [System](#) and [System.Windows.Forms](#) assemblies.

See Also

[DataGridView](#)

[DataGridView.AllowUserToOrderColumns](#)

[Basic Column, Row, and Cell Features in the Windows Forms DataGridView Control](#)

[How to: Freeze Columns in the Windows Forms DataGridView Control](#)

How to: Freeze Columns in the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

When users view data displayed in a Windows Forms [DataGridView](#) control, they sometimes need to refer to a single column or set of columns frequently. For example, when displaying a table of customer information that contains many columns, it is useful to display the customer name at all times while enabling other columns to scroll outside the visible region.

To achieve this behavior, you can freeze columns in the control. When you freeze a column, all the columns to its left (or to its right in right-to-left language scripts) are frozen as well. Frozen columns remain in place while all other columns can scroll.

NOTE

If column reordering is enabled, the frozen columns are treated as a group distinct from the unfrozen columns. Users can reposition columns in either group, but they cannot move a column from one group to the other.

The [Frozen](#) property of a column determines whether the column is always visible within the grid.

There is support for this task in Visual Studio. Also see [How to: Freeze Columns in the Windows Forms DataGridView Control Using the Designer](#).

To freeze a column programmatically

- Set the [DataGridViewColumn.Frozen](#) property to `true`.

```
this.dataGridView1.Columns["AddToCartButton"].Frozen = true;
```

```
Me.dataGridView1.Columns("AddToCartButton").Frozen = True
```

Compiling the Code

This example requires:

- A [DataGridView](#) control named `dataGridView1` that contains a column named `AddToCartButton`.
- References to the [System](#) and [System.Windows.Forms](#) assemblies.

See Also

[DataGridViewColumn.Frozen](#)

[DataGridView](#)

[Basic Column, Row, and Cell Features in the Windows Forms DataGridView Control](#)

[How to: Enable Column Reordering in the Windows Forms DataGridView Control](#)

How to: Make Columns Read-Only in the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

Not all data is meant for editing. In the [DataGridView](#) control, the column [ReadOnly](#) property value determines whether users can edit cells in that column. For information about how to make the control entirely read-only, see [How to: Prevent Row Addition and Deletion in the Windows Forms DataGridView Control](#).

There is support for this task in Visual Studio. Also see [How to: Make Columns Read-Only in the Windows Forms DataGridView Control Using the Designer](#).

To make a column read-only programmatically

- Set the [DataGridViewColumn.ReadOnly](#) property to `true`.

```
dataGridView1.Columns["CompanyName"].ReadOnly = true;
```

```
dataGridView1.Columns("CompanyName").ReadOnly = True
```

Compiling the Code

This example requires:

- A [DataGridView](#) control named `dataGridView1` with a column named `CompanyName`.
- References to the [System](#) and [System.Windows.Forms](#) assemblies.

See Also

[DataGridView](#)

[DataGridView.Columns](#)

[DataGridViewColumn.ReadOnly](#)

[Basic Column, Row, and Cell Features in the Windows Forms DataGridView Control](#)

[How to: Prevent Row Addition and Deletion in the Windows Forms DataGridView Control](#)

How to: Prevent Row Addition and Deletion in the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

Sometimes you will want to prevent users from entering new rows of data or deleting existing rows in your [DataGridView](#) control. The [AllowUserToAddRows](#) property indicates whether the row for new records is present at the bottom of the control, while the [AllowUserToDeleteRows](#) property indicates whether rows can be removed. The following code example uses these properties and also sets the [ReadOnly](#) property to make the control entirely read-only.

There is support for this task in Visual Studio. Also see [How to: Prevent Row Addition and Deletion in the Windows Forms DataGridView Control Using the Designer](#).

Example

```
private void MakeReadOnly()
{
    dataGridView1.AllowUserToAddRows = false;
    dataGridView1.AllowUserToDeleteRows = false;
    dataGridView1.ReadOnly = true;
}
```

```
Private Sub MakeReadOnly()

    With dataGridView1
        .AllowUserToAddRows = False
        .AllowUserToDeleteRows = False
        .ReadOnly = True
    End With

End Sub
```

Compiling the Code

This example requires:

- A [DataGridView](#) control named `dataGridView1`.
- References to the [System](#) and [System.Windows.Forms](#) assemblies.

See Also

[DataGridView](#)
[DataGridView.AllowUserToAddRows](#)
[DataGridView.ReadOnly](#)
[DataGridView.AllowUserToAddRows](#)
[DataGridView.AllowUserToDeleteRows](#)
[Basic Column, Row, and Cell Features in the Windows Forms DataGridView Control](#)

How to: Get and Set the Current Cell in the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

Interaction with the [DataGridView](#) often requires that you programmatically discover which cell is currently active. You may also need to change the current cell. You can perform these tasks with the [CurrentCell](#) property.

NOTE

You cannot set the current cell in a row or column that has its [Visible](#) property set to `false`.

Depending on the [DataGridView](#) control's selection mode, changing the current cell can change the selection. For more information, see [Selection Modes in the Windows Forms DataGridView Control](#).

To get the current cell programmatically

- Use the [DataGridView](#) control's [CurrentCell](#) property.

```
private void getCurrentCellButton_Click(object sender, System.EventArgs e)
{
    string msg = String.Format("Row: {0}, Column: {1}",
        dataGridView1.CurrentCell.RowIndex,
        dataGridView1.CurrentCell.ColumnIndex);
    MessageBox.Show(msg, "Current Cell");
}
```

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

    Dim msg As String = String.Format("Row: {0}, Column: {1}", _
        dataGridView1.CurrentCell.RowIndex, _
        dataGridView1.CurrentCell.ColumnIndex)
    MessageBox.Show(msg, "Current Cell")

End Sub
```

To set the current cell programmatically

- Set the [CurrentCell](#) property of the [DataGridView](#) control. In the following code example, the current cell is set to row 0, column 1.

```
private void setCurrentCellButton_Click(object sender, System.EventArgs e)
{
    // Set the current cell to the cell in column 1, Row 0.
    this.dataGridView1.CurrentCell = this.dataGridView1[1,0];
}
```

```
Private Sub setCurrentCellButton_Click(ByVal sender As Object, _  
    ByVal e As System.EventArgs) Handles setCurrentCellButton.Click  
  
    ' Set the current cell to the cell in column 1, Row 0.  
    Me.dataGridView1.CurrentCell = Me.dataGridView1(1, 0)  
  
End Sub
```

Compiling the Code

This example requires:

- **Button** controls named `getCurrentCellButton` and `setCurrentCellButton`. In Visual C#, you must attach the `Click` events for each button to the associated event handler in the example code.
- A **DataGridView** control named `dataGridView1`.
- References to the **System** and **System.Windows.Forms** assemblies.

See Also

[DataGridView](#)

[DataGridView.CurrentCell](#)

[Basic Column, Row, and Cell Features in the Windows Forms DataGridView Control](#)

[Selection Modes in the Windows Forms DataGridView Control](#)

How to: Display Images in Cells of the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

A picture or graphic is one of the values that you can display in a row of data. Frequently, these graphics take the form of an employee's photograph or a company logo.

Incorporating pictures is simple when you display data within the [DataGridView](#) control. The [DataGridView](#) control natively handles any image format supported by the [Image](#) class, as well as the OLE picture format used by some databases.

If the [DataGridView](#) control's data source has a column of images, they will be displayed automatically by the [DataGridView](#) control.

The following code example demonstrates how to extract an icon from an embedded resource and convert it to a bitmap for display in every cell of an image column. For another example that replaces textual cell values with corresponding images, see [How to: Customize Data Formatting in the Windows Forms DataGridView Control](#).

Example

```
private void createGraphicsColumn()
{
    Icon treeIcon = new Icon(this.GetType(), "tree.ico");
    DataGridViewImageColumn iconColumn = new DataGridViewImageColumn();
    iconColumn.Image = treeIcon.ToBitmap();
    iconColumn.Name = "Tree";
    iconColumn.HeaderText = "Nice tree";
    dataGridView1.Columns.Insert(2, iconColumn);
}
```

```
Public Sub CreateGraphicsColumn()

    Dim treeIcon As New Icon(Me.GetType(), "tree.ico")
    Dim iconColumn As New DataGridViewImageColumn()

    With iconColumn
        .Image = treeIcon.ToBitmap()
        .Name = "Tree"
        .HeaderText = "Nice tree"
    End With

    dataGridView1.Columns.Insert(2, iconColumn)

End Sub
```

Compiling the Code

This example requires:

- A [DataGridView](#) control named `dataGridView1`.
- An embedded icon resource named `tree.ico`.

- References to the [System](#), [System.Windows.Forms](#), and [System.Drawing](#) assemblies.

See Also

[DataGridView](#)

[Basic Column, Row, and Cell Features in the Windows Forms DataGridView Control](#)

[How to: Customize Data Formatting in the Windows Forms DataGridView Control](#)

Basic Formatting and Styling in the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

The `DataGridView` control makes it easy to define the basic appearance of cells and the display formatting of cell values. You can define appearance and formatting styles for individual cells, for cells in specific columns and rows, or for all cells in the control by setting the properties of the `DataGridViewCellStyle` objects accessed through various `DataGridView` control properties. Additionally, you can modify these styles dynamically based on factors such as the cell value by handling the `CellFormatting` event.

In This Section

[How to: Change the Border and Gridline Styles in the Windows Forms DataGridView Control](#)

Describes how to set `DataGridView` properties that define the appearance of the control border and the boundary lines between cells.

[Cell Styles in the Windows Forms DataGridView Control](#)

Describes the `DataGridViewCellStyle` class and how properties of that type interact to define how cells are displayed in the control.

[How to: Set Default Cell Styles for the Windows Forms DataGridView Control](#)

Describes how to use `DataGridViewCellStyle` properties to define the default appearance of cells in specific rows and columns and in the entire control.

[How to: Format Data in the Windows Forms DataGridView Control](#)

Describes how to format cell display values using `DataGridViewCellStyle` properties.

[How to: Set Font and Color Styles in the Windows Forms DataGridView Control](#)

Describes how to use the `DefaultCellStyle` property to set basic display characteristics for all cells in the control.

[How to: Set Alternating Row Styles for the Windows Forms DataGridView Control](#)

Describes how to create a ledger-like effect in the control using alternating rows that are displayed differently.

[How to: Use the Row Template to Customize Rows in the Windows Forms DataGridView Control](#)

Describes how to use the `RowTemplate` property to set row properties that will be used for all rows in the control.

Reference

[DataGridView](#)

Provides reference documentation for the `DataGridView` control.

[DataGridViewCellStyle](#)

Provides reference documentation for the `DataGridViewCellStyle` class.

[CellFormatting](#)

Provides reference documentation for the `CellFormatting` event.

[RowTemplate](#)

Provides reference documentation for the `RowTemplate` property.

Related Sections

[Customizing the Windows Forms DataGridView Control](#)

Provides topics that describe custom painting [DataGridView](#) cells and rows, and creating derived cell, column, and row types.

[Basic Column, Row, and Cell Features in the Windows Forms DataGridView Control](#)

Provides topics that describe commonly used cell, row, and column properties.

See Also

[DataGridView Control](#)

How to: Change the Border and Gridline Styles in the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

With the [DataGridView](#) control, you can customize the appearance of the control's border and gridlines to improve the user experience. You can modify the gridline color and the control border style in addition to the border styles for the cells within the control. You can also apply different cell border styles for ordinary cells, row header cells, and column header cells.

NOTE

The gridline color is used only with the [Single](#), [SingleHorizontal](#), and [SingleVertical](#) values of the [DataGridViewCellBorderStyle](#) enumeration and the [Single](#) value of the [DataGridViewHeaderBorderStyle](#) enumeration. The other values of these enumerations use colors specified by the operating system. Additionally, when visual styles are enabled on Windows XP and the Windows Server 2003 family through the [Application.EnableVisualStyles](#) method, the [GridColor](#) property value is not used.

To change the gridline color programmatically

- Set the [GridColor](#) property.

```
this.dataGridView1.GridColor = Color.BlueViolet;
```

```
Me.dataGridView1.GridColor = Color.BlueViolet
```

To change the border style of the entire DataGridView control programmatically

- Set the [BorderStyle](#) property to one of the [BorderStyle](#) enumeration values.

```
this.dataGridView1.BorderStyle = BorderStyle.Fixed3D;
```

```
Me.dataGridView1.BorderStyle = BorderStyle.Fixed3D
```

To change the border styles for DataGridView cells programmatically

- Set the [CellBorderStyle](#), [RowHeadersBorderStyle](#), and [ColumnHeadersBorderStyle](#) properties.

```
this.dataGridView1.CellBorderStyle =
    DataGridViewCellBorderStyle.None;
this.dataGridView1.RowHeadersBorderStyle =
    DataGridViewHeaderBorderStyle.Single;
this.dataGridView1.ColumnHeadersBorderStyle =
    DataGridViewHeaderBorderStyle.Single;
```

```

With Me.dataGridView1
    .CellBorderStyle = DataGridViewCellBorderStyle.None
    .RowHeadersBorderStyle = _
        DataGridViewHeaderBorderStyle.Single
    .ColumnHeadersBorderStyle = _
        DataGridViewHeaderBorderStyle.Single
End With

```

Example

```

private void SetBorderAndGridlineStyles()
{
    this.dataGridView1.GridColor = Color.BlueViolet;
    this.dataGridView1.BorderStyle = BorderStyle.Fixed3D;
    this.dataGridView1.CellBorderStyle =
        DataGridViewCellBorderStyle.None;
    this.dataGridView1.RowHeadersBorderStyle =
        DataGridViewHeaderBorderStyle.Single;
    this.dataGridView1.ColumnHeadersBorderStyle =
        DataGridViewHeaderBorderStyle.Single;
}

```

```

Private Sub SetBorderAndGridlineStyles()

    With Me.dataGridView1
        .GridColor = Color.BlueViolet
        .BorderStyle = BorderStyle.Fixed3D
        .CellBorderStyle = DataGridViewCellBorderStyle.None
        .RowHeadersBorderStyle = _
            DataGridViewHeaderBorderStyle.Single
        .ColumnHeadersBorderStyle = _
            DataGridViewHeaderBorderStyle.Single
    End With

End Sub

```

Compiling the Code

This example requires:

- A [DataGridView](#) control named `dataGridView1`.
- References to the [System](#), [System.Windows.Forms](#), and [System.Drawing](#) assemblies.

See Also

[BorderStyle](#)

[DataGridView.BorderStyle](#)

[DataGridView.CellBorderStyle](#)

[DataGridView.ColumnHeadersBorderStyle](#)

[DataGridView.GridColor](#)

[DataGridView.RowHeadersBorderStyle](#)

[DataGridViewCellBorderStyle](#)

[DataGridViewHeaderBorderStyle](#)

[Basic Formatting and Styling in the Windows Forms DataGridView Control](#)

Cell Styles in the Windows Forms DataGridView Control

5/4/2018 • 6 min to read • [Edit Online](#)

Each cell within the [DataGridView](#) control can have its own style, such as text format, background color, foreground color, and font. Typically, however, multiple cells will share particular style characteristics.

Groups of cells that share styles may include all cells within particular rows or columns, all cells that contain particular values, or all cells in the control. Because these groups overlap, each cell may get its styling information from more than one place. For example, you may want every cell in a [DataGridView](#) control to use the same font, but only cells in currency columns to use currency format, and only currency cells with negative numbers to use a red foreground color.

The DataGridViewCellStyle Class

The [DataGridViewCellStyle](#) class contains the following properties related to visual style:

- [BackColor](#) and [ForeColor](#)
- [SelectionBackColor](#) and [SelectionForeColor](#)
- [Font](#)

This class also contains the following properties related to formatting:

- [Format](#) and [FormatProvider](#)
- [NullValue](#) and [DataSourceNullValue](#)
- [WrapMode](#)
- [Alignment](#)
- [Padding](#)

For more information on these properties and other cell-style properties, see the [DataGridViewCellStyle](#) reference documentation and the topics listed in the See Also section below.

Using DataGridViewCellStyle Objects

You can retrieve [DataGridViewCellStyle](#) objects from various properties of the [DataGridView](#), [DataGridViewColumn](#), [DataGridViewRow](#), and [DataGridViewCell](#) classes and their derived classes. If one of these properties has not yet been set, retrieving its value will create a new [DataGridViewCellStyle](#) object. You can also instantiate your own [DataGridViewCellStyle](#) objects and assign them to these properties.

You can avoid unnecessary duplication of style information by sharing [DataGridViewCellStyle](#) objects among multiple [DataGridView](#) elements. Because the styles set at the control, column, and row levels filter down through each level to the cell level, you can also avoid style duplication by setting only those style properties at each level that differ from the levels above. This is described in more detail in the Style Inheritance section that follows.

The following table lists the primary properties that get or set [DataGridViewCellStyle](#) objects.

PROPERTY	CLASSES	DESCRIPTION
<code>DefaultCellStyle</code>	<code>DataGridView</code> , <code>DataGridViewColumn</code> , <code>DataGridViewRow</code> , and derived classes	Gets or sets default styles used by all cells in the entire control (including header cells), in a column, or in a row.
<code>RowsDefaultCellStyle</code>	<code>DataGridView</code>	Gets or sets default cell styles used by all rows in the control. This does not include header cells.
<code>AlternatingRowsDefaultCellStyle</code>	<code>DataGridView</code>	Gets or sets default cell styles used by alternating rows in the control. Used to create a ledger-like effect.
<code>RowHeadersDefaultCellStyle</code>	<code>DataGridView</code>	Gets or sets default cell styles used by the control's row headers. Overridden by the current theme if visual styles are enabled.
<code>ColumnHeadersDefaultCellStyle</code>	<code>DataGridView</code>	Gets or sets default cell styles used by the control's column headers. Overridden by the current theme if visual styles are enabled.
<code>Style</code>	<code>DataGridViewCell</code> and derived classes	Gets or sets styles specified at the cell level. These styles override those inherited from higher levels.
<code>InheritedStyle</code>	<code>DataGridViewCell</code> , <code>DataGridViewRow</code> , <code>DataGridViewColumn</code> , and derived classes	Gets all the styles currently applied to the cell, row, or column, including styles inherited from higher levels.

As mentioned above, getting the value of a style property automatically instantiates a new `DataGridViewCellStyle` object if the property has not been previously set. To avoid creating these objects unnecessarily, the row and column classes have a `HasDefaultCellStyle` property that you can check to determine whether the `DefaultCellStyle` property has been set. Similarly, the cell classes have a `HasStyle` property that indicates whether the `Style` property has been set.

Each of the style properties has a corresponding `PropertyNameChanged` event on the `DataGridView` control. For row, column, and cell properties, the name of the event begins with "`Row`", "`Column`", or "`Cell`" (for example, `RowDefaultCellStyleChanged`). Each of these events occurs when the corresponding style property is set to a different `DataGridViewCellStyle` object. These events do not occur when you retrieve a `DataGridViewCellStyle` object from a style property and modify its property values. To respond to changes to the cell style objects themselves, handle the `CellStyleContentChanged` event.

Style Inheritance

Each `DataGridViewCell` gets its appearance from its `InheritedStyle` property. The `DataGridViewCellStyle` object returned by this property inherits its values from a hierarchy of properties of type `DataGridViewCellStyle`. These properties are listed below in the order in which the `InheritedStyle` for non-header cells obtains its values.

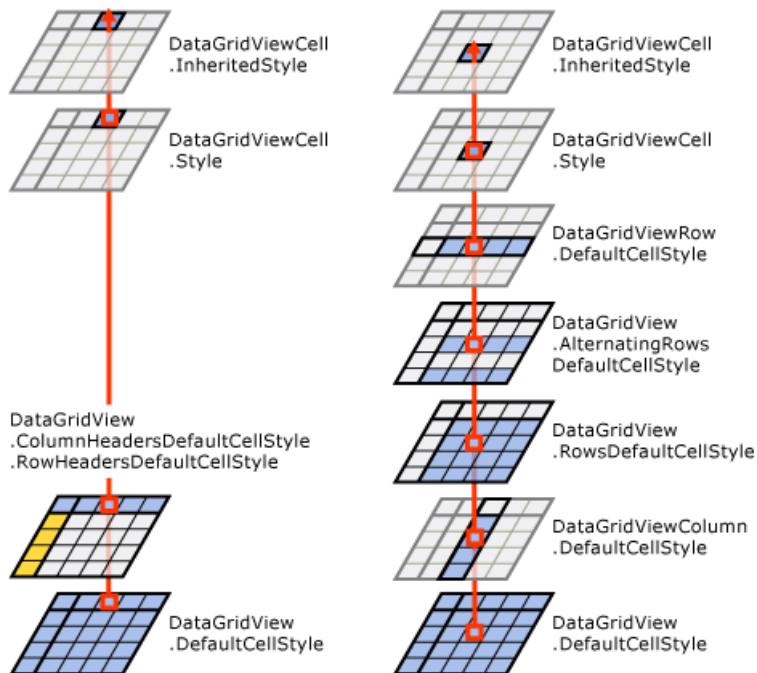
1. `DataGridViewCell.Style`
2. `DataGridViewRow.DefaultCellStyle`
3. `DataGridView.AlternatingRowsDefaultCellStyle` (only for cells in rows with odd index numbers)

4. [DataGridView.RowsDefaultCellStyle](#)
5. [DataGridViewColumn.DefaultCellStyle](#)
6. [DataGridView.DefaultCellStyle](#)

For row and column header cells, the [InheritedStyle](#) property is populated by values from the following list of source properties in the given order.

1. [DataGridViewCell.Style](#)
2. [DataGridView.ColumnHeadersDefaultCellStyle](#) or [DataGridView.RowHeadersDefaultCellStyle](#)
3. [DataGridView.DefaultCellStyle](#)

The following diagram illustrates this process.



You can also access the styles inherited by specific rows and columns. The column [InheritedStyle](#) property inherits its values from the following properties.

1. [DataGridViewColumn.DefaultCellStyle](#)
2. [DataGridView.DefaultCellStyle](#)

The row [InheritedStyle](#) property inherits its values from the following properties.

1. [DataGridViewRow.DefaultCellStyle](#)
2. [DataGridView.AlternatingRowsDefaultCellStyle](#) (only for cells in rows with odd index numbers)
3. [DataGridView.RowsDefaultCellStyle](#)
4. [DataGridView.DefaultCellStyle](#)

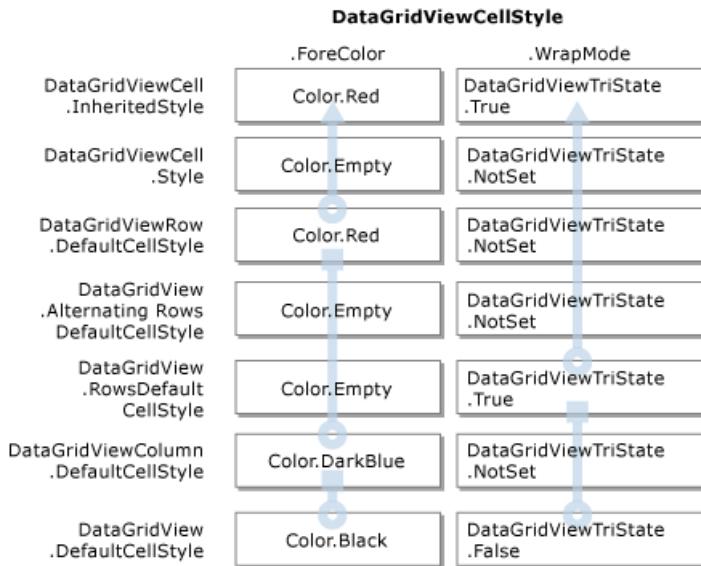
For each property in a [DataGridViewCellStyle](#) object returned by an [InheritedStyle](#) property, the property value is obtained from the first cell style in the appropriate list that has the corresponding property set to a value other than the [DataGridViewCellStyle](#) class defaults.

The following table illustrates how the [ForeColor](#) property value for an example cell is inherited from its containing column.

PROPERTY OF TYPE DATAGRIDVIEWCELLSTYLE	EXAMPLE FORECOLOR VALUE FOR RETRIEVED OBJECT
DataGridViewCell.Style	Color.Empty
DataGridViewRow.DefaultCellStyle	Color.Red
DataGridView.AlternatingRowsDefaultCellStyle	Color.Empty
DataGridView.RowsDefaultCellStyle	Color.Empty
DataGridViewColumn.DefaultCellStyle	Color.DarkBlue
DataGridView.DefaultCellStyle	Color.Black

In this case, the [Color.Red](#) value from the cell's row is the first real value on the list. This becomes the [ForeColor](#) property value of the cell's [InheritedStyle](#).

The following diagram illustrates how different [DataGridViewCellStyle](#) properties can inherit their values from different places.



By taking advantage of style inheritance, you can provide appropriate styles for the entire control without having to specify the same information in multiple places.

Although header cells participate in style inheritance as described, the objects returned by the `ColumnHeadersDefaultCellStyle` and `RowHeadersDefaultCellStyle` properties of the `DataGridView` control have initial property values that override the property values of the object returned by the `DefaultCellStyle` property. If you want the properties set for the object returned by the `DefaultCellStyle` property to apply to row and column headers, you must set the corresponding properties of the objects returned by the `ColumnHeadersDefaultCellStyle` and `RowHeadersDefaultCellStyle` properties to the defaults indicated for the `DataGridViewCellStyle` class.

NOTE

If visual styles are enabled, the row and column headers (except for the `TopLeftHeaderCell`) are automatically styled by the current theme, overriding any styles specified by these properties.

The `DataGridViewButtonColumn`, `DataGridViewImageColumn`, and `DataGridViewCheckBoxColumn` types also initialize some values of the object returned by the column `DefaultCellStyle` property. For more information, see

the reference documentation for these types.

Setting Styles Dynamically

To customize the styles of cells with particular values, implement a handler for the [DataGridView.CellFormatting](#) event. Handlers for this event receive an argument of the [DataGridViewCellFormattingEventArgs](#) type. This object contains properties that let you determine the value of the cell being formatted along with its location in the [DataGridView](#) control. This object also contains a [CellStyle](#) property that is initialized to the value of the [InheritedStyle](#) property of the cell being formatted. You can modify the cell style properties to specify style information appropriate to the cell value and location.

NOTE

The [RowPrePaint](#) and [RowPostPaint](#) events also receive a [DataGridViewCellStyle](#) object in the event data, but in their case, it is a copy of the row [InheritedStyle](#) property for read-only purposes, and changes to it do not affect the control.

You can also dynamically modify the styles of individual cells in response to events such as the [DataGridView.CellMouseEnter](#) and [CellMouseLeave](#) events. For example, in a handler for the [CellMouseEnter](#) event, you could store the current value of the cell background color (retrieved through the cell's [Style](#) property), then set it to a new color that will highlight the cell when the mouse hovers over it. In a handler for the [CellMouseLeave](#) event, you can then restore the background color to the original value.

NOTE

Caching the values stored in the cell's [Style](#) property is important regardless of whether a particular style value is set. If you temporarily replace a style setting, restoring it to its original "not set" state ensures that the cell will go back to inheriting the style setting from a higher level. If you need to determine the actual style in effect for a cell regardless of whether the style is inherited, use the cell's [InheritedStyle](#) property.

See Also

- [DataGridView](#)
- [DataGridViewCellStyle](#)
- [DataGridView.AlternatingRowsDefaultCellStyle](#)
- [DataGridView.ColumnHeadersDefaultCellStyle](#)
- [DataGridView.DefaultCellStyle](#)
- [DataGridView.RowHeadersDefaultCellStyle](#)
- [DataGridView.RowsDefaultCellStyle](#)
- [DataGridViewBand.InheritedStyle](#)
- [DataGridViewRow.InheritedStyle](#)
- [DataGridViewColumn.InheritedStyle](#)
- [DataGridViewBand.DefaultCellStyle](#)
- [DataGridViewCell.InheritedStyle](#)
- [DataGridViewCell.Style](#)
- [DataGridView.CellFormatting](#)
- [DataGridView.CellStyleContentChanged](#)
- [DataGridView.RowPrePaint](#)
- [DataGridView.RowPostPaint](#)
- [Basic Formatting and Styling in the Windows Forms DataGridView Control](#)
- [How to: Set Default Cell Styles for the Windows Forms DataGridView Control](#)
- [Data Formatting in the Windows Forms DataGridView Control](#)

How to: Set Default Cell Styles for the Windows Forms DataGridView Control

5/4/2018 • 2 min to read • [Edit Online](#)

With the [DataGridView](#) control, you can specify default cell styles for the entire control and for specific columns and rows. These defaults filter down from the control level to the column level, then to the row level, then to the cell level. If a particular [DataGridViewCellStyle](#) property is not set at the cell level, the default property setting at the row level is used. If the property is also not set at the row level, the default column setting is used. Finally, if the property is also not set at the column level, the default [DataGridView](#) setting is used. With this setting, you can avoid having to duplicate the property settings at multiple levels. At each level, simply specify the styles that differ from the levels above it. For more information, see [Cell Styles in the Windows Forms DataGridView Control](#).

There is extensive support for this task in Visual Studio. Also see [How to: Set Default Cell Styles and Data Formats for the Windows Forms DataGridView Control Using the Designer](#).

To set the default cell styles programmatically

1. Set the properties of the [DataGridViewCellStyle](#) retrieved through the [DataGridView.DefaultCellStyle](#) property.

```
this.dataGridView1.DefaultCellStyle.BackColor = Color.Beige;  
this.dataGridView1.DefaultCellStyle.Font = new Font("Tahoma", 12);
```

```
Me.dataGridView1.DefaultCellStyle.BackColor = Color.Beige  
Me.dataGridView1.DefaultCellStyle.Font = New Font("Tahoma", 12)
```

2. Create and initialize new [DataGridViewCellStyle](#) objects for use by multiple rows and columns.

```
DataGridViewCellStyle highlightCellStyle = new DataGridViewCellStyle();  
highlightCellStyle.BackColor = Color.Red;  
  
DataGridViewCellStyle currencyCellStyle = new DataGridViewCellStyle();  
currencyCellStyle.Format = "C";  
currencyCellStyle.ForeColor = Color.Green;
```

```
Dim highlightCellStyle As New DataGridViewCellStyle()  
highlightCellStyle.BackColor = Color.Red  
  
Dim currencyCellStyle As New DataGridViewCellStyle()  
currencyCellStyle.Format = "C"  
currencyCellStyle.ForeColor = Color.Green
```

3. Set the [DefaultCellStyle](#) property of specific rows and columns.

```

this.dataGridView1.Rows[3].DefaultCellStyle = highlightCellStyle;
this.dataGridView1.Rows[8].DefaultCellStyle = highlightCellStyle;
this.dataGridView1.Columns["UnitPrice"].DefaultCellStyle =
    currencyCellStyle;
this.dataGridView1.Columns["TotalPrice"].DefaultCellStyle =
    currencyCellStyle;

```

```

With Me.dataGridView1
    .Rows(3).DefaultCellStyle = highlightCellStyle
    .Rows(8).DefaultCellStyle = highlightCellStyle
    .Columns("UnitPrice").DefaultCellStyle = currencyCellStyle
    .Columns("TotalPrice").DefaultCellStyle = currencyCellStyle
End With

```

Example

```

private void SetDefaultCellStyles()
{
    this.dataGridView1.DefaultCellStyle.BackColor = Color.Beige;
    this.dataGridView1.DefaultCellStyle.Font = new Font("Tahoma", 12);

    DataGridViewCellStyle highlightCellStyle = new DataGridViewCellStyle();
    highlightCellStyle.BackColor = Color.Red;

    DataGridViewCellStyle currencyCellStyle = new DataGridViewCellStyle();
    currencyCellStyle.Format = "C";
    currencyCellStyle.ForeColor = Color.Green;

    this.dataGridView1.Rows[3].DefaultCellStyle = highlightCellStyle;
    this.dataGridView1.Rows[8].DefaultCellStyle = highlightCellStyle;
    this.dataGridView1.Columns["UnitPrice"].DefaultCellStyle =
        currencyCellStyle;
    this.dataGridView1.Columns["TotalPrice"].DefaultCellStyle =
        currencyCellStyle;
}

```

```

Private Sub SetDefaultCellStyles()

    Dim highlightCellStyle As New DataGridViewCellStyle
    highlightCellStyle.BackColor = Color.Red

    Dim currencyCellStyle As New DataGridViewCellStyle
    currencyCellStyle.Format = "C"
    currencyCellStyle.ForeColor = Color.Green

    With Me.dataGridView1
        .DefaultCellStyle.BackColor = Color.Beige
        .DefaultCellStyle.Font = New Font("Tahoma", 12)
        .Rows(3).DefaultCellStyle = highlightCellStyle
        .Rows(8).DefaultCellStyle = highlightCellStyle
        .Columns("UnitPrice").DefaultCellStyle = currencyCellStyle
        .Columns("TotalPrice").DefaultCellStyle = currencyCellStyle
    End With

End Sub

```

Compiling the Code

This example requires:

- A [DataGridView](#) control named `dataGridView1`.
- References to the [System](#), [System.Drawing](#), and [System.Windows.Forms](#) assemblies.

Robust Programming

To achieve maximum scalability when you work with very large data sets, you should share [DataGridViewCellStyle](#) objects across multiple rows, columns, or cells that use the same styles, rather than set the style properties for individual elements separately. Additionally, you should create shared rows and access them by using the [DataGridViewRowCollection.SharedRow](#) property. For more information, see [Best Practices for Scaling the Windows Forms DataGridView Control](#).

See Also

[DataGridView](#)

[DataGridViewCellStyle](#)

[DataGridView.DefaultCellStyle](#)

[DataGridViewBand.DefaultCellStyle](#)

[Basic Formatting and Styling in the Windows Forms DataGridView Control](#)

[Cell Styles in the Windows Forms DataGridView Control](#)

[Best Practices for Scaling the Windows Forms DataGridView Control](#)

[How to: Set Alternating Row Styles for the Windows Forms DataGridView Control](#)

How to: Format Data in the Windows Forms DataGridView Control

5/4/2018 • 2 min to read • [Edit Online](#)

The following procedures demonstrate basic formatting of cell values using the `DefaultCellStyle` property of a `DataGridView` control and of specific columns in a control. For information about advanced data formatting, see [How to: Customize Data Formatting in the Windows Forms DataGridView Control](#).

To format currency and date values

- Set the `Format` property of a `DataGridViewCellStyle`. The following code example sets the format for specific columns using the `DefaultCellStyle` property of the columns. Values in the `UnitPrice` column appear in the current culture-specific currency format, with negative values surrounded by parentheses. Values in the `ShipDate` column appear in the current culture-specific short date format. For more information about `Format` values, see [Formatting Types](#).

```
this.dataGridView1.Columns["UnitPrice"].DefaultCellStyle.Format = "c";
this.dataGridView1.Columns["ShipDate"].DefaultCellStyle.Format = "d";
```

```
Me.dataGridView1.Columns("UnitPrice").DefaultCellStyle.Format = "c"
Me.dataGridView1.Columns("ShipDate").DefaultCellStyle.Format = "d"
```

To customize the display of null database values

- Set the `NullValue` property of a `DataGridViewCellStyle`. The following code example uses the `DataGridView.DefaultCellStyle` property to display "no entry" in all cells containing values equal to `DBNull.Value`.

```
this.dataGridView1.DefaultCellStyle.NullValue = "no entry";
```

```
Me.dataGridView1.DefaultCellStyle.NullValue = "no entry"
```

To enable wordwrap in text-based cells

- Set the `WrapMode` property of a `DataGridViewCellStyle` to one of the `DataGridViewTriState` enumeration values. The following code example uses the `DataGridView.DefaultCellStyle` property to set the wrap mode for the entire control.

```
this.dataGridView1.DefaultCellStyle.WrapMode =
DataGridViewTriState.True;
```

```
Me.dataGridView1.DefaultCellStyle.WrapMode = DataGridViewTriState.True
```

To specify the text alignment of DataGridView cells

- Set the `Alignment` property of a `DataGridViewCellStyle` to one of the `DataGridViewContentAlignment` enumeration values. The following code example sets the alignment for a specific column using the `DefaultCellStyle` property of the column.

```
this.dataGridView1.Columns["CustomerName"].DefaultCellStyle  
    .Alignment = DataGridViewContentAlignment.MiddleRight;
```

```
Me.dataGridView1.Columns("CustomerName").DefaultCellStyle _  
    .Alignment = DataGridViewContentAlignment.MiddleRight
```

Example

```
private void SetFormatting()  
{  
    this.dataGridView1.Columns["UnitPrice"].DefaultCellStyle.Format = "c";  
    this.dataGridView1.Columns["ShipDate"].DefaultCellStyle.Format = "d";  
    this.dataGridView1.Columns["CustomerName"].DefaultCellStyle  
        .Alignment = DataGridViewContentAlignment.MiddleRight;  
    this.dataGridView1.DefaultCellStyle.NullValue = "no entry";  
    this.dataGridView1.DefaultCellStyle.WrapMode =  
        DataGridViewTriState.True;  
}
```

```
Private Sub SetFormatting()  
    With Me.dataGridView1  
        .Columns("UnitPrice").DefaultCellStyle.Format = "c"  
        .Columns("ShipDate").DefaultCellStyle.Format = "d"  
        .Columns("CustomerName").DefaultCellStyle.Alignment = _  
            DataGridViewContentAlignment.MiddleRight  
        .DefaultCellStyle.NullValue = "no entry"  
        .DefaultCellStyle.WrapMode = DataGridViewTriState.True  
    End With  
End Sub
```

Compiling the Code

These examples require:

- A [DataGridView](#) control named `dataGridView1` that contains a column named `UnitPrice`, a column named `ShipDate`, and a column named `CustomerName`.
- References to the [System](#), [System.Drawing](#), and [System.Windows.Forms](#) assemblies.

Robust Programming

For maximum scalability, you should share [DataGridViewCellStyle](#) objects across multiple rows, columns, or cells that use the same styles rather than setting the style properties for each element separately. For more information, see [Best Practices for Scaling the Windows Forms DataGridView Control](#).

See Also

[DataGridView.DefaultCellStyle](#)

[DataGridViewBand.DefaultCellStyle](#)

[DataGridViewCellStyle](#)

[Basic Formatting and Styling in the Windows Forms DataGridView Control](#)

[Cell Styles in the Windows Forms DataGridView Control](#)

[Data Formatting in the Windows Forms DataGridView Control](#)

[How to: Customize Data Formatting in the Windows Forms DataGridView Control](#)

Formatting Types

How to: Set Font and Color Styles in the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

You can specify the visual appearance of cells within a [DataGridView](#) control by setting properties of the [DataGridViewCellStyle](#) class. You can retrieve instances of this class from various properties of the [DataGridView](#) class and its companion classes, or you can instantiate [DataGridViewCellStyle](#) objects for assignment to these properties.

The following procedures demonstrate basic customization of cell appearance using the [DefaultCellStyle](#) property. Every cell in the control inherits the styles specified through this property unless they are overridden at the column, row, or cell level. For an example of style inheritance, see [How to: Set Default Cell Styles for the Windows Forms DataGridView Control](#). For information about additional uses of the [DataGridViewCellStyle](#) class, see the topics listed in the See Also section.

There is extensive support for this task in Visual Studio. Also see [How to: Set Default Cell Styles and Data Formats for the Windows Forms DataGridView Control Using the Designer](#).

To specify the font used by DataGridView cells

- Set the [Font](#) property of a [DataGridViewCellStyle](#). The following code example uses the [DataGridView.DefaultCellStyle](#) property to set the font for the entire control.

```
this.dataGridView1.DefaultCellStyle.Font = new Font("Tahoma", 15);
```

```
Me.dataGridView1.DefaultCellStyle.Font = New Font("Tahoma", 15)
```

To specify the foreground and background colors of DataGridView cells

- Set the [ForeColor](#) and [BackColor](#) properties of a [DataGridViewCellStyle](#). The following code example uses the [DataGridView.DefaultCellStyle](#) property to set these styles for the entire control.

```
this.dataGridView1.DefaultCellStyle.ForeColor = Color.Blue;  
this.dataGridView1.DefaultCellStyle.BackColor = Color.Beige;
```

```
Me.dataGridView1.DefaultCellStyle.ForeColor = Color.Blue  
Me.dataGridView1.DefaultCellStyle.BackColor = Color.Beige
```

To specify the foreground and background colors of selected DataGridView cells

- Set the [SelectionForeColor](#) and [SelectionBackColor](#) properties of a [DataGridViewCellStyle](#). The following code example uses the [DataGridView.DefaultCellStyle](#) property to set these styles for the entire control.

```
this.dataGridView1.DefaultCellStyle.SelectionForeColor = Color.Yellow;  
this.dataGridView1.DefaultCellStyle.SelectionBackColor = Color.Black;
```

```
Me.dataGridView1.DefaultCellStyle.SelectionForeColor = Color.Yellow  
Me.dataGridView1.DefaultCellStyle.SelectionBackColor = Color.Black
```

Example

```
private void SetFontAndColors()
{
    this.dataGridView1.DefaultCellStyle.Font = new Font("Tahoma", 15);
    this.dataGridView1.DefaultCellStyle.ForeColor = Color.Blue;
    this.dataGridView1.DefaultCellStyle.BackColor = Color.Beige;
    this.dataGridView1.DefaultCellStyle.SelectionForeColor = Color.Yellow;
    this.dataGridView1.DefaultCellStyle.SelectionBackColor = Color.Black;
}
```

```
Private Sub SetFontAndColors()

    With Me.dataGridView1.DefaultCellStyle
        .Font = New Font("Tahoma", 15)
        .ForeColor = Color.Blue
        .BackColor = Color.Beige
        .SelectionForeColor = Color.Yellow
        .SelectionBackColor = Color.Black
    End With

End Sub
```

Compiling the Code

This example requires:

- A [DataGridView](#) control named `dataGridView1`.
- References to the [System](#), [System.Drawing](#), and [System.Windows.Forms](#) assemblies.

Robust Programming

For maximum scalability, you should share [DataGridViewCellStyle](#) objects across multiple rows, columns, or cells that use the same styles, rather than setting the style properties for each element separately. For more information, see [Best Practices for Scaling the Windows Forms DataGridView Control](#).

See Also

[DataGridView.DefaultCellStyle](#)

[DataGridViewCellStyle](#)

[Basic Formatting and Styling in the Windows Forms DataGridView Control](#)

[Cell Styles in the Windows Forms DataGridView Control](#)

How to: Set Alternating Row Styles for the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

Tabular data is often presented to users in a ledger-like format where alternating rows have different background colors. This format makes it easier for users to tell which cells are in each row, especially with wide tables that have many columns.

With the [DataGridView](#) control, you can specify complete style information for alternating rows. This enables you use style characteristics like foreground color and font, in addition to background color, to differentiate alternating rows.

There is support for this task in Visual Studio. Also see [How to: Set Alternating Row Styles for the Windows Forms DataGridView Control Using the Designer](#).

To set alternating row styles programmatically

- Set the properties of the [DataGridViewCellStyle](#) objects returned by the [RowsDefaultCellStyle](#) and [AlternatingRowsDefaultCellStyle](#) properties of the [DataGridView](#).

```
this.dataGridView1.RowsDefaultCellStyle.BackColor = Color.Bisque;
this.dataGridView1.AlternatingRowsDefaultCellStyle.BackColor =
    Color.Beige;
```

```
With Me.dataGridView1
    .RowsDefaultCellStyle.BackColor = Color.Bisque
    .AlternatingRowsDefaultCellStyle.BackColor = Color.Beige
End With
```

NOTE

The styles specified using the [RowsDefaultCellStyle](#) and [AlternatingRowsDefaultCellStyle](#) properties override the styles specified on the column and [DataGridView](#) level, but are overridden by the styles set on the individual row and cell level. For more information, see [Cell Styles in the Windows Forms DataGridView Control](#).

Compiling the Code

This example requires:

- A [DataGridView](#) control named `dataGridView1`.
- References to the [System](#), [System.Drawing](#), and [System.Windows.Forms](#) assemblies.

Robust Programming

For maximum scalability, you should share [DataGridViewCellStyle](#) objects across multiple rows, columns, or cells that use the same styles, rather than setting the style properties for each element separately. For more information, see [Best Practices for Scaling the Windows Forms DataGridView Control](#).

See Also

[DataGridView.AlternatingRowsDefaultCellStyle](#)

[DataGridView.RowsDefaultCellStyle](#)

[DataGridView](#)

[DataGridViewCellStyle](#)

[Basic Formatting and Styling in the Windows Forms DataGridView Control](#)

[Cell Styles in the Windows Forms DataGridView Control](#)

[Best Practices for Scaling the Windows Forms DataGridView Control](#)

[How to: Set Font and Color Styles in the Windows Forms DataGridView Control](#)

How to: Use the Row Template to Customize Rows in the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

The [DataGridView](#) control uses the row template as a basis for all rows that it adds to the control either through data binding or when you call the [DataGridViewRowCollection.Add](#) method without specifying an existing row to use.

The row template gives you greater control over the appearance and behavior of rows than the [RowsDefaultCellStyle](#) property provides. With the row template, you can set any [DataGridViewRow](#) properties, including [DefaultCellStyle](#).

There are some situations where you must use the row template to achieve a particular effect. For example, row height information cannot be stored in a [DataGridViewCellStyle](#), so you must use a row template to change the default height used by all rows. The row template is also useful when you create your own classes derived from [DataGridViewRow](#) and you want your custom type used when new rows are added to the control.

NOTE

The row template is used only when rows are added. You cannot change existing rows by changing the row template.

To use the row template

- Set properties on the object retrieved from the [DataGridView.RowTemplate](#) property.

```
DataGridViewRow^ row = this->dataGridView1->RowTemplate;
row->DefaultCellStyle->BackColor = Color::Bisque;
row->Height = 35;
row->MinimumHeight = 20;
```

```
DataGridViewRow row = this.dataGridView1.RowTemplate;
row.DefaultCellStyle.BackColor = Color.Bisque;
row.Height = 35;
row.MinimumHeight = 20;
```

```
With Me.dataGridView1.RowTemplate
    .DefaultCellStyle.BackColor = Color.Bisque
    .Height = 35
    .MinimumHeight = 20
End With
```

Compiling the Code

This example requires:

- A [DataGridView](#) control named `dataGridView1`.
- References to the [System](#), [System.Drawing](#), and [System.Windows.Forms](#) assemblies.

See Also

[DataGridView](#)

[DataGridViewCellStyle](#)

[DataGridViewRow](#)

[DataGridView.RowTemplate](#)

[Basic Formatting and Styling in the Windows Forms DataGridView Control](#)

[Cell Styles in the Windows Forms DataGridView Control](#)

Displaying Data in the Windows Forms DataGridView Control

5/4/2018 • 2 min to read • [Edit Online](#)

The `DataGridView` control is used to display data from a variety of external data sources. Alternatively, you can add rows and columns to the control and manually populate it with data.

When you bind the control to a data source, you can generate columns automatically based on the schema of the data source. If these columns do not appear just as you want them to, you can hide, remove, or rearrange them. You can also add unbound columns to display supplemental data that does not come from the data source.

Additionally, you can display your data using standard formats (such as currency format), or you can customize the display formatting to present your data however you need to (such as changing the background color for negative numbers, or replacing string values with corresponding images).

In This Section

[Data Display Modes in the Windows Forms DataGridView Control](#)

Describes the options for populating the control with data.

[Data Formatting in the Windows Forms DataGridView Control](#)

Describes the options for formatting cell display values.

[Walkthrough: Creating an Unbound Windows Forms DataGridView Control](#)

Describes how to manually populate the control with data.

[How to: Bind Data to the Windows Forms DataGridView Control](#)

Describes how to populate the control with data by binding it to a `BindingSource` that contains information pulled from a database.

[How to: Autogenerate Columns in a Data-Bound Windows Forms DataGridView Control](#)

Describes how to automatically generate columns based on a bound data source.

[How to: Remove Autogenerated Columns from a Windows Forms DataGridView Control](#)

Describes how to hide or delete columns generated automatically from a bound data source.

[How to: Change the Order of Columns in the Windows Forms DataGridView Control](#)

Describes how to rearrange columns generated automatically from a bound data source.

[How to: Add an Unbound Column to a Data-Bound Windows Forms DataGridView Control](#)

Describes how to supplement data from a bound data source by displaying additional, unbound columns.

[How to: Bind Objects to Windows Forms DataGridView Controls](#)

Describes how to bind the control to a collection of arbitrary objects so that each object is displayed in its own row.

[How to: Access Objects Bound to Windows Forms DataGridView Rows](#)

Describes how to retrieve an object bound to a particular row of the control.

[Walkthrough: Creating a Master/Detail Form Using Two Windows Forms DataGridView Controls](#)

Describes how to display data from two related database tables so that the values shown in one `DataGridView` control depend on the currently selected row in another control.

[How to: Customize Data Formatting in the Windows Forms DataGridView Control](#)

Describes how to handle the [DataGridView.CellFormatting](#) event to change the appearance of cells depending on their values.

Reference

[DataGridView](#)

Provides reference documentation for the [DataGridView](#) control.

[DataGridView.DataSource](#)

Provides reference documentation for the [DataSource](#) property.

[BindingSource](#)

Provides reference documentation for the [BindingSource](#) component.

Related Sections

[Data Entry in the Windows Forms DataGridView Control](#)

Provides topics that describe how to change the way users add and modify data in the control.

See Also

[DataGridView Control](#)

[Column Types in the Windows Forms DataGridView Control](#)

Data Display Modes in the Windows Forms DataGridView Control

5/4/2018 • 3 min to read • [Edit Online](#)

The [DataGridView](#) control can display data in three distinct modes: bound, unbound, and virtual. Choose the most suitable mode based on your requirements.

Unbound

Unbound mode is suitable for displaying relatively small amounts of data that you manage programmatically. You do not attach the [DataGridView](#) control directly to a data source as in bound mode. Instead, you must populate the control yourself, typically by using the [DataGridViewRowCollection.Add](#) method.

Unbound mode can be particularly useful for static, read-only data, or when you want to provide your own code that interacts with an external data store. When you want your users to interact with an external data source, however, you will typically use bound mode.

For an example that uses a read-only unbound [DataGridView](#), see [How to: Create an Unbound Windows Forms DataGridView Control](#).

Bound

Bound mode is suitable for managing data using automatic interaction with the data store. You can attach the [DataGridView](#) control directly to its data source by setting the [DataSource](#) property. When the control is data bound, data rows are pushed and pulled without the need of explicit management on your part. When the [AutoGenerateColumns](#) property is `true`, each column in your data source will cause a corresponding column to be created in the control. If you prefer to create your own columns, you can set this property to `false` and use the [DataPropertyName](#) property to bind each column when you configure it. This is useful when you want to use a column type other than the types that are generated by default. For more information, see [Column Types in the Windows Forms DataGridView Control](#).

For an example that uses a bound [DataGridView](#) control, see [Walkthrough: Validating Data in the Windows Forms DataGridView Control](#).

You can also add unbound columns to a [DataGridView](#) control in bound mode. This is useful when you want to display a column of buttons or links that enable users to perform actions on specific rows. It is also useful to display columns with values calculated from bound columns. You can populate the cell values for calculated columns in a handler for the [CellFormatting](#) event. If you are using a [DataSet](#) or [DataTable](#) as the data source, however, you might want to use the [DataColumn.Expression](#) property to create a calculated column instead. In this case, the [DataGridView](#) control will treat calculated column just like any other column in the data source.

Sorting by unbound columns in bound mode is not supported. If you create an unbound column in bound mode that contains user-editable values, you must implement virtual mode to maintain these values when the control is sorted by a bound column.

Virtual

With virtual mode, you can implement your own data management operations. This is necessary to maintain the values of unbound columns in bound mode when the control is sorted by bound columns. The primary use of virtual mode, however, is to optimize performance when interacting with large amounts of data.

You attach the [DataGridView](#) control to a cache that you manage, and your code controls when data rows are pushed and pulled. To keep the memory footprint small, the cache should be similar in size to the number of rows currently displayed. When the user scrolls new rows into view, your code requests new data from the cache and optionally flushes old data from memory.

When you are implementing virtual mode, you will need to track when a new row is needed in the data model and when to rollback the addition of the new row. The exact implementation of this functionality will depend on the implementation of the data model and the transaction semantics of the data model; whether commit scope is at the cell or row level.

For more information about virtual mode, see [Virtual Mode in the Windows Forms DataGridView Control](#). For an example that shows how to use virtual mode events, see [Walkthrough: Implementing Virtual Mode in the Windows Forms DataGridView Control](#).

See Also

[DataGridView](#)

[DataGridView.DataSource](#)

[DataGridView.VirtualMode](#)

[BindingSource](#)

[DataGridViewColumn.DataPropertyName](#)

[Displaying Data in the Windows Forms DataGridView Control](#)

[Column Types in the Windows Forms DataGridView Control](#)

[Walkthrough: Creating an Unbound Windows Forms DataGridView Control](#)

[How to: Bind Data to the Windows Forms DataGridView Control](#)

[Virtual Mode in the Windows Forms DataGridView Control](#)

[Walkthrough: Implementing Virtual Mode in the Windows Forms DataGridView Control](#)

Data Formatting in the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

The [DataGridView](#) control provides automatic conversion between cell values and the data types that the parent columns display. Text box columns, for example, display string representations of date, time, number, and enumeration values, and convert user-entered string values to the types required by the data store.

Formatting with the DataGridViewCellStyle class

The [DataGridView](#) control provides basic data formatting of cell values through the [DataGridViewCellStyle](#) class. You can use the [Format](#) property to format date, time, number, and enumeration values for the current default culture using the format specifiers described in [Formatting Types](#). You can also format these values for specific cultures using the [FormatProvider](#) property. The specified format is used both to display data and to parse data that the user enters in the specified format.

The [DataGridViewCellStyle](#) class provides additional formatting properties for wordwrap, text alignment, and the custom display of null database values. For more information, see [How to: Format Data in the Windows Forms DataGridView Control](#).

Formatting with the CellFormatting Event

If the basic formatting does not meet your needs, you can provide custom data formatting in a handler for the [DataGridView.CellFormatting](#) event. The [DataGridViewCellFormattingEventArgs](#) passed to the handler has a [Value](#) property that initially contains the cell value. Normally, this value is automatically converted to the display type. To convert the value yourself, set the [Value](#) property to a value of the display type.

NOTE

If a format string is in effect for the cell, it overrides your change of the [Value](#) property value unless you set the [FormattingApplied](#) property to `true`.

The [CellFormatting](#) event is also useful when you want to set [DataGridViewCellStyle](#) properties for individual cells based on their values. For more information, see [How to: Customize Data Formatting in the Windows Forms DataGridView Control](#).

If the default parsing of user-specified values does not meet your needs, you can handle the [CellParsing](#) event of the [DataGridView](#) control to provide custom parsing.

See Also

[DataGridView](#)

[DataGridViewCellStyle](#)

[Displaying Data in the Windows Forms DataGridView Control](#)

[Cell Styles in the Windows Forms DataGridView Control](#)

[How to: Format Data in the Windows Forms DataGridView Control](#)

[How to: Customize Data Formatting in the Windows Forms DataGridView Control](#)

Walkthrough: Creating an Unbound Windows Forms DataGridView Control

5/4/2018 • 7 min to read • [Edit Online](#)

You may frequently want to display tabular data that does not originate from a database. For example, you may want to show the contents of a two-dimensional array of strings. The [DataGridView](#) class provides an easy and highly customizable way to display data without binding to a data source. This walkthrough shows how to populate a [DataGridView](#) control and manage the addition and deletion of rows in "unbound" mode. By default, the user can add new rows. To prevent row addition, set the [AllowUserToAddRows](#) property is `false`.

To copy the code in this topic as a single listing, see [How to: Create an Unbound Windows Forms DataGridView Control](#).

Creating the Form

To use an unbound DataGridView control

1. Create a class that derives from [Form](#) and contains the following variable declarations and [Main](#) method.

```
using System;
using System.Drawing;
using System.Windows.Forms;

public class Form1 : System.Windows.Forms.Form
{
    private Panel buttonPanel = new Panel();
    private DataGridView songsDataGridView = new DataGridView();
    private Button addNewRowButton = new Button();
    private Button deleteRowButton = new Button();
```

```
Imports System
Imports System.Drawing
Imports System.Windows.Forms

Public Class Form1
    Inherits System.Windows.Forms.Form

    Private buttonPanel As New Panel
    Private WithEvents songsDataGridView As New DataGridView
    Private WithEvents addNewRowButton As New Button
    Private WithEvents deleteRowButton As New Button
```

```
[STAThreadAttribute()]
static void Main()
{
    Application.EnableVisualStyles();
    Application.Run(new Form1());
}
```

```

<STAThreadAttribute()> _
Public Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())
End Sub

End Class

```

2. Implement a `SetupLayout` method in your form's class definition to set up the form's layout.

```

private void SetupLayout()
{
    this.Size = new Size(600, 500);

    addNewRowButton.Text = "Add Row";
    addNewRowButton.Location = new Point(10, 10);
    addNewRowButton.Click += new EventHandler(addNewRowButton_Click);

    deleteRowButton.Text = "Delete Row";
    deleteRowButton.Location = new Point(100, 10);
    deleteRowButton.Click += new EventHandler(deleteRowButton_Click);

    buttonPanel.Controls.Add(addNewRowButton);
    buttonPanel.Controls.Add(deleteRowButton);
    buttonPanel.Height = 50;
    buttonPanel.Dock = DockStyle.Bottom;

    this.Controls.Add(this.buttonPanel);
}

```

```

Private Sub SetupLayout()

    Me.Size = New Size(600, 500)

    With addNewRowButton
        .Text = "Add Row"
        .Location = New Point(10, 10)
    End With

    With deleteRowButton
        .Text = "Delete Row"
        .Location = New Point(100, 10)
    End With

    With buttonPanel
        .Controls.Add(addNewRowButton)
        .Controls.Add(deleteRowButton)
        .Height = 50
        .Dock = DockStyle.Bottom
    End With

    Me.Controls.Add(Me.buttonPanel)

End Sub

```

3. Create a `SetupDataGridView` method to set up the `DataGridView` columns and properties.

This method first adds the `DataGridView` control to the form's `Controls` collection. Next, the number of columns to be displayed is set using the `ColumnCount` property. The default style for the column headers is set by setting the `BackColor`, `ForeColor`, and `Font` properties of the `DataGridViewCellStyle` returned by the `ColumnHeadersDefaultCellStyle` property.

Layout and appearance properties are set, and then the column names are assigned. When this method exits, the [DataGridView](#) control is ready to be populated.

```
private void SetupDataGridView()
{
    this.Controls.Add(songsDataGridView);

    songsDataGridView.ColumnCount = 5;

    songsDataGridView.ColumnHeadersDefaultCellStyle.BackColor = Color.Navy;
    songsDataGridView.ColumnHeadersDefaultCellStyle.ForeColor = Color.White;
    songsDataGridView.ColumnHeadersDefaultCellStyle.Font =
        new Font(songsDataGridView.Font, FontStyle.Bold);

    songsDataGridView.Name = "songsDataGridView";
    songsDataGridView.Location = new Point(8, 8);
    songsDataGridView.Size = new Size(500, 250);
    songsDataGridView.AutoSizeRowsMode =
        DataGridViewAutoSizeRowsMode.DisplayedCellsExceptHeaders;
    songsDataGridView.ColumnHeadersBorderStyle =
        DataGridViewHeaderBorderStyle.Single;
    songsDataGridView.CellBorderStyle = DataGridViewCellBorderStyle.Single;
    songsDataGridView.GridColor = Color.Black;
    songsDataGridView.RowHeadersVisible = false;

    songsDataGridView.Columns[0].Name = "Release Date";
    songsDataGridView.Columns[1].Name = "Track";
    songsDataGridView.Columns[2].Name = "Title";
    songsDataGridView.Columns[3].Name = "Artist";
    songsDataGridView.Columns[4].Name = "Album";
    songsDataGridView.Columns[4].DefaultCellStyle.Font =
        new Font(songsDataGridView.DefaultCellStyle.Font, FontStyle.Italic);

    songsDataGridView.SelectionMode =
        DataGridViewSelectionMode.FullRowSelect;
    songsDataGridView.MultiSelect = false;
    songsDataGridView.Dock = DockStyle.Fill;

    songsDataGridView.CellFormatting += new
        DataGridViewCellFormattingEventHandler(
            songsDataGridView_CellFormatting);
}
```

```

Private Sub SetupDataGridView()

    Me.Controls.Add(songsDataGridView)

    songsDataGridView.ColumnCount = 5
    With songsDataGridView.ColumnHeadersDefaultCellStyle
        .BackColor = Color.Navy
        .ForeColor = Color.White
        .Font = New Font(songsDataGridView.Font, FontStyle.Bold)
    End With

    With songsDataGridView
        .Name = "songsDataGridView"
        .Location = New Point(8, 8)
        .Size = New Size(500, 250)
        .AutoSizeRowsMode = _
            DataGridViewAutoSizeRowsMode.DisplayedCellsExceptHeaders
        .ColumnHeadersBorderStyle = DataGridViewHeaderBorderStyle.Single
        .CellBorderStyle = DataGridViewCellBorderStyle.Single
        .GridColor = Color.Black
        .RowHeadersVisible = False

        .Columns(0).Name = "Release Date"
        .Columns(1).Name = "Track"
        .Columns(2).Name = "Title"
        .Columns(3).Name = "Artist"
        .Columns(4).Name = "Album"
        .Columns(4).DefaultCellStyle.Font = _
            New Font(Me.songsDataGridView.DefaultCellStyle.Font, FontStyle.Italic)

        .SelectionMode = DataGridViewSelectionMode.FullRowSelect
        .MultiSelect = False
        .Dock = DockStyle.Fill
    End With

End Sub

```

4. Create a `PopulateDataGridView` method to add rows to the `DataGridView` control.

Each row represents a song and its associated information.

```
private void PopulateDataGridView()
{
    string[] row0 = { "11/22/1968", "29", "Revolution 9",
        "Beatles", "The Beatles [White Album]" };
    string[] row1 = { "1960", "6", "Fools Rush In",
        "Frank Sinatra", "Nice 'N' Easy" };
    string[] row2 = { "11/11/1971", "1", "One of These Days",
        "Pink Floyd", "Meddle" };
    string[] row3 = { "1988", "7", "Where Is My Mind?",
        "Pixies", "Surfer Rosa" };
    string[] row4 = { "5/1981", "9", "Can't Find My Mind",
        "Cramps", "Psychedelic Jungle" };
    string[] row5 = { "6/10/2003", "13",
        "Scatterbrain. (As Dead As Leaves.)",
        "Radiohead", "Hail to the Thief" };
    string[] row6 = { "6/30/1992", "3", "Dress", "P J Harvey", "Dry" };

    songsDataGridView.Rows.Add(row0);
    songsDataGridView.Rows.Add(row1);
    songsDataGridView.Rows.Add(row2);
    songsDataGridView.Rows.Add(row3);
    songsDataGridView.Rows.Add(row4);
    songsDataGridView.Rows.Add(row5);
    songsDataGridView.Rows.Add(row6);

    songsDataGridView.Columns[0].DisplayIndex = 3;
    songsDataGridView.Columns[1].DisplayIndex = 4;
    songsDataGridView.Columns[2].DisplayIndex = 0;
    songsDataGridView.Columns[3].DisplayIndex = 1;
    songsDataGridView.Columns[4].DisplayIndex = 2;
}
```

```

Private Sub PopulateDataGridView()

    Dim row0 As String() = {"11/22/1968", "29", "Revolution 9", _
        "Beatles", "The Beatles [White Album]"}
    Dim row1 As String() = {"1960", "6", "Fools Rush In", _
        "Frank Sinatra", "Nice 'N' Easy"}
    Dim row2 As String() = {"11/11/1971", "1", "One of These Days", _
        "Pink Floyd", "Meddle"}
    Dim row3 As String() = {"1988", "7", "Where Is My Mind?", _
        "Pixies", "Surfer Rosa"}
    Dim row4 As String() = {"5/1981", "9", "Can't Find My Mind", _
        "Cramps", "Psychedelic Jungle"}
    Dim row5 As String() = {"6/10/2003", "13", _
        "Scatterbrain. (As Dead As Leaves.)", _
        "Radiohead", "Hail to the Thief"}
    Dim row6 As String() = {"6/30/1992", "3", "Dress", "P J Harvey", "Dry"}

    With Me.songsDataGridView.Rows
        .Add(row0)
        .Add(row1)
        .Add(row2)
        .Add(row3)
        .Add(row4)
        .Add(row5)
        .Add(row6)
    End With

    With Me.songsDataGridView
        .Columns(0).DisplayIndex = 3
        .Columns(1).DisplayIndex = 4
        .Columns(2).DisplayIndex = 0
        .Columns(3).DisplayIndex = 1
        .Columns(4).DisplayIndex = 2
    End With

End Sub

```

5. With the utility methods in place, you can attach event handlers.

You will handle the **Add** and **Delete** buttons' [Click](#) events, the form's [Load](#) event, and the [DataGridView](#) control's [CellFormatting](#) event.

When the **Add** button's [Click](#) event is raised, a new, empty row is added to the [DataGridView](#).

When the **Delete** button's [Click](#) event is raised, the selected row is deleted, unless it is the row for new records, which enables the user add new rows. This row is always the last row in the [DataGridView](#) control.

When the form's [Load](#) event is raised, the `SetupLayout`, `SetupDataGridView`, and `PopulateDataGridView` utility methods are called.

When the [CellFormatting](#) event is raised, each cell in the `Date` column is formatted as a long date, unless the cell's value cannot be parsed.

```
public Form1()
{
    this.Load += new EventHandler(Form1_Load);
}

private void Form1_Load(System.Object sender, System.EventArgs e)
{
    SetupLayout();
    SetupDataGridView();
    PopulateDataGridView();
}

private void songsDataGridView_CellFormatting(object sender,
    System.Windows.Forms.DataGridViewCellFormattingEventArgs e)
{
    if (e != null)
    {
        if (this.songsDataGridView.Columns[e.ColumnIndex].Name == "Release Date")
        {
            if (e.Value != null)
            {
                try
                {
                    e.Value = DateTime.Parse(e.Value.ToString())
                        .ToString("yyyy-MM-dd");
                    e.FormattingApplied = true;
                }
                catch (FormatException)
                {
                    Console.WriteLine("{0} is not a valid date.", e.Value.ToString());
                }
            }
        }
    }
}

private void addNewRowButton_Click(object sender, EventArgs e)
{
    this.songsDataGridView.Rows.Add();
}

private void deleteRowButton_Click(object sender, EventArgs e)
{
    if (this.songsDataGridView.SelectedRows.Count > 0 &&
        this.songsDataGridView.SelectedRows[0].Index !=
        this.songsDataGridView.Rows.Count - 1)
    {
        this.songsDataGridView.Rows.RemoveAt(
            this.songsDataGridView.SelectedRows[0].Index);
    }
}
```

```

Private Sub Form1_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load

    SetupLayout()
    SetupDataGridView()
    PopulateDataGridView()

End Sub

Private Sub songsDataGridView_CellFormatting(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DataGridViewCellFormattingEventArgs) _
    Handles songsDataGridView.CellFormatting

    If e IsNot Nothing Then

        If Me.songsDataGridView.Columns(e.ColumnIndex).Name = _
            "Release Date" Then
            If e.Value IsNot Nothing Then
                Try
                    e.Value = DateTime.Parse(e.Value.ToString()) _
                        .ToString("yyyy-MM-dd")
                    e.FormattingApplied = True
                Catch ex As FormatException
                    Console.WriteLine("{0} is not a valid date.", e.Value.ToString())
                End Try
            End If
        End If

    End If

End Sub

Private Sub addNewRowButton_Click(ByVal sender As Object, _
    ByVal e As EventArgs) Handles addNewRowButton.Click

    Me.songsDataGridView.Rows.Add()

End Sub

Private Sub deleteRowButton_Click(ByVal sender As Object, _
    ByVal e As EventArgs) Handles deleteRowButton.Click

    If Me.songsDataGridView.SelectedRows.Count > 0 AndAlso _
        Not Me.songsDataGridView.SelectedRows(0).Index = _
        Me.songsDataGridView.Rows.Count - 1 Then

        Me.songsDataGridView.Rows.RemoveAt( _
            Me.songsDataGridView.SelectedRows(0).Index)

    End If

End Sub

```

Testing the Application

You can now test the form to make sure it behaves as expected.

To test the form

- Press F5 to run the application.

You will see a [DataGridView](#) control that displays the songs listed in [PopulateDataGridView](#). You can add new rows with the **Add Row** button, and you can delete selected rows with the **Delete Row** button. The unbound [DataGridView](#) control is the data store, and its data is independent of any external source, such as a [DataSet](#) or an array.

Next Steps

This application gives you a basic understanding of the [DataGridView](#) control's capabilities. You can customize the appearance and behavior of the [DataGridView](#) control in several ways:

- Change border and header styles. For more information, see [How to: Change the Border and Gridline Styles in the Windows Forms DataGridView Control](#).
- Enable or restrict user input to the [DataGridView](#) control. For more information, see [How to: Prevent Row Addition and Deletion in the Windows Forms DataGridView Control](#), and [How to: Make Columns Read-Only in the Windows Forms DataGridView Control](#).
- Check user input for database-related errors. For more information, see [Walkthrough: Handling Errors that Occur During Data Entry in the Windows Forms DataGridView Control](#).
- Handle very large data sets using virtual mode. For more information, see [Walkthrough: Implementing Virtual Mode in the Windows Forms DataGridView Control](#).
- Customize the appearance of cells. For more information, see [How to: Customize the Appearance of Cells in the Windows Forms DataGridView Control](#) and [How to: Set Default Cell Styles for the Windows Forms DataGridView Control](#).

See Also

[DataGridView](#)

[Displaying Data in the Windows Forms DataGridView Control](#)

[How to: Create an Unbound Windows Forms DataGridView Control](#)

[Data Display Modes in the Windows Forms DataGridView Control](#)

How to: Create an Unbound Windows Forms DataGridView Control

5/4/2018 • 5 min to read • [Edit Online](#)

The following code example demonstrates how to populate a [DataGridView](#) control programmatically without binding it to a data source. This is useful when you have a small amount of data that you want to display in a table format.

For a complete explanation of this code example, see [Walkthrough: Creating an Unbound Windows Forms DataGridView Control](#).

Example

```
using System;
using System.Drawing;
using System.Windows.Forms;

public class Form1 : System.Windows.Forms.Form
{
    private Panel buttonPanel = new Panel();
    private DataGridView songsDataGridView = new DataGridView();
    private Button addNewRowButton = new Button();
    private Button deleteRowButton = new Button();

    public Form1()
    {
        this.Load += new EventHandler(Form1_Load);
    }

    private void Form1_Load(System.Object sender, System.EventArgs e)
    {
        SetupLayout();
        SetupDataGridView();
        PopulateDataGridView();
    }

    private void songsDataGridView_CellFormatting(object sender,
        System.Windows.Forms.DataGridViewCellFormattingEventArgs e)
    {
        if (e != null)
        {
            if (this.songsDataGridView.Columns[e.ColumnIndex].Name == "Release Date")
            {
                if (e.Value != null)
                {
                    try
                    {
                        e.Value = DateTime.Parse(e.Value.ToString())
                            .ToString("yyyy-MM-dd");
                        e.FormattingApplied = true;
                    }
                    catch (FormatException)
                    {
                        Console.WriteLine("{0} is not a valid date.", e.Value.ToString());
                    }
                }
            }
        }
    }
}
```

```

private void addNewRowButton_Click(object sender, EventArgs e)
{
    this.songsDataGridView.Rows.Add();
}

private void deleteRowButton_Click(object sender, EventArgs e)
{
    if (this.songsDataGridView.SelectedRows.Count > 0 &&
        this.songsDataGridView.SelectedRows[0].Index !=
        this.songsDataGridView.Rows.Count - 1)
    {
        this.songsDataGridView.Rows.RemoveAt(
            this.songsDataGridView.SelectedRows[0].Index);
    }
}

private void SetupLayout()
{
    this.Size = new Size(600, 500);

    addNewRowButton.Text = "Add Row";
    addNewRowButton.Location = new Point(10, 10);
    addNewRowButton.Click += new EventHandler(addNewRowButton_Click);

    deleteRowButton.Text = "Delete Row";
    deleteRowButton.Location = new Point(100, 10);
    deleteRowButton.Click += new EventHandler(deleteRowButton_Click);

    buttonPanel.Controls.Add(addNewRowButton);
    buttonPanel.Controls.Add(deleteRowButton);
    buttonPanel.Height = 50;
    buttonPanel.Dock = DockStyle.Bottom;

    this.Controls.Add(this.buttonPanel);
}

private void SetupDataGridView()
{
    this.Controls.Add(songsDataGridView);

    songsDataGridView.ColumnCount = 5;

    songsDataGridView.ColumnHeadersDefaultCellStyle.BackColor = Color.Navy;
    songsDataGridView.ColumnHeadersDefaultCellStyle.ForeColor = Color.White;
    songsDataGridView.ColumnHeadersDefaultCellStyle.Font =
        new Font(songsDataGridView.Font, FontStyle.Bold);

    songsDataGridView.Name = "songsDataGridView";
    songsDataGridView.Location = new Point(8, 8);
    songsDataGridView.Size = new Size(500, 250);
    songsDataGridView.AutoSizeColumnsMode =
        DataGridViewAutoSizeColumnsMode.DisplayedCellsExceptHeaders;
    songsDataGridView.ColumnHeadersBorderStyle =
        DataGridViewHeaderBorderStyle.Single;
    songsDataGridView.CellBorderStyle = DataGridViewCellBorderStyle.Single;
    songsDataGridView.GridColor = Color.Black;
    songsDataGridView.RowHeadersVisible = false;

    songsDataGridView.Columns[0].Name = "Release Date";
    songsDataGridView.Columns[1].Name = "Track";
    songsDataGridView.Columns[2].Name = "Title";
    songsDataGridView.Columns[3].Name = "Artist";
    songsDataGridView.Columns[4].Name = "Album";
    songsDataGridView.Columns[4].DefaultCellStyle.Font =
        new Font(songsDataGridView.DefaultCellStyle.Font, FontStyle.Italic);

    songsDataGridView.SelectionMode =
        DataGridViewSelectionMode.FullRowSelect;
}

```

```

        songsDataGridView.MultiSelect = false;
        songsDataGridView.Dock = DockStyle.Fill;

        songsDataGridView.CellFormatting += new
            DataGridViewCellFormattingEventHandler(
            songsDataGridView_CellFormatting);
    }

    private void PopulateDataGridView()
    {

        string[] row0 = { "11/22/1968", "29", "Revolution 9",
            "Beatles", "The Beatles [White Album]" };
        string[] row1 = { "1960", "6", "Fools Rush In",
            "Frank Sinatra", "Nice 'N' Easy" };
        string[] row2 = { "11/11/1971", "1", "One of These Days",
            "Pink Floyd", "Meddle" };
        string[] row3 = { "1988", "7", "Where Is My Mind?",
            "Pixies", "Surfer Rosa" };
        string[] row4 = { "5/1981", "9", "Can't Find My Mind",
            "Cramps", "Psychedelic Jungle" };
        string[] row5 = { "6/10/2003", "13",
            "Scatterbrain. (As Dead As Leaves.)",
            "Radiohead", "Hail to the Thief" };
        string[] row6 = { "6/30/1992", "3", "Dress", "P J Harvey", "Dry" };

        songsDataGridView.Rows.Add(row0);
        songsDataGridView.Rows.Add(row1);
        songsDataGridView.Rows.Add(row2);
        songsDataGridView.Rows.Add(row3);
        songsDataGridView.Rows.Add(row4);
        songsDataGridView.Rows.Add(row5);
        songsDataGridView.Rows.Add(row6);

        songsDataGridView.Columns[0].DisplayIndex = 3;
        songsDataGridView.Columns[1].DisplayIndex = 4;
        songsDataGridView.Columns[2].DisplayIndex = 0;
        songsDataGridView.Columns[3].DisplayIndex = 1;
        songsDataGridView.Columns[4].DisplayIndex = 2;
    }

    [STAThreadAttribute()]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.Run(new Form1());
    }
}

```

```

Imports System
Imports System.Drawing
Imports System.Windows.Forms

Public Class Form1
    Inherits System.Windows.Forms.Form

    Private buttonPanel As New Panel
    Private WithEvents songsDataGridView As New DataGridView
    Private WithEvents addNewRowButton As New Button
    Private WithEvents deleteRowButton As New Button

    Private Sub Form1_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load

        SetupLayout()
        SetupDataGridView()

```

```

        PopulateDataGridView()

End Sub

Private Sub songsDataGridView_CellFormatting(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DataGridViewCellFormattingEventArgs) _
Handles songsDataGridView.CellFormatting

If e IsNot Nothing Then

    If Me.songsDataGridView.Columns(e.ColumnIndex).Name = _
    "Release Date" Then
        If e.Value IsNot Nothing Then
            Try
                e.Value = DateTime.Parse(e.Value.ToString()) _
                    .ToString("yyyy-MM-dd")
                e.FormattingApplied = True
            Catch ex As FormatException
                Console.WriteLine("{0} is not a valid date.", e.Value.ToString())
            End Try
        End If
    End If
End If

End If

End Sub

Private Sub addNewRowButton_Click(ByVal sender As Object, _
    ByVal e As EventArgs) Handles addNewRowButton.Click

    Me.songsDataGridView.Rows.Add()

End Sub

Private Sub deleteRowButton_Click(ByVal sender As Object, _
    ByVal e As EventArgs) Handles deleteRowButton.Click

    If Me.songsDataGridView.SelectedRows.Count > 0 AndAlso _
        Not Me.songsDataGridView.SelectedRows(0).Index = _
        Me.songsDataGridView.Rows.Count - 1 Then

        Me.songsDataGridView.Rows.RemoveAt( _
            Me.songsDataGridView.SelectedRows(0).Index)

    End If
End Sub

Private Sub SetupLayout()

    Me.Size = New Size(600, 500)

    With addNewRowButton
        .Text = "Add Row"
        .Location = New Point(10, 10)
    End With

    With deleteRowButton
        .Text = "Delete Row"
        .Location = New Point(100, 10)
    End With

    With buttonPanel
        .Controls.Add(addNewRowButton)
        .Controls.Add(deleteRowButton)
        .Height = 50
        .Dock = DockStyle.Bottom
    End With
End Sub

```

```

Me.Controls.Add(Me.buttonPanel)

End Sub

Private Sub SetupDataGridView()

    Me.Controls.Add(songsDataGridView)

    songsDataGridView.ColumnCount = 5
    With songsDataGridView.ColumnHeadersDefaultCellStyle
        .BackColor = Color.Navy
        .ForeColor = Color.White
        .Font = New Font(songsDataGridView.Font, FontStyle.Bold)
    End With

    With songsDataGridView
        .Name = "songsDataGridView"
        .Location = New Point(8, 8)
        .Size = New Size(500, 250)
        .AutoSizeRowsMode = _
            DataGridViewAutoSizeRowsMode.DisplayedCellsExceptHeaders
        .ColumnHeadersBorderStyle = DataGridViewHeaderBorderStyle.Single
        .CellBorderStyle = DataGridViewCellBorderStyle.Single
        .GridColor = Color.Black
        .RowHeadersVisible = False

        .Columns(0).Name = "Release Date"
        .Columns(1).Name = "Track"
        .Columns(2).Name = "Title"
        .Columns(3).Name = "Artist"
        .Columns(4).Name = "Album"
        .Columns(4).DefaultCellStyle.Font = _
            New Font(Me.songsDataGridView.DefaultCellStyle.Font, FontStyle.Italic)

        .SelectionMode = DataGridViewSelectionMode.FullRowSelect
        .MultiSelect = False
        .Dock = DockStyle.Fill
    End With

End Sub

Private Sub PopulateDataGridView()

    Dim row0 As String() = {"11/22/1968", "29", "Revolution 9", _
        "Beatles", "The Beatles [White Album]"}
    Dim row1 As String() = {"1960", "6", "Fools Rush In", _
        "Frank Sinatra", "Nice 'N' Easy"}
    Dim row2 As String() = {"11/11/1971", "1", "One of These Days", _
        "Pink Floyd", "Meddle"}
    Dim row3 As String() = {"1988", "7", "Where Is My Mind?", _
        "Pixies", "Surfer Rosa"}
    Dim row4 As String() = {"5/1981", "9", "Can't Find My Mind", _
        "Cramps", "Psychedelic Jungle"}
    Dim row5 As String() = {"6/10/2003", "13", _
        "Scatterbrain. (As Dead As Leaves.)", _
        "Radiohead", "Hail to the Thief"}
    Dim row6 As String() = {"6/30/1992", "3", "Dress", "P J Harvey", "Dry"}

    With Me.songsDataGridView.Rows
        .Add(row0)
        .Add(row1)
        .Add(row2)
        .Add(row3)
        .Add(row4)
        .Add(row5)
        .Add(row6)
    End With

    With Me.songsDataGridView

```

```

With Me.SongsDataGridView
    .Columns(0).DisplayIndex = 3
    .Columns(1).DisplayIndex = 4
    .Columns(2).DisplayIndex = 0
    .Columns(3).DisplayIndex = 1
    .Columns(4).DisplayIndex = 2
End With

End Sub

<STAThreadAttribute()> _
Public Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())
End Sub

End Class

```

Compiling the Code

This example requires:

- References to the System, System.Drawing, and System.Windows.Forms assemblies.

For information about building this example from the command line for visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[DataGridView](#)

[Walkthrough: Creating an Unbound Windows Forms DataGridView Control](#)

[Displaying Data in the Windows Forms DataGridView Control](#)

[Data Display Modes in the Windows Forms DataGridView Control](#)

How to: Bind Data to the Windows Forms DataGridView Control

5/4/2018 • 9 min to read • [Edit Online](#)

The [DataGridView](#) control supports the standard Windows Forms data binding model, so it will bind to a variety of data sources. In most circumstances, however, you will bind to a [BindingSource](#) component which will manage the details of interacting with the data source. The [BindingSource](#) component can represent any Windows Forms data source and gives you great flexibility when choosing or modifying the location of your data. For more information about the data sources supported by the [DataGridView](#) control, see [DataGridView Control Overview](#).

There is extensive support for this task in Visual Studio. Also see [How to: Bind Data to the Windows Forms DataGridView Control Using the Designer](#).

Procedure

To connect a DataGridView control to data

1. Implement a method to handle the details of retrieving data from a database. The following code example implements a `GetData` method that initializes a [SqlDataAdapter](#) component and uses it to populate a [DataTable](#). The [DataTable](#) is then bound to the [BindingSource](#) component. Be sure to set the `connectionString` variable to a value that is appropriate for your database. You will need access to a server with the Northwind SQL Server sample database installed.

```
private:
    void GetData(String^ selectCommand)
    {
        try
        {
            // Specify a connection string. Replace the given value with a
            // valid connection string for a Northwind SQL Server sample
            // database accessible to your system.
            String^ connectionString =
                "Integrated Security=SSPI;Persist Security Info=False;" +
                "Initial Catalog=Northwind;Data Source=localhost";

            // Create a new data adapter based on the specified query.
            dataAdapter = gcnew SqlDataAdapter(selectCommand, connectionString);

            // Create a command builder to generate SQL update, insert, and
            // delete commands based on selectCommand. These are used to
            // update the database.
            gcnew SqlCommandBuilder(dataAdapter);

            // Populate a new data table and bind it to the BindingSource.
            DataTable^ table = gcnew DataTable();
            dataAdapter->Fill(table);
            bindingSource1->DataSource = table;

            // Resize the DataGridView columns to fit the newly loaded content.
            dataGridView1->AutoSizeColumnsMode::AllCellsExceptHeader);
        }
        catch (SqlException^)
        {
            MessageBox::Show("To run this example, replace the value of the " +
                "connectionString variable with a connection string that is " +
                "valid for your system.");
        }
    }
}
```

```
private void GetData(string selectCommand)
{
    try
    {
        // Specify a connection string. Replace the given value with a
        // valid connection string for a Northwind SQL Server sample
        // database accessible to your system.
        String connectionString =
            "Integrated Security=SSPI;Persist Security Info=False;" +
            "Initial Catalog=Northwind;Data Source=localhost";

        // Create a new data adapter based on the specified query.
        dataAdapter = new SqlDataAdapter(selectCommand, connectionString);

        // Create a command builder to generate SQL update, insert, and
        // delete commands based on selectCommand. These are used to
        // update the database.
        SqlCommandBuilder commandBuilder = new SqlCommandBuilder(dataAdapter);

        // Populate a new data table and bind it to the BindingSource.
        DataTable table = new DataTable();
        table.Locale = System.Globalization.CultureInfo.InvariantCulture;
        dataAdapter.Fill(table);
        bindingSource1.DataSource = table;

        // Resize the DataGridView columns to fit the newly loaded content.
        dataGridView1.AutoResizeColumns(
            DataGridViewAutoSizeColumnsMode.AllCellsExceptHeader);
    }
    catch (SqlException)
    {
        MessageBox.Show("To run this example, replace the value of the " +
            "connectionString variable with a connection string that is " +
            "valid for your system.");
    }
}
```

```

Private Sub GetData(ByVal selectCommand As String)

    Try
        ' Specify a connection string. Replace the given value with a
        ' valid connection string for a Northwind SQL Server sample
        ' database accessible to your system.
        Dim connectionString As String = _
            "Integrated Security=SSPI;Persist Security Info=False;" + _
            "Initial Catalog=Northwind;Data Source=localhost"

        ' Create a new data adapter based on the specified query.
        Me.dataAdapter = New SqlDataAdapter(selectCommand, connectionString)

        ' Create a command builder to generate SQL update, insert, and
        ' delete commands based on selectCommand. These are used to
        ' update the database.
        Dim commandBuilder As New SqlCommandBuilder(Me.dataAdapter)

        ' Populate a new data table and bind it to the BindingSource.
        Dim table As New DataTable()
        table.Locale = System.Globalization.CultureInfo.InvariantCulture
        Me.dataAdapter.Fill(table)
        Me.bindingSource1.DataSource = table

        ' Resize the DataGridView columns to fit the newly loaded content.
        Me.dataGridView1.AutoResizeColumns( _
            DataGridViewAutoSizeColumnsMode.AllCellsExceptHeader)
    Catch ex As SqlException
        MessageBox.Show("To run this example, replace the value of the " + _
            "ConnectionString variable with a connection string that is " + _
            "valid for your system.")
    End Try

```

```
End Sub
```

2. In your form's **Load** event handler, bind the **DataGridView** control to the **BindingSource** component and call the **GetData** method to retrieve the data from the database.

```

void Form1_Load(Object^ /*sender*/, System::EventArgs^ /*e*/)
{
    // Bind the DataGridView to the BindingSource
    // and load the data from the database.
    dataGridView1->DataSource = bindingSource1;
    GetData("select * from Customers");
}

```

```

private void Form1_Load(object sender, System.EventArgs e)
{
    // Bind the DataGridView to the BindingSource
    // and load the data from the database.
    dataGridView1.DataSource = bindingSource1;
    GetData("select * from Customers");
}

```

```

Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles Me.Load

    ' Bind the DataGridView to the BindingSource
    ' and load the data from the database.
    Me.dataGridView1.DataSource = Me.bindingSource1
    GetData("select * from Customers")

End Sub

```

Example

The following complete code example provides buttons for reloading data from the database and submitting changes to the database.

```

#using <System.Data.dll>
#using <System.Transactions.dll>
#using <System.EnterpriseServices.dll>
#using <System.Xml.dll>
#using <System.Drawing.dll>
#using <System.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Data;
using namespace System::Data::SqlClient;
using namespace System::Windows::Forms;

public ref class Form1 : public System::Windows::Forms::Form
{
private:
    DataGridView^ dataGridView1;
private:
    BindingSource^ bindingSource1;
private:
    SqlDataAdapter^ dataAdapter;
private:
    Button^ reloadButton;
private:
    Button^ submitButton;

public:
    static void Main()
    {
        Application::Run(gcnew Form1());
    }

    // Initialize the form.
public:
    Form1()
    {
        dataGridView1 = gcnew DataGridView();
        bindingSource1 = gcnew BindingSource();
        dataAdapter = gcnew SqlDataAdapter();
        reloadButton = gcnew Button();
        submitButton = gcnew Button();

        dataGridView1->Dock = DockStyle::Fill;

        reloadButton->Text = "reload";
        submitButton->Text = "submit";
        reloadButton->Click += gcnew System::EventHandler(this,&Form1::reloadButton_Click);
        submitButton->Click += gcnew System::EventHandler(this,&Form1::submitButton_Click);

        FlowLayoutPanel^ panel = gcnew FlowLayoutPanel();

```

```

    panel->Dock = DockStyle::Top;
    panel->AutoSize = true;
    panel->Controls->AddRange(gcnew array<Control^> { reloadButton, submitButton });

    this->Controls->AddRange(gcnew array<Control^> { dataGridView1, panel });
    this->Load += gcnew System::EventHandler(this,&Form1::Form1_Load);
}

void Form1_Load(Object^ /*sender*/, System::EventArgs^ /*e*/)
{
    // Bind the DataGridView to the BindingSource
    // and load the data from the database.
    dataGridView1->DataSource = bindingSource1;
    GetData("select * from Customers");
}

void reloadButton_Click(Object^ /*sender*/, System::EventArgs^ /*e*/)
{
    // Reload the data from the database.
    GetData(dataAdapter->SelectCommand->CommandText);
}

void submitButton_Click(Object^ /*sender*/, System::EventArgs^ /*e*/)
{
    // Update the database with the user's changes.
    dataAdapter->Update((DataTable^)bindingSource1->DataSource);
}

private:
    void GetData(String^ selectCommand)
    {
        try
        {
            // Specify a connection string. Replace the given value with a
            // valid connection string for a Northwind SQL Server sample
            // database accessible to your system.
            String^ connectionString =
                "Integrated Security=SSPI;Persist Security Info=False;" +
                "Initial Catalog=Northwind;Data Source=localhost";

            // Create a new data adapter based on the specified query.
            dataAdapter = gcnew SqlDataAdapter(selectCommand, connectionString);

            // Create a command builder to generate SQL update, insert, and
            // delete commands based on selectCommand. These are used to
            // update the database.
            gcnew SqlCommandBuilder(dataAdapter);

            // Populate a new data table and bind it to the BindingSource.
            DataTable^ table = gcnew DataTable();
            dataAdapter->Fill(table);
            bindingSource1->DataSource = table;

            // Resize the DataGridView columns to fit the newly loaded content.
            dataGridView1->AutoSizeColumnsMode::AllCellsExceptHeader);
        }
        catch (SqlException^)
        {
            MessageBox::Show("To run this example, replace the value of the " +
                "connectionString variable with a connection string that is " +
                "valid for your system.");
        }
    }
};

using System;

```

```
using System.Data;
using System.Data.SqlClient;
using System.Windows.Forms;

public class Form1 : System.Windows.Forms.Form
{
    private DataGridView dataGridView1 = new DataGridView();
    private BindingSource bindingSource1 = new BindingSource();
    private SqlDataAdapter dataAdapter = new SqlDataAdapter();
    private Button reloadButton = new Button();
    private Button submitButton = new Button();

    [STAThreadAttribute()]
    public static void Main()
    {
        Application.Run(new Form1());
    }

    // Initialize the form.
    public Form1()
    {
        dataGridView1.Dock = DockStyle.Fill;

        reloadButton.Text = "reload";
        submitButton.Text = "submit";
        reloadButton.Click += new System.EventHandler(reloadButton_Click);
        submitButton.Click += new System.EventHandler(submitButton_Click);

        FlowLayoutPanel panel = new FlowLayoutPanel();
        panel.Dock = DockStyle.Top;
        panel.AutoSize = true;
        panel.Controls.AddRange(new Control[] { reloadButton, submitButton });

        this.Controls.AddRange(new Control[] { dataGridView1, panel });
        this.Load += new System.EventHandler(Form1_Load);
        this.Text = "DataGridView databinding and updating demo";
    }

    private void Form1_Load(object sender, System.EventArgs e)
    {
        // Bind the DataGridView to the BindingSource
        // and load the data from the database.
        dataGridView1.DataSource = bindingSource1;
        GetData("select * from Customers");
    }

    private void reloadButton_Click(object sender, System.EventArgs e)
    {
        // Reload the data from the database.
        GetData(dataAdapter.SelectCommand.CommandText);
    }

    private void submitButton_Click(object sender, System.EventArgs e)
    {
        // Update the database with the user's changes.
        dataAdapter.Update((DataTable)bindingSource1.DataSource);
    }

    private void GetData(string selectCommand)
    {
        try
        {
            // Specify a connection string. Replace the given value with a
            // valid connection string for a Northwind SQL Server sample
            // database accessible to your system.
            String connectionString =
                "Integrated Security=SSPI;Persist Security Info=False;" +
                "Initial Catalog=Northwind;Data Source=localhost";
        }
    }
}
```

```

// Create a new data adapter based on the specified query.
dataAdapter = new SqlDataAdapter(selectCommand, connectionString);

// Create a command builder to generate SQL update, insert, and
// delete commands based on selectCommand. These are used to
// update the database.
SqlCommandBuilder commandBuilder = new SqlCommandBuilder(dataAdapter);

// Populate a new data table and bind it to the BindingSource.
DataTable table = new DataTable();
table.Locale = System.Globalization.CultureInfo.InvariantCulture;
dataAdapter.Fill(table);
bindingSource1.DataSource = table;

// Resize the DataGridView columns to fit the newly loaded content.
dataGridView1.AutoResizeColumns(
    DataGridViewAutoSizeColumnsMode.AllCellsExceptHeader);
}

catch (SqlException)
{
    MessageBox.Show("To run this example, replace the value of the " +
        "connectionString variable with a connection string that is " +
        "valid for your system.");
}
}

}

}

```

```

Imports System
Imports System.Data
Imports System.Data.SqlClient
Imports System.Windows.Forms

Public Class Form1
    Inherits System.Windows.Forms.Form

    Private dataGridView1 As New DataGridView()
    Private bindingSource1 As New BindingSource()
    Private dataAdapter As New SqlDataAdapter()
    Private WithEvents reloadButton As New Button()
    Private WithEvents submitButton As New Button()

    <STAThreadAttribute()> _
    Public Shared Sub Main()
        Application.Run(New Form1())
    End Sub

    ' Initialize the form.
    Public Sub New()

        Me.dataGridView1.Dock = DockStyle.Fill

        Me.reloadButton.Text = "reload"
        Me.submitButton.Text = "submit"

        Dim panel As New FlowLayoutPanel()
        panel.Dock = DockStyle.Top
        panel.AutoSize = True
        panel.Controls.AddRange(New Control() {Me.reloadButton, Me.submitButton})

        Me.Controls.AddRange(New Control() {Me.dataGridView1, panel})
        Me.Text = "DataGridView databinding and updating demo"

    End Sub

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
        Handles Me.Load

```

```

' Bind the DataGridView to the BindingSource
' and load the data from the database.
Me.dataGridView1.DataSource = Me.bindingSource1
GetData("select * from Customers")

End Sub

Private Sub reloadButton_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles reloadButton.Click

    ' Reload the data from the database.
    GetData(Me.dataAdapter.SelectCommand.CommandText)

End Sub

Private Sub submitButton_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles submitButton.Click

    ' Update the database with the user's changes.
    Me.dataAdapter.Update(CType(Me.bindingSource1.DataSource, DataTable))

End Sub

Private Sub GetData(ByVal selectCommand As String)

    Try
        ' Specify a connection string. Replace the given value with a
        ' valid connection string for a Northwind SQL Server sample
        ' database accessible to your system.
        Dim connectionString As String = _
            "Integrated Security=SSPI;Persist Security Info=False;" + _
            "Initial Catalog=Northwind;Data Source=localhost"

        ' Create a new data adapter based on the specified query.
        Me.dataAdapter = New SqlDataAdapter(selectCommand, connectionString)

        ' Create a command builder to generate SQL update, insert, and
        ' delete commands based on selectCommand. These are used to
        ' update the database.
        Dim commandBuilder As New SqlCommandBuilder(Me.dataAdapter)

        ' Populate a new data table and bind it to the BindingSource.
        Dim table As New DataTable()
        table.Locale = System.Globalization.CultureInfo.InvariantCulture
        Me.dataAdapter.Fill(table)
        Me.bindingSource1.DataSource = table

        ' Resize the DataGridView columns to fit the newly loaded content.
        Me.dataGridView1.AutoSizeColumnsMode = _
            DataGridViewAutoSizeColumnsMode.AllCellsExceptHeader)
    Catch ex As SqlException
        MessageBox.Show("To run this example, replace the value of the " + _
            "connectionString variable with a connection string that is " + _
            "valid for your system.")
    End Try

End Sub

End Class

```

Compiling the Code

This example requires:

- References to the System, System.Windows.Forms, System.Data, and System.XML assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

.NET Framework Security

Storing sensitive information, such as a password, within the connection string can affect the security of your application. Using Windows Authentication (also known as integrated security) is a more secure way to control access to a database. For more information, see [Protecting Connection Information](#).

See Also

- [DataGridView](#)
- [DataGridView.DataSource](#)
- [BindingSource](#)
- [Displaying Data in the Windows Forms DataGridView Control](#)
- [Protecting Connection Information](#)

How to: Autogenerate Columns in a Data-Bound Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

The following code example demonstrates how to display columns from a bound data source in a `DataGridView` control. When the `AutoGenerateColumns` property value is `true` (the default), a `DataGridViewColumn` is created for each column in the data source table.

If the `DataGridView` control already has columns when you set the `DataSource` property, the existing bound columns are compared to the columns in the data source and preserved whenever there is a match. Unbound columns are always preserved. Bound columns for which there is no match in the data source are removed. Columns in the data source for which there is no match in the control generate new `DataGridViewColumn` objects, which are added to the end of the `Columns` collection.

Example

```
private void BindData()
{
    customersDataGridView.AutoGenerateColumns = true;
    customersDataGridView.DataSource = customersDataSet;
    customersDataGridView.DataMember = "Customers";
}
```

```
Private Sub BindData()

    With customersDataGridView
        .AutoGenerateColumns = True
        .DataSource = customersDataSet
        .DataMember = "Customers"
    End With

End Sub
```

Compiling the Code

This example requires:

- A `DataGridView` control named `customersDataGridView`.
- A `DataSet` object named `customersDataSet` that has a table named `Customers`.
- References to the `System`, `System.Windows.Forms`, `System.Data`, and `System.Xml` assemblies.

See Also

[DataGridView](#)

[DataGridView.AutoGenerateColumns](#)

[Displaying Data in the Windows Forms DataGridView Control](#)

[How to: Remove Autogenerated Columns from a Windows Forms DataGridView Control](#)

How to: Remove Autogenerated Columns from a Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

When your [DataGridView](#) control is set to autogenerate its columns based on data from its data source, you can selectively omit certain columns. You can do this by calling the [Remove](#) method on the [Columns](#) collection. Alternatively, you can hide columns from view by setting the [Visible](#) property to `false`. This technique is useful when you want to display the hidden columns in certain conditions, or when you need to access the data in the columns without displaying it.

To remove autogenerated columns

- Call the [Remove](#) method on the [Columns](#) collection.

```
dataGridView1.AutoGenerateColumns = true;
dataGridView1.DataSource = customersDataSet;
dataGridView1.Columns.Remove("Fax");
```

```
With dataGridView1
    .AutoGenerateColumns = True
    .DataSource = customersDataSet
    .Columns.Remove("Fax")
End With
```

To hide autogenerated columns

- Set the column's [Visible](#) property to `false`.

```
dataGridView1.Columns["CustomerID"].Visible = false;
```

```
dataGridView1.Columns("CustomerID").Visible = False
```

Example

```
private void BindDataAndInitializeColumns()
{
    dataGridView1.AutoGenerateColumns = true;
    dataGridView1.DataSource = customersDataSet;
    dataGridView1.Columns.Remove("Fax");
    dataGridView1.Columns["CustomerID"].Visible = false;
}
```

```
Private Sub BindDataAndInitializeColumns()

    With dataGridView1
        .AutoGenerateColumns = True
        .DataSource = customersDataSet
        .Columns.Remove("Fax")
        .Columns("CustomerID").Visible = False
    End With

End Sub
```

Compiling the Code

This example requires:

- A [DataGridView](#) control named `dataGridView1` bound to a table that contains `Fax` and `CustomerID` columns, such as the `Customers` table in the Northwind sample database.
- References to the [System](#) and [System.Windows.Forms](#) assemblies.

See Also

[DataGridView](#)
[DataGridView.AutoGenerateColumns](#)
[DataGridView.Columns](#)
[DataGridViewColumnCollection.Remove](#)
[DataGridViewColumn.Visible](#)
[Displaying Data in the Windows Forms DataGridView Control](#)

How to: Change the Order of Columns in the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

When you use a [DataGridView](#) to display data from a data source, the columns in the data source's schema sometimes do not appear in the order you would like to display them. You can change the displayed order of the columns by using the [DisplayIndex](#) property of the [DataGridViewColumn](#) class.

The following code example repositions some of the columns automatically generated when binding to the Customers table in the Northwind sample database. For more information about how to bind the [DataGridView](#) control to a database table, see [How to: Bind Data to the Windows Forms DataGridView Control](#).

There is support for this task in Visual Studio. Also see [How to: Change the Order of Columns in the Windows Forms DataGridView Control Using the Designer](#)

Example

```
private void AdjustColumnOrder()
{
    customersDataGridView.Columns["CustomerID"].Visible = false;
    customersDataGridView.Columns["ContactName"].DisplayIndex = 0;
    customersDataGridView.Columns["ContactTitle"].DisplayIndex = 1;
    customersDataGridView.Columns["City"].DisplayIndex = 2;
    customersDataGridView.Columns["Country"].DisplayIndex = 3;
    customersDataGridView.Columns["CompanyName"].DisplayIndex = 4;
}
```

```
Private Sub AdjustColumnOrder()

    With customersDataGridView
        .Columns("CustomerID").Visible = False
        .Columns("ContactName").DisplayIndex = 0
        .Columns("ContactTitle").DisplayIndex = 1
        .Columns("City").DisplayIndex = 2
        .Columns("Country").DisplayIndex = 3
        .Columns("CompanyName").DisplayIndex = 4
    End With

End Sub
```

Compiling the Code

This example requires:

- A [DataGridView](#) control named `customersDataGridView` that is bound to a table with the indicated column names, such as the `Customers` table in the Northwind sample database.
- References to the [System](#), [System.Windows.Forms](#), [System.Data](#), and [System.Xml](#) assemblies.

See Also

[DataGridView](#)

[DataGridViewColumn](#)

[DataGridViewColumn.DisplayIndex](#)

[DataGridViewColumn.Visible](#)

[Displaying Data in the Windows Forms DataGridView Control](#)

[How to: Bind Data to the Windows Forms DataGridView Control](#)

How to: Add an Unbound Column to a Data-Bound Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

The data you display in the [DataGridView](#) control will normally come from a data source of some kind, but you might want to display a column of data that does not come from the data source. This kind of column is called an unbound column. Unbound columns can take many forms. Frequently, they are used to provide access to the details of a data row.

The following code example demonstrates how to create an unbound column of **Details** buttons to display a child table related to a particular row in a parent table when you implement a master/detail scenario. To respond to button clicks, implement a [DataGridView.CellClick](#) event handler that displays a form containing the child table.

There is support for this task in Visual Studio. Also see [How to: Add and Remove Columns in the Windows Forms DataGridView Control Using the Designer](#)

Example

```
private void CreateUnboundButtonColumn()
{
    // Initialize the button column.
    DataGridViewButtonColumn buttonColumn =
        new DataGridViewButtonColumn();
    buttonColumn.Name = "Details";
    buttonColumn.HeaderText = "Details";
    buttonColumn.Text = "View Details";

    // Use the Text property for the button text for all cells rather
    // than using each cell's value as the text for its own button.
    buttonColumn.UseColumnTextForButtonValue = true;

    // Add the button column to the control.
    dataGridView1.Columns.Insert(0, buttonColumn);
}
```

```
Private Sub CreateUnboundButtonColumn()

    ' Initialize the button column.
    Dim buttonColumn As New DataGridViewButtonColumn

    With buttonColumn
        .HeaderText = "Details"
        .Name = "Details"
        .Text = "View Details"

        ' Use the Text property for the button text for all cells rather
        ' than using each cell's value as the text for its own button.
        .UseColumnTextForButtonValue = True
    End With

    ' Add the button column to the control.
    dataGridView1.Columns.Insert(0, buttonColumn)

End Sub
```

Compiling the Code

This example requires:

- A [DataGridView](#) control named `dataGridView1`.
- References to the [System](#) and [System.Windows.Forms](#) assemblies.

See Also

[DataGridView](#)

[Displaying Data in the Windows Forms DataGridView Control](#)

[Data Display Modes in the Windows Forms DataGridView Control](#)

How to: Bind Objects to Windows Forms DataGridView Controls

5/4/2018 • 3 min to read • [Edit Online](#)

The following code example demonstrates how to bind a collection of objects to a [DataGridView](#) control so that each object displays as a separate row. This example also illustrates how to display a property with an enumeration type in a [DataGridViewComboBoxColumn](#) so that the combo box drop-down list contains the enumeration values.

Example

```
using System;
using System.Windows.Forms;
public enum Title
{
    King,
    Sir
};
public class EnumsAndComboBox : Form
{
    private DataGridView dataGridView1 = new DataGridView();
    private BindingSource bindingSource1 = new BindingSource();

    public EnumsAndComboBox()
    {
        this.Load += new System.EventHandler(EnumsAndComboBox_Load);
    }

    private void EnumsAndComboBox_Load(object sender, System.EventArgs e)
    {
        // Populate the data source.
        bindingSource1.Add(new Knight(Title.King, "Uther", true));
        bindingSource1.Add(new Knight(Title.King, "Arthur", true));
        bindingSource1.Add(new Knight(Title.Sir, "Mordred", false));
        bindingSource1.Add(new Knight(Title.Sir, "Gawain", true));
        bindingSource1.Add(new Knight(Title.Sir, "Galahad", true));

        // Initialize the DataGridView.
        dataGridView1.AutoGenerateColumns = false;
        dataGridView1.AutoSize = true;
        dataGridView1.DataSource = bindingSource1;

        dataGridView1.Columns.Add(CreateComboBoxWithEnums());

        // Initialize and add a text box column.
        DataGridViewColumn column = new DataGridViewTextBoxColumn();
        column.DataPropertyName = "Name";
        column.Name = "Knight";
        dataGridView1.Columns.Add(column);

        // Initialize and add a check box column.
        column = new DataGridViewCheckBoxColumn();
        column.DataPropertyName = "GoodGuy";
        column.Name = "Good";
        dataGridView1.Columns.Add(column);

        // Initialize the form.
        this.Controls.Add(dataGridView1);
        this.AutoSize = true;
        this.Text = "DataGridView object binding demo";
    }
}
```

```

}

DataGridViewComboBoxColumn CreateComboBoxWithEnums()
{
    DataGridViewComboBoxColumn combo = new DataGridViewComboBoxColumn();
    combo.DataSource = Enum.GetValues(typeof(TITLE));
    combo.DataPropertyName = "Title";
    combo.Name = "Title";
    return combo;
}
#region "business object"
private class Knight
{
    private string hisName;
    private bool good;
    private TITLE hisTitle;

    public Knight(TITLE title, string name, bool good)
    {
        hisTitle = title;
        hisName = name;
        this.good = good;
    }

    public Knight()
    {
        hisTitle = TITLE.Sir;
        hisName = "<enter name>";
        good = true;
    }

    public string Name
    {
        get
        {
            return hisName;
        }

        set
        {
            hisName = value;
        }
    }

    public bool GoodGuy
    {
        get
        {
            return good;
        }
        set
        {
            good = value;
        }
    }

    public TITLE Title
    {
        get
        {
            return hisTitle;
        }
        set
        {
            hisTitle = value;
        }
    }
}
#endregion

```

```

[STAThread]
public static void Main()
{
    Application.Run(new EnumsAndComboBox());
}

}

```

```

Imports System.Windows.Forms
Imports System.Collections.Generic
Public Enum Title
    King
    Sir
End Enum
Public Class EnumsAndComboBox
    Inherits Form

    Private flow As New FlowLayoutPanel()
    Private WithEvents checkForChange As Button = New Button()
    Private knights As List(Of Knight)
    Private dataGridView1 As New DataGridView()

    Public Sub New()
        MyBase.New()
        SetupForm()
        SetupGrid()
    End Sub

    Private Sub SetupForm()
        AutoSize = True
    End Sub

    Private Sub SetupGrid()
        knights = New List(Of Knight)
        knights.Add(New Knight(Title.King, "Uther", True))
        knights.Add(New Knight(Title.King, "Arthur", True))
        knights.Add(New Knight(Title.Sir, "Mordred", False))
        knights.Add(New Knight(Title.Sir, "Gawain", True))
        knights.Add(New Knight(Title.Sir, "Galahad", True))

        ' Initialize the DataGridView.
        dataGridView1.AutoGenerateColumns = False
        dataGridView1.AutoSize = True
        dataGridView1.DataSource = knights

        dataGridView1.Columns.Add(CreateComboBoxWithEnums())

        ' Initialize and add a text box column.
        Dim column As DataGridViewTextBoxColumn = _
            New DataGridViewTextBoxColumn()
        column.DataPropertyName = "Name"
        column.Name = "Knight"
        dataGridView1.Columns.Add(column)

        ' Initialize and add a check box column.
        column = New DataGridViewCheckBoxColumn()
        column.DataPropertyName = "GoodGuy"
        column.Name = "Good"
        dataGridView1.Columns.Add(column)

        ' Initialize the form.
        Controls.Add(dataGridView1)
        Me.AutoSize = True
        Me.Text = "DataGridView object binding demo"
    End Sub

```

```

Private Function CreateComboBoxWithEnums() As DataGridViewComboBoxColumn
    Dim combo As New DataGridViewComboBoxColumn()
    combo.DataSource = [Enum].GetValues(GetType(TITLE))
    combo.DataPropertyName = "Title"
    combo.Name = "Title"
    Return combo
End Function

#Region "business object"
Private Class Knight
    Private hisName As String
    Private good As Boolean
    Private hisTitle As TITLE

    Public Sub New(ByVal title As TITLE, ByVal name As String, _
                  ByVal good As Boolean)

        hisTitle = title
        hisName = name
        Me.good = good
    End Sub

    Public Property Name() As String
        Get
            Return hisName
        End Get

        Set(ByVal Value As String)
            hisName = Value
        End Set
    End Property

    Public Property GoodGuy() As Boolean
        Get
            Return good
        End Get
        Set(ByVal Value As Boolean)
            good = Value
        End Set
    End Property

    Public Property Title() As TITLE
        Get
            Return hisTitle
        End Get
        Set(ByVal Value As TITLE)
            hisTitle = Value
        End Set
    End Property
End Class
#End Region

Public Shared Sub Main()
    Application.Run(New EnumsAndComboBox())
End Sub

End Class

```

Compiling the Code

This example requires:

- References to the System and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio

by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[DataGridView](#)

[Displaying Data in the Windows Forms DataGridView Control](#)

[How to: Access Objects Bound to Windows Forms DataGridView Rows](#)

How to: Access Objects Bound to Windows Forms DataGridView Rows

5/4/2018 • 3 min to read • [Edit Online](#)

Sometimes it is useful to display a table of information stored in a collection of business objects. When you bind a [DataGridView](#) control to such a collection, each public property is displayed in its own column unless the property has been marked non-browsable with a [BrowsableAttribute](#). For example, a collection of `Customer` objects would have columns such as **Name** and **Address**.

If these objects contain additional information and code that you want to access, you can reach it through row objects. In the following code example, users can select multiple rows and click a button to send an invoice to each of the corresponding customers.

To access row-bound objects

- Use the [DataGridViewRow.DataBoundItem](#) property.

```
void invoiceButton_Click(object sender, EventArgs e)
{
    foreach (DataGridViewRow row in this.dataGridView1.SelectedRows)
    {
        Customer cust = row.DataBoundItem as Customer;
        if (cust != null)
        {
            cust.SendInvoice();
        }
    }
}
```

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

    For Each row As DataGridViewRow In Me.DataGridView1.SelectedRows

        Dim cust As Customer = TryCast(row.DataBoundItem, Customer)
        If cust IsNot Nothing Then
            cust.SendInvoice()
        End If

        Next

    End Sub
}
```

Example

The complete code example includes a simple `Customer` implementation and binds the [DataGridView](#) to an `ArrayList` containing a few `Customer` objects. The [Click](#) event handler of the `System.Windows.Forms.Button` must access the `Customer` objects through the rows, because the customer collection is not accessible outside the [Form.Load](#) event handler.

```
using System;
using System.Windows.Forms;

public class DataGridViewObjectBinding : Form
```

```

{
    // These declarations and the Main() and New() methods
    // below can be replaced with designer-generated code.
    private Button invoiceButton = new Button();
    private DataGridView dataGridView1 = new DataGridView();

    // Entry point code.
    [STAThreadAttribute()]
    public static void Main()
    {
        Application.Run(new DataGridViewObjectBinding());
    }

    // Sets up the form.
    public DataGridViewObjectBinding()
    {
        this.dataGridView1.Dock = DockStyle.Fill;
        this.Controls.Add(this.dataGridView1);

        this.invoiceButton.Text = "invoice the selected customers";
        this.invoiceButton.Dock = DockStyle.Top;
        this.invoiceButton.Click += new EventHandler(invoiceButton_Click);
        this.Controls.Add(this.invoiceButton);

        this.Load += new EventHandler(DataGridViewObjectBinding_Load);
        this.Text = "DataGridView collection-binding demo";
    }

    void DataGridViewObjectBinding_Load(object sender, EventArgs e)
    {
        // Set up a collection of objects for binding.
        System.Collections.ArrayList customers = new System.Collections.ArrayList();
        customers.Add(new Customer("Harry"));
        customers.Add(new Customer("Sally"));
        customers.Add(new Customer("Roy"));
        customers.Add(new Customer("Pris"));

        // Initialize and bind the DataGridView.
        this.dataGridView1.SelectionMode =
            DataGridViewSelectionMode.FullRowSelect;
        this.dataGridView1.AutoGenerateColumns = true;
        this.dataGridView1.DataSource = customers;
    }

    // Calls the SendInvoice() method for the Customer
    // object bound to each selected row.
    void invoiceButton_Click(object sender, EventArgs e)
    {
        foreach (DataGridViewRow row in this.dataGridView1.SelectedRows)
        {
            Customer cust = row.DataBoundItem as Customer;
            if (cust != null)
            {
                cust.SendInvoice();
            }
        }
    }
}

public class Customer
{
    private String nameValue;

    public Customer(String name)
    {
        nameValue = name;
    }

    public String Name
}

```

```

    public string Name
    {
        get
        {
            return nameValue;
        }
        set
        {
            nameValue = value;
        }
    }

    public void SendInvoice()
    {
        MessageBox.Show(nameValue + " has been billed.");
    }
}

```

```

Imports System
Imports System.Windows.Forms

Public Class DataGridViewObjectBinding
    Inherits Form

    ' These declarations and the Main() and New() methods
    ' below can be replaced with designer-generated code.
    Private WithEvents InvoiceButton As New Button()
    Private WithEvents DataGridView1 As New DataGridView()

    ' Entry point code.
    <STAThreadAttribute()> _
    Public Shared Sub Main()

        Application.Run(New DataGridViewObjectBinding())

    End Sub

    ' Sets up the form.
    Public Sub New()

        Me.DataGridView1.Dock = DockStyle.Fill
        Me.Controls.Add(Me.DataGridView1)

        Me.InvoiceButton.Text = "invoice the selected customers"
        Me.InvoiceButton.Dock = DockStyle.Top
        Me.Controls.Add(Me.InvoiceButton)
        Me.Text = "DataGridView collection-binding demo"

    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load

        ' Set up a collection of objects for binding.
        Dim customers As New System.Collections.ArrayList()
        customers.Add(New Customer("Harry"))
        customers.Add(New Customer("Sally"))
        customers.Add(New Customer("Roy"))
        customers.Add(New Customer("Pris"))

        ' Initialize and bind the DataGridView.
        Me.DataGridView1.SelectionMode = _
            DataGridViewSelectionMode.FullRowSelect
        Me.DataGridView1.AutoGenerateColumns = True
        Me.DataGridView1.DataSource = customers

    End Sub

```

```

' Calls the SendInvoice() method for the Customer
' object bound to each selected row.
Private Sub InvoiceButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles InvoiceButton.Click

    For Each row As DataGridViewRow In Me.DataGridView1.SelectedRows

        Dim cust As Customer = TryCast(row.DataBoundItem, Customer)
        If cust IsNot Nothing Then
            cust.SendInvoice()
        End If

    Next

End Sub

End Class

Public Class Customer

    Private nameValue As String

    Public Sub New(ByVal name As String)
        nameValue = name
    End Sub

    Public Property Name() As String
        Get
            Return nameValue
        End Get
        Set(ByVal value As String)
            nameValue = value
        End Set
    End Property

    Public Sub SendInvoice()
        MsgBox(nameValue & " has been billed.")
    End Sub

End Class

```

Compiling the Code

This example requires:

- References to the System and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[DataGridView](#)
[DataGridViewRow](#)
[DataGridViewRow.DataBoundItem](#)
[Displaying Data in the Windows Forms DataGridView Control](#)
[How to: Bind Objects to Windows Forms DataGridView Controls](#)

How to: Access Objects in a Windows Forms DataGridViewComboBoxCell Drop-Down List

5/4/2018 • 8 min to read • [Edit Online](#)

Like the [ComboBox](#) control, the [DataGridViewComboBoxColumn](#) and [DataGridViewComboBoxCell](#) types enable you to add arbitrary objects to their drop-down lists. With this feature, you can represent complex states in a drop-down list without having to store corresponding objects in a separate collection.

Unlike the [ComboBox](#) control, the [DataGridView](#) types do not have a [SelectedItem](#) property for retrieving the currently selected object. Instead, you must set the [DataGridViewComboBoxColumn.ValueMember](#) or [DataGridViewComboBoxCell.ValueMember](#) property to the name of a property on your business object. When the user makes a selection, the indicated property of the business object sets the cell [Value](#) property.

To retrieve the business object through the cell value, the [ValueMember](#) property must indicate a property that returns a reference to the business object itself. Therefore, if the type of the business object is not under your control, you must add such a property by extending the type through inheritance.

The following procedures demonstrate how to populate a drop-down list with business objects and retrieve the objects through the cell [Value](#) property.

To add business objects to the drop-down list

1. Create a new [DataGridViewComboBoxColumn](#) and populate its [Items](#) collection. Alternatively, you can set the column [DataSource](#) property to the collection of business objects. In that case, however, you cannot add "unassigned" to the drop-down list without creating a corresponding business object in your collection.

```
DataGridViewComboBoxColumn assignedToColumn =
    new DataGridViewComboBoxColumn();

// Populate the combo box drop-down list with Employee objects.
foreach (Employee e in employees) assignedToColumn.Items.Add(e);

// Add "unassigned" to the drop-down list and display it for
// empty AssignedTo values or when the user presses CTRL+0.
assignedToColumn.Items.Add("unassigned");
assignedToColumn.DefaultCellStyle.NullValue = "unassigned";
```

```
Dim assignedToColumn As New DataGridViewComboBoxColumn()

' Populate the combo box drop-down list with Employee objects.
For Each e As Employee In employees
    assignedToColumn.Items.Add(e)
Next

' Add "unassigned" to the drop-down list and display it for
' empty AssignedTo values or when the user presses CTRL+0.
assignedToColumn.Items.Add("unassigned")
assignedToColumn.DefaultCellStyle.NullValue = "unassigned"
```

2. Set the [DisplayMember](#) and [ValueMember](#) properties. [DisplayMember](#) indicates the property of the business object to display in the drop-down list. [ValueMember](#) indicates the property that returns a reference to the business object.

```
assignedToColumn.DisplayMember = "Name";
assignedToColumn.ValueMember = "Self";
```

```
assignedToColumn.DisplayMember = "Name"
assignedToColumn.ValueMember = "Self"
```

3. Make sure that your business object type contains a property that returns a reference to the current instance. This property must be named with the value assigned to **ValueMember** in the previous step.

```
public Employee Self
{
    get { return this; }
}
```

```
Public ReadOnly Property Self() As Employee
    Get
        Return Me
    End Get
End Property
```

To retrieve the currently selected business object

- Get the cell **Value** property and cast it to the business object type.

```
// Retrieve the Employee object from the "Assigned To" cell.
Employee assignedTo = dataGridView1.Rows[e.RowIndex]
    .Cells["Assigned To"].Value as Employee;
```

```
' Retrieve the Employee object from the "Assigned To" cell.
Dim assignedTo As Employee = TryCast(dataGridView1.Rows(e.RowIndex) _
    .Cells("Assigned To").Value, Employee)
```

Example

The complete example demonstrates the use of business objects in a drop-down list. In the example, a **DataGridView** control is bound to a collection of **Task** objects. Each **Task** object has an **AssignedTo** property that indicates the **Employee** object currently assigned to that task. The **Assigned To** column displays the **Name** property value for each assigned employee, or "unassigned" if the **Task.AssignedTo** property value is **null**.

To view the behavior of this example, perform the following steps:

1. Change assignments in the **Assigned To** column by selecting different values from the drop-down lists or pressing CTRL+0 in a combo-box cell.
2. Click **Generate Report** to display the current assignments. This demonstrates that a change in the **Assigned To** column automatically updates the **tasks** collection.
3. Click a **Request Status** button to call the **RequestStatus** method of the current **Employee** object for that row. This demonstrates that the selected object has been successfully retrieved.

```
using System;
using System.Text;
using System.Collections.Generic;
using System.Windows.Forms.
```

```
using System.Windows.Forms;

public class Form1 : Form
{
    private List<Employee> employees = new List<Employee>();
    private List<Task> tasks = new List<Task>();
    private Button reportButton = new Button();
    private DataGridView dataGridView1 = new DataGridView();

    [STAThread]
    public static void Main()
    {
        Application.Run(new Form1());
    }

    public Form1()
    {
        dataGridView1.Dock = DockStyle.Fill;
        dataGridView1.AutoSizeColumnsMode =
            DataGridViewAutoSizeColumnsMode.AllCells;
        reportButton.Text = "Generate Report";
        reportButton.Dock = DockStyle.Top;
        reportButton.Click += new EventHandler(reportButton_Click);

        Controls.Add(dataGridView1);
        Controls.Add(reportButton);
        Load += new EventHandler(Form1_Load);
        Text = "DataGridViewComboBoxColumn Demo";
    }

    // Initializes the data source and populates the DataGridView control.
    private void Form1_Load(object sender, EventArgs e)
    {
        PopulateLists();
        dataGridView1.AutoGenerateColumns = false;
        dataGridView1.DataSource = tasks;
        AddColumns();
    }

    // Populates the employees and tasks lists.
    private void PopulateLists()
    {
        employees.Add(new Employee("Harry"));
        employees.Add(new Employee("Sally"));
        employees.Add(new Employee("Roy"));
        employees.Add(new Employee("Pris"));
        tasks.Add(new Task(1, employees[1]));
        tasks.Add(new Task(2));
        tasks.Add(new Task(3, employees[2]));
        tasks.Add(new Task(4));
    }

    // Configures columns for the DataGridView control.
    private void AddColumns()
    {
        DataGridViewTextBoxColumn idColumn =
            new DataGridViewTextBoxColumn();
        idColumn.Name = "Task";
        idColumn.DataPropertyName = "Id";
        idColumn.ReadOnly = true;

        DataGridViewComboBoxColumn assignedToColumn =
            new DataGridViewComboBoxColumn();

        // Populate the combo box drop-down list with Employee objects.
        foreach (Employee e in employees) assignedToColumn.Items.Add(e);

        // Add "unassigned" to the drop-down list and display it for
        // empty AssignedTo values or when the user presses CTRL+0.
        assignedToColumn.Items.Add("Unassigned");
    }
}
```

```

assignedToColumn.Items.Add("unassigned");
assignedToColumn.DefaultCellStyle.NullValue = "unassigned";

assignedToColumn.Name = "Assigned To";
assignedToColumn.DataPropertyName = "AssignedTo";
assignedToColumn.AutoComplete = true;
assignedToColumn.DisplayMember = "Name";
assignedToColumn.ValueMember = "Self";

// Add a button column.
DataGridViewButtonColumn buttonColumn =
    new DataGridViewButtonColumn();
buttonColumn.HeaderText = "";
buttonColumn.Name = "Status Request";
buttonColumn.Text = "Request Status";
buttonColumn.UseColumnTextForButtonValue = true;

dataGridView1.Columns.Add(idColumn);
dataGridView1.Columns.Add(assignedToColumn);
dataGridView1.Columns.Add(buttonColumn);

// Add a CellClick handler to handle clicks in the button column.
dataGridView1.CellClick +=
    new DataGridViewCellEventHandler(dataGridView1_CellClick);
}

// Reports on task assignments.
private void reportButton_Click(object sender, EventArgs e)
{
    StringBuilder report = new StringBuilder();
    foreach (Task t in tasks)
    {
        String assignment =
            t.AssignedTo == null ?
            "unassigned" : "assigned to " + t.AssignedTo.Name;
        report.AppendFormat("Task {0} is {1}.", t.Id, assignment);
        report.Append(Environment.NewLine);
    }
    MessageBox.Show(report.ToString(), "Task Assignments");
}

// Calls the Employee.RequestStatus method.
void dataGridView1_CellClick(object sender, DataGridViewCellEventArgs e)
{
    // Ignore clicks that are not on button cells.
    if (e.RowIndex < 0 || e.ColumnIndex !=
        dataGridView1.Columns["Status Request"].Index) return;

    // Retrieve the task ID.
    Int32 taskID = (Int32)dataGridView1[0, e.RowIndex].Value;

    // Retrieve the Employee object from the "Assigned To" cell.
    Employee assignedTo = dataGridView1.Rows[e.RowIndex]
        .Cells["Assigned To"].Value as Employee;

    // Request status through the Employee object if present.
    if (assignedTo != null)
    {
        assignedTo.RequestStatus(taskID);
    }
    else
    {
        MessageBox.Show(String.Format(
            "Task {0} is unassigned.", taskID), "Status Request");
    }
}
}

```

```

public class Task
{
    public Task(Int32 id)
    {
        idValue = id;
    }

    public Task(Int32 id, Employee assignedTo)
    {
        idValue = id;
        assignedToValue = assignedTo;
    }

    private Int32 idValue;
    public Int32 Id
    {
        get { return idValue; }
        set { idValue = value; }
    }

    private Employee assignedToValue;
    public Employee AssignedTo
    {
        get { return assignedToValue; }
        set { assignedToValue = value; }
    }
}

public class Employee
{
    public Employee(String name)
    {
        nameValue = name;
    }

    private String nameValue;
    public String Name
    {
        get { return nameValue; }
        set { nameValue = value; }
    }

    public Employee Self
    {
        get { return this; }
    }

    public void RequestStatus(Int32 taskID)
    {
        MessageBox.Show(String.Format(
            "Status for task {0} has been requested from {1}.",
            taskID, nameValue), "Status Request");
    }
}

```

```

Imports System
Imports System.Text
Imports System.Collections.Generic
Imports System.Windows.Forms

Public Class Form1
    Inherits Form

    Private employees As New List(Of Employee)
    Private tasks As New List(Of Task)
    Private WithEvents reportButton As New Button
    Private WithEvents dataGridView1 As New DataGridView

```

```

<STAThread()> _
Public Sub Main()
    Application.Run(New Form1)
End Sub

Sub New()
    dataGridView1.Dock = DockStyle.Fill
    dataGridView1.AutoSizeColumnsMode = _
        DataGridViewAutoSizeColumnsMode.AllCells
    reportButton.Text = "Generate Report"
    reportButton.Dock = DockStyle.Top

    Controls.Add(dataGridView1)
    Controls.Add(reportButton)
    Text = "DataGridViewComboBoxColumn Demo"
End Sub

' Initializes the data source and populates the DataGridView control.
Private Sub Form1_Load(ByVal sender As Object, _
    ByVal e As EventArgs) Handles Me.Load

    PopulateLists()
    dataGridView1.AutoGenerateColumns = False
    dataGridView1.DataSource = tasks
    AddColumns()

End Sub

' Populates the employees and tasks lists.
Private Sub PopulateLists()
    employees.Add(New Employee("Harry"))
    employees.Add(New Employee("Sally"))
    employees.Add(New Employee("Roy"))
    employees.Add(New Employee("Pris"))
    tasks.Add(New Task(1, employees(1)))
    tasks.Add(New Task(2))
    tasks.Add(New Task(3, employees(2)))
    tasks.Add(New Task(4))
End Sub

' Configures columns for the DataGridView control.
Private Sub AddColumns()

    Dim idColumn As New DataGridViewTextBoxColumn()
    idColumn.Name = "Task"
    idColumn.DataPropertyName = "Id"
    idColumn.ReadOnly = True

    Dim assignedToColumn As New DataGridViewComboBoxColumn()

    ' Populate the combo box drop-down list with Employee objects.
    For Each e As Employee In employees
        assignedToColumn.Items.Add(e)
    Next

    ' Add "unassigned" to the drop-down list and display it for
    ' empty AssignedTo values or when the user presses CTRL+0.
    assignedToColumn.Items.Add("unassigned")
    assignedToColumn.DefaultCellStyle.NullValue = "unassigned"

    assignedToColumn.Name = "Assigned To"
    assignedToColumn.DataPropertyName = "AssignedTo"
    assignedToColumn.AutoComplete = True
    assignedToColumn.DisplayMember = "Name"
    assignedToColumn.ValueMember = "Self"

    ' Add a button column.
    Dim buttonColumn As New DataGridViewButtonColumn()

```

```

buttonColumn.HeaderText = ""
buttonColumn.Name = "Status Request"
buttonColumn.Text = "Request Status"
buttonColumn.UseColumnTextForButtonValue = True

dataGridView1.Columns.Add(idColumn)
dataGridView1.Columns.Add(assignedToColumn)
dataGridView1.Columns.Add(buttonColumn)

End Sub

' Reports on task assignments.
Private Sub reportButton_Click(ByVal sender As Object, _
    ByVal e As EventArgs) Handles reportButton.Click

    Dim report As New StringBuilder()
    For Each t As Task In tasks
        Dim assignment As String
        If t.AssignedTo Is Nothing Then
            assignment = "unassigned"
        Else
            assignment = "assigned to " + t.AssignedTo.Name
        End If
        report.AppendFormat("Task {0} is {1}.", t.Id, assignment)
        report.Append(Environment.NewLine)
    Next
    MessageBox.Show(report.ToString(), "Task Assignments")

End Sub

' Calls the Employee.RequestStatus method.
Private Sub dataGridView1_CellClick(ByVal sender As Object, _
    ByVal e As DataGridViewCellEventArgs) _
    Handles dataGridView1.CellClick

    ' Ignore clicks that are not on button cells.
    If e.RowIndex < 0 OrElse Not e.ColumnIndex = _
        dataGridView1.Columns("Status Request").Index Then Return

    ' Retrieve the task ID.
    Dim taskID As Int32 = CInt(dataGridView1(0, e.RowIndex).Value)

    ' Retrieve the Employee object from the "Assigned To" cell.
    Dim assignedTo As Employee = TryCast(dataGridView1.Rows(e.RowIndex) _
        .Cells("Assigned To").Value, Employee)

    ' Request status through the Employee object if present.
    If assignedTo IsNot Nothing Then
        assignedTo.RequestStatus(taskID)
    Else
        MessageBox.Show(String.Format( _
            "Task {0} is unassigned.", taskID), "Status Request")
    End If

End Sub

End Class

Public Class Task

Sub New(ByVal id As Int32)
    idValue = id
End Sub

Sub New(ByVal id As Int32, ByVal assignedTo As Employee)
    idValue = id
    assignedToValue = assignedTo
End Sub

```

```

Private idValue As Int32
Public Property Id() As Int32
    Get
        Return idValue
    End Get
    Set(ByVal value As Int32)
        idValue = value
    End Set
End Property

Private assignedToValue As Employee
Public Property AssignedTo() As Employee
    Get
        Return assignedToValue
    End Get
    Set(ByVal value As Employee)
        assignedToValue = value
    End Set
End Property

End Class

Public Class Employee

Sub New(ByVal name As String)
    nameValue = name
End Sub

Private nameValue As String
Public Property Name() As String
    Get
        Return nameValue
    End Get
    Set(ByVal value As String)
        nameValue = value
    End Set
End Property

Public ReadOnly Property Self() As Employee
    Get
        Return Me
    End Get
End Property

Public Sub RequestStatus(ByVal taskID As Int32)
    MessageBox.Show(String.Format(
        "Status for task {0} has been requested from {1}.",
        taskID, nameValue), "Status Request")
End Sub

End Class

```

Compiling the Code

This example requires:

- References to the System and System.Windows.Forms assemblies.

See Also

[DataGridView](#)

[DataGridViewComboBoxColumn](#)

[DataGridViewComboBoxColumn.Items](#)

[DataGridViewComboBoxColumn.DataSource](#)

[DataGridViewComboBoxColumn.ValueMember](#)

[DataGridViewComboBoxCell](#)

[DataGridViewComboBoxCell.Items](#)

[DataGridViewComboBoxCell.DataSource](#)

[DataGridViewComboBoxCell.ValueMember](#)

[DataGridViewCell.Value](#)

[ComboBox](#)

[Displaying Data in the Windows Forms DataGridView Control](#)

Walkthrough: Creating a Master/Detail Form Using Two Windows Forms DataGridView Controls

5/4/2018 • 6 min to read • [Edit Online](#)

One of the most common scenarios for using the [DataGridView](#) control is the *master/detail* form, in which a parent/child relationship between two database tables is displayed. Selecting rows in the master table causes the detail table to update with the corresponding child data.

Implementing a master/detail form is easy using the interaction between the [DataGridView](#) control and the [BindingSource](#) component. In this walkthrough, you will build the form using two [DataGridView](#) controls and two [BindingSource](#) components. The form will show two related tables in the Northwind SQL Server sample database: `Customers` and `Orders`. When you are finished, you will have a form that shows all the customers in the database in the master [DataGridView](#) and all the orders for the selected customer in the detail [DataGridView](#).

To copy the code in this topic as a single listing, see [How to: Create a Master/Detail Form Using Two Windows Forms DataGridView Controls](#).

Prerequisites

In order to complete this walkthrough, you will need:

- Access to a server that has the Northwind SQL Server sample database.

Creating the form

To create a master/detail form

1. Create a class that derives from [Form](#) and contains two [DataGridView](#) controls and two [BindingSource](#) components. The following code provides basic form initialization and includes a `Main` method. If you use the Visual Studio designer to create your form, you can use the designer generated code instead of this code, but be sure to use the names shown in the variable declarations here.

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Windows.Forms;

public class Form1 : System.Windows.Forms.Form
{
    private DataGridView masterDataGridView = new DataGridView();
    private BindingSource masterBindingSource = new BindingSource();
    private DataGridView detailsDataGridView = new DataGridView();
    private BindingSource detailsBindingSource = new BindingSource();

    [STAThreadAttribute()]
    public static void Main()
    {
        Application.Run(new Form1());
    }

    // Initializes the form.
    public Form1()
    {
        masterDataGridView.Dock = DockStyle.Fill;
        detailsDataGridView.Dock = DockStyle.Fill;

        SplitContainer splitContainer1 = new SplitContainer();
        splitContainer1.Dock = DockStyle.Fill;
        splitContainer1.Orientation = Orientation.Horizontal;
        splitContainer1.Panel1.Controls.Add(masterDataGridView);
        splitContainer1.Panel2.Controls.Add(detailsDataGridView);

        this.Controls.Add(splitContainer1);
        this.Load += new System.EventHandler(Form1_Load);
        this.Text = "DataGridView master/detail demo";
    }
}
```

```

Imports System
Imports System.Data
Imports System.Data.SqlClient
Imports System.Windows.Forms

Public Class Form1
    Inherits System.Windows.Forms.Form

    Private masterDataGridView As New DataGridView()
    Private masterBindingSource As New BindingSource()
    Private detailsDataGridView As New DataGridView()
    Private detailsBindingSource As New BindingSource()

    <STAThreadAttribute()> _
    Public Shared Sub Main()
        Application.Run(New Form1())
    End Sub

    ' Initializes the form.
    Public Sub New()

        masterDataGridView.Dock = DockStyle.Fill
        detailsDataGridView.Dock = DockStyle.Fill

        Dim splitContainer1 As New SplitContainer()
        splitContainer1.Dock = DockStyle.Fill
        splitContainer1.Orientation = Orientation.Horizontal
        splitContainer1.Panel1.Controls.Add(masterDataGridView)
        splitContainer1.Panel2.Controls.Add(detailsDataGridView)

        Me.Controls.Add(splitContainer1)
        Me.Text = "DataGridView master/detail demo"

    End Sub

```

```
}
```

```
End Class
```

2. Implement a method in your form's class definition for handling the detail of connecting to the database. This example uses a `GetData` method that populates a `DataSet` object, adds a `DataRelation` object to the data set, and binds the `BindingSource` components. Be sure to set the `connectionString` variable to a value that is appropriate for your database.

IMPORTANT

Storing sensitive information, such as a password, within the connection string can affect the security of your application. Using Windows Authentication (also known as integrated security) is a more secure way to control access to a database. For more information, see [Protecting Connection Information](#).

```
private void GetData()
{
    try
    {
        // Specify a connection string. Replace the given value with a
        // valid connection string for a Northwind SQL Server sample
        // database accessible to your system.
        String connectionString =
            "Integrated Security=SSPI;Persist Security Info=False;" +
            "Initial Catalog=Northwind;Data Source=localhost";
        SqlConnection connection = new SqlConnection(connectionString);

        // Create a DataSet.
        DataSet data = new DataSet();
        data.Locale = System.Globalization.CultureInfo.InvariantCulture;

        // Add data from the Customers table to the DataSet.
        SqlDataAdapter masterDataAdapter = new
            SqlDataAdapter("select * from Customers", connection);
        masterDataAdapter.Fill(data, "Customers");

        // Add data from the Orders table to the DataSet.
        SqlDataAdapter detailsDataAdapter = new
            SqlDataAdapter("select * from Orders", connection);
        detailsDataAdapter.Fill(data, "Orders");

        // Establish a relationship between the two tables.
        DataRelation relation = new DataRelation("CustomersOrders",
            data.Tables["Customers"].Columns["CustomerID"],
            data.Tables["Orders"].Columns["CustomerID"]);
        data.Relations.Add(relation);

        // Bind the master data connector to the Customers table.
        masterBindingSource.DataSource = data;
        masterBindingSource.DataMember = "Customers";

        // Bind the details data connector to the master data connector,
        // using the DataRelation name to filter the information in the
        // details table based on the current row in the master table.
        detailsBindingSource.DataSource = masterBindingSource;
        detailsBindingSource.DataMember = "CustomersOrders";
    }
    catch (SqlException)
    {
        MessageBox.Show("To run this example, replace the value of the " +
            "connectionString variable with a connection string that is " +
            "valid for your system.");
    }
}
```

```

Private Sub GetData()

    Try
        ' Specify a connection string. Replace the given value with a
        ' valid connection string for a Northwind SQL Server sample
        ' database accessible to your system.
        Dim connectionString As String = _
            "Integrated Security=SSPI;Persist Security Info=False;" & _
            "Initial Catalog=Northwind;Data Source=localhost"
        Dim connection As New SqlConnection(connectionString)

        ' Create a DataSet.
        Dim data As New DataSet()
        data.Locale = System.Globalization.CultureInfo.InvariantCulture

        ' Add data from the Customers table to the DataSet.
        Dim masterDataAdapter As _
            New SqlDataAdapter("select * from Customers", connection)
        masterDataAdapter.Fill(data, "Customers")

        ' Add data from the Orders table to the DataSet.
        Dim detailsDataAdapter As _
            New SqlDataAdapter("select * from Orders", connection)
        detailsDataAdapter.Fill(data, "Orders")

        ' Establish a relationship between the two tables.
        Dim relation As New DataRelation("CustomersOrders", _
            data.Tables("Customers").Columns("CustomerID"), _
            data.Tables("Orders").Columns("CustomerID"))
        data.Relations.Add(relation)

        ' Bind the master data connector to the Customers table.
        masterBindingSource.DataSource = data
        masterBindingSource.DataMember = "Customers"

        ' Bind the details data connector to the master data connector,
        ' using the DataRelation name to filter the information in the
        ' details table based on the current row in the master table.
        detailsBindingSource.DataSource = masterBindingSource
        detailsBindingSource.DataMember = "CustomersOrders"
    Catch ex As SqlException
        MessageBox.Show("To run this example, replace the value of the " & _
            "connectionString variable with a connection string that is " & _
            "valid for your system.")
    End Try

End Sub

```

3. Implement a handler for your form's [Load](#) event that binds the [DataGridView](#) controls to the [BindingSource](#) components and calls the [GetData](#) method. The following example includes code that resizes [DataGridView](#) columns to fit the displayed data.

```

private void Form1_Load(object sender, System.EventArgs e)
{
    // Bind the DataGridView controls to the BindingSource
    // components and load the data from the database.
    masterDataGridView.DataSource = masterBindingSource;
    detailsDataGridView.DataSource = detailsBindingSource;
    GetData();

    // Resize the master DataGridView columns to fit the newly loaded data.
    masterDataGridView.AutoResizeColumns();

    // Configure the details DataGridView so that its columns automatically
    // adjust their widths when the data changes.
    detailsDataGridView.AutoSizeColumnsMode =
        DataGridViewAutoSizeColumnsMode.AllCells;
}

```

```

Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles Me.Load

    ' Bind the DataGridView controls to the BindingSource
    ' components and load the data from the database.
    masterDataGridView.DataSource = masterBindingSource
    detailsDataGridView.DataSource = detailsBindingSource
    GetData()

    ' Resize the master DataGridView columns to fit the newly loaded data.
    masterDataGridView.AutoResizeColumns()

    ' Configure the details DataGridView so that its columns automatically
    ' adjust their widths when the data changes.
    detailsDataGridView.AutoSizeColumnsMode = _
        DataGridViewAutoSizeColumnsMode.AllCells

End Sub

```

Testing the Application

You can now test the form to make sure it behaves as expected.

To test the form

- Compile and run the application.

You will see two [DataGridView](#) controls, one above the other. On top are the customers from the Northwind `customers` table, and at the bottom are the `Orders` corresponding to the selected customer. As you select different rows in the upper [DataGridView](#), the contents of the lower [DataGridView](#) change accordingly.

Next Steps

This application gives you a basic understanding of the [DataGridView](#) control's capabilities. You can customize the appearance and behavior of the [DataGridView](#) control in several ways:

- Change border and header styles. For more information, see [How to: Change the Border and Gridline Styles in the Windows Forms DataGridView Control](#).
- Enable or restrict user input to the [DataGridView](#) control. For more information, see [How to: Prevent Row Addition and Deletion in the Windows Forms DataGridView Control](#), and [How to: Make Columns Read-Only in the Windows Forms DataGridView Control](#).

- Validate user input to the [DataGridView](#) control. For more information, see [Walkthrough: Validating Data in the Windows Forms DataGridView Control](#).
- Handle very large data sets using virtual mode. For more information, see [Walkthrough: Implementing Virtual Mode in the Windows Forms DataGridView Control](#).
- Customize the appearance of cells. For more information, see [How to: Customize the Appearance of Cells in the Windows Forms DataGridView Control](#) and [How to: Set Default Cell Styles for the Windows Forms DataGridView Control](#).

See Also

[DataGridView](#)

[BindingSource](#)

[Displaying Data in the Windows Forms DataGridView Control](#)

[How to: Create a Master/Detail Form Using Two Windows Forms DataGridView Controls](#)

[Protecting Connection Information](#)

How to: Create a Master/Detail Form Using Two Windows Forms DataGridView Controls

5/4/2018 • 5 min to read • [Edit Online](#)

The following code example creates a master/detail form using two `DataGridView` controls bound to two `BindingSource` components. The data source is a `DataSet` that contains the `customers` and `orders` tables from the Northwind SQL Server sample database along with a `DataRelation` that relates the two through the `CustomerID` column.

One `BindingSource` is bound to the parent `customers` table in the data set. This data is displayed in the master `DataGridView` control. The other `BindingSource` is bound to the first data connector. The `DataMember` property of the second `BindingSource` is set to the `DataRelation` name. This causes the associated detail `DataGridView` control to display the rows of the child `orders` table that correspond to the current row in the master `DataGridView` control.

For a complete explanation of this code example, see [Walkthrough: Creating a Master/Detail Form Using Two Windows Forms DataGridView Controls](#).

Example

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Windows.Forms;

public class Form1 : System.Windows.Forms.Form
{
    private DataGridView masterDataGridView = new DataGridView();
    private BindingSource masterBindingSource = new BindingSource();
    private DataGridView detailsDataGridView = new DataGridView();
    private BindingSource detailsBindingSource = new BindingSource();

    [STAThreadAttribute()]
    public static void Main()
    {
        Application.Run(new Form1());
    }

    // Initializes the form.
    public Form1()
    {
        masterDataGridView.Dock = DockStyle.Fill;
        detailsDataGridView.Dock = DockStyle.Fill;

        SplitContainer splitContainer1 = new SplitContainer();
        splitContainer1.Dock = DockStyle.Fill;
        splitContainer1.Orientation = Orientation.Horizontal;
        splitContainer1.Panel1.Controls.Add(masterDataGridView);
        splitContainer1.Panel2.Controls.Add(detailsDataGridView);

        this.Controls.Add(splitContainer1);
        this.Load += new System.EventHandler(Form1_Load);
        this.Text = "DataGridView master/detail demo";
    }

    private void Form1_Load(object sender, System.EventArgs e)
    {
```

```

// Bind the DataGridView controls to the BindingSource
// components and load the data from the database.
masterDataGridView.DataSource = masterBindingSource;
detailsDataGridView.DataSource = detailsBindingSource;
GetData();

// Resize the master DataGridView columns to fit the newly loaded data.
masterDataGridView.AutoResizeColumns();

// Configure the details DataGridView so that its columns automatically
// adjust their widths when the data changes.
detailsDataGridView.AutoSizeColumnsMode =
    DataGridViewAutoSizeColumnsMode.AllCells;
}

private void GetData()
{
    try
    {
        // Specify a connection string. Replace the given value with a
        // valid connection string for a Northwind SQL Server sample
        // database accessible to your system.
        String connectionString =
            "Integrated Security=SSPI;Persist Security Info=False;" +
            "Initial Catalog=Northwind;Data Source=localhost";
        SqlConnection connection = new SqlConnection(connectionString);

        // Create a DataSet.
        DataSet data = new DataSet();
        data.Locale = System.Globalization.CultureInfo.InvariantCulture;

        // Add data from the Customers table to the DataSet.
        SqlDataAdapter masterDataAdapter = new
            SqlDataAdapter("select * from Customers", connection);
        masterDataAdapter.Fill(data, "Customers");

        // Add data from the Orders table to the DataSet.
        SqlDataAdapter detailsDataAdapter = new
            SqlDataAdapter("select * from Orders", connection);
        detailsDataAdapter.Fill(data, "Orders");

        // Establish a relationship between the two tables.
        DataRelation relation = new DataRelation("CustomersOrders",
            data.Tables["Customers"].Columns["CustomerID"],
            data.Tables["Orders"].Columns["CustomerID"]);
        data.Relations.Add(relation);

        // Bind the master data connector to the Customers table.
        masterBindingSource.DataSource = data;
        masterBindingSource.DataMember = "Customers";

        // Bind the details data connector to the master data connector,
        // using the DataRelation name to filter the information in the
        // details table based on the current row in the master table.
        detailsBindingSource.DataSource = masterBindingSource;
        detailsBindingSource.DataMember = "CustomersOrders";
    }
    catch (SqlException)
    {
        MessageBox.Show("To run this example, replace the value of the " +
            "connectionString variable with a connection string that is " +
            "valid for your system.");
    }
}
}

```

```

Imports System.Data
Imports System.Data.SqlClient
Imports System.Windows.Forms

Public Class Form1
    Inherits System.Windows.Forms.Form

    Private masterDataGridView As New DataGridView()
    Private masterBindingSource As New BindingSource()
    Private detailsDataGridView As New DataGridView()
    Private detailsBindingSource As New BindingSource()

    <STAThreadAttribute()> _
    Public Shared Sub Main()
        Application.Run(New Form1())
    End Sub

    ' Initializes the form.
    Public Sub New()

        masterDataGridView.Dock = DockStyle.Fill
        detailsDataGridView.Dock = DockStyle.Fill

        Dim splitContainer1 As New SplitContainer()
        splitContainer1.Dock = DockStyle.Fill
        splitContainer1.Orientation = Orientation.Horizontal
        splitContainer1.Panel1.Controls.Add(masterDataGridView)
        splitContainer1.Panel2.Controls.Add(detailsDataGridView)

        Me.Controls.Add(splitContainer1)
        Me.Text = "DataGridView master/detail demo"

    End Sub

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
        Handles Me.Load

        ' Bind the DataGridView controls to the BindingSource
        ' components and load the data from the database.
        masterDataGridView.DataSource = masterBindingSource
        detailsDataGridView.DataSource = detailsBindingSource
        GetData()

        ' Resize the master DataGridView columns to fit the newly loaded data.
        masterDataGridView.AutoResizeColumns()

        ' Configure the details DataGridView so that its columns automatically
        ' adjust their widths when the data changes.
        detailsDataGridView.AutoSizeColumnsMode = _
            DataGridViewAutoSizeColumnsMode.AllCells

    End Sub

    Private Sub GetData()

        Try
            ' Specify a connection string. Replace the given value with a
            ' valid connection string for a Northwind SQL Server sample
            ' database accessible to your system.
            Dim connectionString As String = _
                "Integrated Security=SSPI;Persist Security Info=False;" & _
                "Initial Catalog=Northwind;Data Source=localhost"
            Dim connection As New SqlConnection(connectionString)

            ' Create a DataSet.
            Dim data As New DataSet()
            data.Locale = System.Globalization.CultureInfo.InvariantCulture

            ' Add data from the Customers table to the DataSet.

```

```

Dim masterDataAdapter As _
    New SqlDataAdapter("select * from Customers", connection)
masterDataAdapter.Fill(data, "Customers")

' Add data from the Orders table to the DataSet.
Dim detailsDataAdapter As _
    New SqlDataAdapter("select * from Orders", connection)
detailsDataAdapter.Fill(data, "Orders")

' Establish a relationship between the two tables.
Dim relation As New DataRelation("CustomersOrders", _
    data.Tables("Customers").Columns("CustomerID"), _
    data.Tables("Orders").Columns("CustomerID"))
data.Relations.Add(relation)

' Bind the master data connector to the Customers table.
masterBindingSource.DataSource = data
masterBindingSource.DataMember = "Customers"

' Bind the details data connector to the master data connector,
' using the DataRelation name to filter the information in the
' details table based on the current row in the master table.
detailsBindingSource.DataSource = masterBindingSource
detailsBindingSource.DataMember = "CustomersOrders"
Catch ex As SqlException
    MessageBox.Show("To run this example, replace the value of the " & _
        "connectionString variable with a connection string that is " & _
        "valid for your system.")
End Try

End Sub

End Class

```

Compiling the Code

This example requires:

References to the System, System.Data, System.Windows.Forms, and System.XML assemblies.

- For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

.NET Framework Security

Storing sensitive information, such as a password, within the connection string can affect the security of your application. Using Windows Authentication (also known as integrated security) is a more secure way to control access to a database. For more information, see [Protecting Connection Information](#).

See Also

[DataGridView](#)

[BindingSource](#)

[Walkthrough: Creating a Master/Detail Form Using Two Windows Forms DataGridView Controls](#)

[Displaying Data in the Windows Forms DataGridView Control](#)

[Protecting Connection Information](#)

How to: Customize Data Formatting in the Windows Forms DataGridView Control

5/4/2018 • 3 min to read • [Edit Online](#)

The following code example demonstrates how to implement a handler for the [DataGridView.CellFormatting](#) event that changes how cells are displayed depending on their columns and values.

Cells in the `Balance` column that contain negative numbers are given a red background. You can also format these cells as currency to display parentheses around negative values. For more information, see [How to: Format Data in the Windows Forms DataGridView Control](#).

Cells in the `Priority` column display images in place of corresponding textual cell values. The `Value` property of the [DataGridViewCellFormattingEventArgs](#) is used both to get the textual cell value and to set the corresponding image display value.

Example

```
using System;
using System.Drawing;
using System.Windows.Forms;

public class Form1 : Form
{
    private DataGridView dataGridView1 = new DataGridView();
    private Bitmap highPriImage;
    private Bitmap mediumPriImage;
    private Bitmap lowPriImage;

    public Form1()
    {
        // Initialize the images.
        try
        {
            highPriImage = new Bitmap("highPri.bmp");
            mediumPriImage = new Bitmap("mediumPri.bmp");
            lowPriImage = new Bitmap("lowPri.bmp");
        }
        catch (ArgumentException)
        {
            MessageBox.Show("The Priority column requires Bitmap images " +
                "named highPri.bmp, mediumPri.bmp, and lowPri.bmp " +
                "residing in the same directory as the executable file.");
        }

        // Initialize the DataGridView.
        dataGridView1.Dock = DockStyle.Fill;
        dataGridView1.AllowUserToAddRows = false;
        dataGridView1.Columns.AddRange(
            new DataGridViewTextBoxColumn(),
            new DataGridViewImageColumn());
        dataGridView1.Columns[0].Name = "Balance";
        dataGridView1.Columns[1].Name = "Priority";
        dataGridView1.Rows.Add("-100", "high");
        dataGridView1.Rows.Add("0", "medium");
        dataGridView1.Rows.Add("100", "low");
        dataGridView1.CellFormatting +=
            new System.Windows.Forms.DataGridViewCellFormattingEventHandler(
                this.dataGridView1_CellFormatting);
    }
}
```

```

        this.Controls.Add(dataGridView1);
    }

    // Changes how cells are displayed depending on their columns and values.
    private void dataGridView1_CellFormatting(object sender,
        System.Windows.Forms.DataGridViewCellFormattingEventArgs e)
    {
        // Set the background to red for negative values in the Balance column.
        if (dataGridView1.Columns[e.ColumnIndex].Name.Equals("Balance"))
        {
            Int32 intValue;
            if (Int32.TryParse((String)e.Value, out intValue) &&
                (intValue < 0))
            {
                e.CellStyle.BackColor = Color.Red;
                e.CellStyle.SelectionBackColor = Color.DarkRed;
            }
        }

        // Replace string values in the Priority column with images.
        if (dataGridView1.Columns[e.ColumnIndex].Name.Equals("Priority"))
        {
            // Ensure that the value is a string.
            String stringValue = e.Value as string;
            if (stringValue == null) return;

            // Set the cell ToolTip to the text value.
            DataGridViewCell cell = dataGridView1[e.ColumnIndex, e.RowIndex];
            cell.ToolTipText = stringValue;

            // Replace the string value with the image value.
            switch (stringValue)
            {
                case "high":
                    e.Value = highPriImage;
                    break;
                case "medium":
                    e.Value = mediumPriImage;
                    break;
                case "low":
                    e.Value = lowPriImage;
                    break;
            }
        }
    }

    public static void Main()
    {
        Application.Run(new Form1());
    }
}

```

```

Imports System
Imports System.Drawing
Imports System.Windows.Forms

Public Class Form1
    Inherits Form

    Private WithEvents dataGridView1 As New DataGridView()
    Private highPriImage As Bitmap
    Private mediumPriImage As Bitmap
    Private lowPriImage As Bitmap

    Public Sub New()

```

```

' Initialize the images.
Try
    highPriImage = New Bitmap("highPri.bmp")
    mediumPriImage = New Bitmap("mediumPri.bmp")
    lowPriImage = New Bitmap("lowPri.bmp")
Catch ex As ArgumentException
    MessageBox.Show("The Priority column requires Bitmap images" & _
        "named highPri.bmp, mediumPri.bmp, and lowPri.bmp " & _
        "residing in the same directory as the executable file.")
End Try

' Initialize the DataGridView.
With dataGridView1
    .Dock = DockStyle.Fill
    .AllowUserToAddRows = False
    .Columns.AddRange( _
        New DataGridViewTextBoxColumn(), _
        New DataGridViewImageColumn())
    .Columns(0).Name = "Balance"
    .Columns(1).Name = "Priority"
    .Rows.Add("-100", "high")
    .Rows.Add("0", "medium")
    .Rows.Add("100", "low")
End With

Me.Controls.Add(dataGridView1)

End Sub

' Changes how cells are displayed depending on their columns and values.
Private Sub dataGridView1_CellFormatting(ByVal sender As Object, _
 ByVal e As System.Windows.Forms.DataGridViewCellFormattingEventArgs) _
 Handles dataGridView1.CellFormatting

    ' Set the background to red for negative values in the Balance column.
    If dataGridView1.Columns(e.ColumnIndex).Name.Equals("Balance") Then

        'Dim intValue As Int32
        If CInt(e.Value) < 0 Then
            'if Int32.TryParse((String)e.Value, out intValue) &&
            '    (intValue < 0))

                e.CellStyle.BackColor = Color.Red
                e.CellStyle.SelectionBackColor = Color.DarkRed
        End If
    End If

    ' Replace string values in the Priority column with images.
    If dataGridView1.Columns(e.ColumnIndex).Name.Equals("Priority") Then

        ' Ensure that the value is a string.
        Dim stringValue As String = TryCast(e.Value, String)
        If stringValue Is Nothing Then Return

        ' Set the cell ToolTip to the text value.
        Dim cell As DataGridViewCell = _
            dataGridView1(e.ColumnIndex, e.RowIndex)
        cell.ToolTipText = stringValue

        ' Replace the string value with the image value.
        Select Case stringValue

            Case "high"
                e.Value = highPriImage
            Case "medium"
                e.Value = mediumPriImage
            Case "low"
                e.Value = lowPriImage
        End Select
    End If
End Sub

```

```
    End Select

    End If

End Sub

Public Sub Main()
    Application.Run(New Form1())
End Sub

End Class
```

Compiling the Code

This example requires:

- References to the System, System.Drawing, and System.Windows.Forms assemblies.
- **Bitmap** images named `highPri.bmp`, `mediumPri.bmp`, and `lowPri.bmp` residing in the same directory as the executable file.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[DataGridView.DefaultCellStyle](#)

[DataGridViewBand.DefaultCellStyle](#)

[DataGridView](#)

[DataGridViewCellStyle](#)

[Bitmap](#)

[Displaying Data in the Windows Forms DataGridView Control](#)

[How to: Format Data in the Windows Forms DataGridView Control](#)

[Cell Styles in the Windows Forms DataGridView Control](#)

[Data Formatting in the Windows Forms DataGridView Control](#)

Resizing Columns and Rows in the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

The `DataGridView` control provides numerous options for customizing the sizing behavior of its columns and rows. Typically, `DataGridView` cells do not resize based on their contents. Instead, they clip any display value that is larger than the cell. If the content can be displayed as a string, the cell displays it in a ToolTip.

By default, users can drag row, column, and header dividers with the mouse to show more information. Users can also double-click a divider to automatically resize the associated row, column, or header band based on its contents.

The `DataGridView` control provides properties, methods, and events that enable you to customize or disable all of these user-directed behaviors. Additionally, you can programmatically resize rows, columns, and headers to fit their contents, or you can configure them to automatically resize themselves whenever their contents change. You can also configure columns to automatically divide the available width of the control in proportions that you specify.

In This Section

[Sizing Options in the Windows Forms DataGridView Control](#)

Describes the options for sizing rows, columns, and headers. Also provides details on sizing-related properties and methods, and describes common usage scenarios.

[Column Fill Mode in the Windows Forms DataGridView Control](#)

Describes column fill mode in detail, and provides demonstration code that you can use to experiment with column fill mode and other modes.

[How to: Set the Sizing Modes of the Windows Forms DataGridView Control](#)

Describes how to configure the sizing modes for common purposes.

[How to: Programmatically Resize Cells to Fit Content in the Windows Forms DataGridView Control](#)

Provides demonstration code that you can use to experiment with programmatic resizing.

[How to: Automatically Resize Cells When Content Changes in the Windows Forms DataGridView Control](#)

Provides demonstration code that you can use to experiment with automatic sizing modes.

Reference

[DataGridView](#)

Provides reference documentation for the `DataGridView` control.

See Also

[DataGridView Control](#)

Sizing Options in the Windows Forms DataGridView Control

5/4/2018 • 8 min to read • [Edit Online](#)

[DataGridView](#) rows, columns, and headers can change size as a result of many different occurrences. The following table shows these occurrences.

OCCURRENCE	DESCRIPTION
User resize	Users can make size adjustments by dragging or double-clicking row, column, or header dividers.
Control resize	In column fill mode, column widths change when the control width changes; for example, when the control is docked to its parent form and the user resizes the form.
Cell value change	In content-based automatic sizing modes, sizes change to fit new display values.
Method call	Programmatic content-based resizing lets you make opportunistic size adjustments based on cell values at the time of the method call.
Property setting	You can also set specific height and width values.

By default, user resizing is enabled, automatic sizing is disabled, and cell values that are wider than their columns are clipped.

The following table shows scenarios that you can use to adjust the default behavior or to use specific sizing options to achieve particular effects.

SCENARIO	IMPLEMENTATION
Use column fill mode for displaying similarly sized data in a relatively small number of columns that occupy the entire width of the control without displaying the horizontal scroll bar.	Set the AutoSizeColumnsMode property to Fill .
Use column fill mode with display values of varying sizes.	Set the AutoSizeColumnsMode property to Fill . Initialize relative column widths by setting the column FillWeight properties or by calling the control AutoResizeColumns method after filling the control with data.
Use column fill mode with values of varying importance.	Set the AutoSizeColumnsMode property to Fill . Set large MinimumWidth values for columns that must always display some of their data or use a sizing option other than fill mode for specific columns.
Use column fill mode to avoid displaying the control background.	Set the AutoSizeMode property of the last column to Fill and use other sizing options for the other columns. If the other columns use too much of the available space, set the MinimumWidth property of the last column.

SCENARIO	IMPLEMENTATION
Display a fixed-width column, such as an icon or ID column.	Set AutoSizeMode to None and Resizable to False for the column. Initialize its width by setting the Width property or by calling the control AutoResizeColumn method after filling the control with data.
Adjust sizes automatically whenever cell contents change to avoid clipping and to optimize the use of space.	Set an automatic sizing property to a value that represents a content-based sizing mode. To avoid a performance penalty when working with large amounts of data, use a sizing mode that calculates displayed rows only.
Adjust sizes to fit values in displayed rows to avoid performance penalties when working with many rows.	Use the appropriate sizing-mode enumeration values with automatic or programmatic resizing. To adjust sizes to fit values in newly displayed rows while scrolling, call a resizing method in a Scroll event handler. To customize user double-click resizing so that only values in displayed rows determine the new sizes, call a resizing method in a RowDividerDoubleClick or ColumnDividerDoubleClick event handler.
Adjust sizes to fit cell contents only at specific times to avoid performance penalties or to enable user resizing.	Call a content-based resizing method in an event handler. For example, use the DataBindingComplete event to initialize sizes after binding, and handle the CellValidated or CellValueChanged event to adjust sizes to compensate for user edits or changes in a bound data source.
Adjust row heights for multiline cell contents.	<p>Ensure that column widths are appropriate for displaying paragraphs of text and use automatic or programmatic content-based row sizing to adjust the heights. Also ensure that cells with multiline content are displayed using a WrapMode cell style value of True.</p> <p>Typically, you will use an automatic column sizing mode to maintain column widths or set them to specific widths before row heights are adjusted.</p>

Resizing with the Mouse

By default, users can resize rows, columns, and headers that do not use an automatic sizing mode based on cell values. To prevent users from resizing with other modes, such as column fill mode, set one or more of the following [DataGridView](#) properties:

- [AllowUserToResizeColumns](#)
- [AllowUserToResizeRows](#)
- [ColumnHeadersHeightSizeMode](#)
- [RowHeadersWidthSizeMode](#)

You can also prevent users from resizing individual rows or columns by setting their [Resizable](#) properties. By default, the [Resizable](#) property value is based on the [AllowUserToResizeColumns](#) property value for columns and the [AllowUserToResizeRows](#) property value for rows. If you explicitly set [Resizable](#) to [True](#) or [False](#), however, the specified value overrides the control value for that row or column. Set [Resizable](#) to [NotSet](#) to restore the inheritance.

Because [NotSet](#) restores the value inheritance, the [Resizable](#) property will never return a [NotSet](#) value unless the row or column has not been added to a [DataGridView](#) control. If you need to determine whether the [Resizable](#)

property value of a row or column is inherited, examine its [State](#) property. If the [State](#) value includes the [ResizableSet](#) flag, the [Resizable](#) property value is not inherited.

Automatic Sizing

There are two kinds of automatic sizing in the [DataGridView](#) control: column fill mode and content-based automatic sizing.

Column fill mode causes the visible columns in the control to fill the width of the control's display area. For more information about this mode, see [Column Fill Mode in the Windows Forms DataGridView Control](#).

You can also configure rows, columns, and headers to automatically adjust their sizes to fit their cell contents. In this case, size adjustment occurs whenever cell contents change.

NOTE

If you maintain cell values in a custom data cache using virtual mode, automatic sizing occurs when the user edits a cell value but does not occur when you alter a cached value outside of a [CellValuePushed](#) event handler. In this case, call the [UpdateCellValue](#) method to force the control to update the cell display and apply the current automatic sizing modes.

If content-based automatic sizing is enabled for one dimension only—that is, for rows but not columns, or for columns but not rows—and [WrapMode](#) is also enabled, size adjustment also occurs whenever the other dimension changes. For example, if rows but not columns are configured for automatic sizing and [WrapMode](#) is enabled, users can drag column dividers to change the width of a column and row heights will automatically adjust so that cell contents are still fully displayed.

If you configure both rows and columns for content-based automatic sizing and [WrapMode](#) is enabled, the [DataGridView](#) control will adjust sizes whenever cell contents changed and will use an ideal cell height-to-width ratio when calculating new sizes.

To configure the sizing mode for headers and rows and for columns that do not override the control value, set one or more of the following [DataGridView](#) properties:

- [ColumnHeadersHeightSizeMode](#)
- [RowHeadersWidthSizeMode](#)
- [AutoSizeColumnsMode](#)
- [AutoSizeRowsMode](#)

To override the control's column sizing mode for an individual column, set its [AutoSizeMode](#) property to a value other than [NotSet](#). The sizing mode for a column is actually determined by its [InheritedAutoSizeMode](#) property. The value of this property is based on the column's [AutoSizeMode](#) property value unless that value is [NotSet](#), in which case the control's [AutoSizeColumnsMode](#) value is inherited.

Use content-based automatic resizing with caution when working with large amounts of data. To avoid performance penalties, use the automatic sizing modes that calculate sizes based only on the displayed rows rather than analyzing every row in the control. For maximum performance, use programmatic resizing instead so that you can resize at specific times, such as immediately after new data is loaded.

Content-based automatic sizing modes do not affect rows, columns, or headers that you have hidden by setting the row or column [Visible](#) property or the control [RowHeadersVisible](#) or [ColumnHeadersVisible](#) properties to `false`. For example, if a column is hidden after it is automatically sized to fit a large cell value, the hidden column will not change its size if the row containing the large cell value is deleted. Automatic sizing does not occur when visibility changes, so changing the column [Visible](#) property back to `true` will not force it to recalculate its size based on its current contents.

Programmatic content-based resizing affects rows, columns, and headers regardless of their visibility.

Programmatic Resizing

When automatic sizing is disabled, you can programmatically set the exact width or height of rows, columns, or headers through the following properties:

- [DataGridView.RowHeadersWidth](#)
- [DataGridView.ColumnHeadersHeight](#)
- [DataGridViewRow.Height](#)
- [DataGridViewColumn.Width](#)

You can also programmatically resize rows, columns, and headers to fit their contents using the following methods:

- [AutoResizeColumn](#)
- [AutoResizeColumns](#)
- [AutoResizeColumnHeadersHeight](#)
- [AutoResizeRow](#)
- [AutoResizeRows](#)
- [AutoResizeRowHeadersWidth](#)

These methods will resize rows, columns, or headers once rather than configuring them for continuous resizing. The new sizes are automatically calculated to display all cell contents without clipping. When you programmatically resize columns that have [InheritedAutoSizeMode](#) property values of [Fill](#), however, the calculated content-based widths are used to proportionally adjust the column [FillWeight](#) property values, and the actually column widths are then calculated according to these new proportions so that all columns fill the available display area of the control.

Programmatic resizing is useful to avoid performance penalties with continuous resizing. It is also useful to provide initial sizes for user-resizable rows, columns, and headers, and for column fill mode.

You will typically call the programmatic resizing methods at specific times. For example, you might programmatically resize all columns immediately after loading data, or you might programmatically resize a specific row after a particular cell value has been modified.

Customizing Content-based Sizing Behavior

You can customize sizing behaviors when working with derived [DataGridView](#) cell, row, and column types by overriding the [DataGridViewCell.GetPreferredSize](#), [DataGridViewRow.GetPreferredSize](#), or [DataGridViewColumn.GetPreferredWidth](#) methods or by calling protected resizing method overloads in a derived [DataGridView](#) control. The protected resizing method overloads are designed to work in pairs to achieve an ideal cell height-to-width ratio, avoiding overly wide or tall cells. For example, if you call the

`AutoResizeRows(DataGridViewAutoSizeRowsMode, Boolean)` overload of the [AutoResizeRows](#) method and pass in a value of `false` for the [Boolean](#) parameter, the overload will calculate the ideal heights and widths for cells in the row, but it will adjust the row heights only. You must then call the [AutoResizeColumns](#) method to adjust the column widths to the calculated ideal.

Content-based Sizing Options

The enumerations used by sizing properties and methods have similar values for content-based sizing. With these values, you can limit which cells are used to calculate the preferred sizes. For all sizing enumerations, values with names that refer to displayed cells limit their calculations to cells in displayed rows. Excluding rows is useful to avoid a performance penalty when you are working with a large quantity of rows. You can also restrict calculations to cell values in header or nonheader cells.

See Also

[DataGridView](#)

[DataGridView.AllowUserToResizeColumns](#)

[DataGridView.AllowUserToResizeRows](#)

[DataGridView.ColumnHeadersHeightSizeMode](#)

[DataGridView.RowHeadersWidthSizeMode](#)

[DataGridViewBand.Resizable](#)

[DataGridView.AutoSizeColumnsMode](#)

[DataGridView.AutoSizeRowsMode](#)

[DataGridViewColumn.AutoSizeMode](#)

[DataGridViewColumn.InheritedAutoSizeMode](#)

[DataGridView.RowHeadersWidth](#)

[DataGridView.ColumnHeadersHeight](#)

[DataGridViewRow.Height](#)

[DataGridViewColumn.Width](#)

[DataGridView.AutoResizeColumn](#)

[DataGridView.AutoResizeColumns](#)

[DataGridView.AutoResizeColumnHeadersHeight](#)

[DataGridView.AutoResizeRow](#)

[DataGridView.AutoResizeRows](#)

[DataGridView.AutoScaleRowHeadersWidth](#)

[DataGridViewAutoSizeRowMode](#)

[DataGridViewAutoSizeColumnsMode](#)

[DataGridViewAutoSizeColumnsMode](#)

[DataGridViewColumnHeadersHeightSizeMode](#)

[DataGridViewRowHeadersWidthSizeMode](#)

[Resizing Columns and Rows in the Windows Forms DataGridView Control](#)

[Column Fill Mode in the Windows Forms DataGridView Control](#)

[How to: Set the Sizing Modes of the Windows Forms DataGridView Control](#)

Column Fill Mode in the Windows Forms DataGridView Control

5/4/2018 • 10 min to read • [Edit Online](#)

In column fill mode, the [DataGridView](#) control resizes its columns automatically so that they fill the width of the available display area. The control does not display the horizontal scroll bar except when it is necessary to keep the width of every column equal to or greater than its [MinimumWidth](#) property value.

The sizing behavior of each column depends on its [InheritedAutoSizeMode](#) property. The value of this property is inherited from the column's [AutoSizeMode](#) property or the control's [AutoSizeColumnsMode](#) property if the column value is [NotSet](#) (the default value).

Each column can have a different size mode, but any columns with a size mode of [Fill](#) will share the display-area width that is not used by the other columns. This width is divided among the fill-mode columns in proportions relative to their [FillWeight](#) property values. For example, if two columns have [FillWeight](#) values of 100 and 200, the first column will be half as wide as the second column.

User Resizing in Fill Mode

Unlike sizing modes that resize based on cell contents, fill mode does not prevent users from resizing columns that have [Resizable](#) property values of `true`. When a user resizes a fill-mode column, any fill-mode columns after the resized column (to the right if [RightToLeft](#) is `false`; otherwise, to the left) are also resized to compensate for the change in the available width. If there are no fill-mode columns after the resized column, then all other fill-mode columns in the control are resized to compensate. If there are no other fill-mode columns in the control, the resize is ignored. If a column that is not in fill mode is resized, all fill-mode columns in the control change sizes to compensate.

After resizing a fill-mode column, the [FillWeight](#) values for all columns that changed are adjusted proportionally. For example, if four fill-mode columns have [FillWeight](#) values of 100, resizing the second column to half its original width will result in [FillWeight](#) values of 100, 50, 125, and 125. Resizing a column that is not in fill mode will not change any [FillWeight](#) values because the fill-mode columns will simply resize to compensate while retaining the same proportions.

Content-Based FillWeight Adjustment

You can initialize [FillWeight](#) values for fill-mode columns by using the [DataGridView](#) automatic resizing methods, such as the [AutoResizeColumns](#) method. This method first calculates the widths required by columns to display their contents. Next, the control adjusts the [FillWeight](#) values for all fill-mode columns so that their proportions match the proportions of the calculated widths. Finally, the control resizes the fill-mode columns using the new [FillWeight](#) proportions so that all columns in the control fill the available horizontal space.

Example

Description

By using appropriate values for the [AutoSizeMode](#), [MinimumWidth](#), [FillWeight](#), and [Resizable](#) properties, you can customize the column-sizing behaviors for many different scenarios.

The following demonstration code enables you to experiment with different values for the [AutoSizeMode](#), [FillWeight](#), and [MinimumWidth](#) properties of different columns. In this example, a [DataGridView](#) control is bound to its own [Columns](#) collection, and one column is bound to each of the [HeaderText](#), [AutoSizeMode](#), [FillWeight](#),

[MinimumWidth](#), and [Width](#) properties. Each of the columns is also represented by a row in the control, and changing values in a row will update the properties of the corresponding column so that you can see how the values interact.

Code

```
using System;
using System.ComponentModel;
using System.Reflection;
using System.Windows.Forms;

public class Form1 : Form
{
    [STAThread]
    public static void Main()
    {
        Application.Run(new Form1());
    }

    private DataGridView dataGridView1 = new DataGridView();

    public Form1()
    {
        dataGridView1.Dock = DockStyle.Fill;
        Controls.Add(dataGridView1);
        InitializeDataGridView();
        Width *= 2;
        Text = "Column Fill-Mode Demo";
    }

    private void InitializeDataGridView()
    {
        // Add columns to the DataGridView, binding them to the
        // specified DataGridViewColumn properties.
        AddReadOnlyColumn("HeaderText", "Column");
        AddColumn("AutoSizeMode");
        AddColumn("FillWeight");
        AddColumn("MinimumWidth");
        AddColumn("Width");

        // Bind the DataGridView to its own Columns collection.
        dataGridView1.AutoGenerateColumns = false;
        dataGridView1.DataSource = dataGridView1.Columns;

        // Configure the DataGridView so that users can manually change
        // only the column widths, which are set to fill mode.
        dataGridView1.AllowUserToAddRows = false;
        dataGridView1.AllowUserToDeleteRows = false;
        dataGridView1.AllowUserToResizeRows = false;
        dataGridView1.RowHeadersWidthSizeMode =
            DataGridViewRowHeadersWidthSizeMode.DisableResizing;
        dataGridView1.ColumnHeadersHeight =
            DataGridViewColumnHeadersHeightSizeMode.DisableResizing;
        dataGridView1.AutoSizeColumnsMode =
            DataGridViewAutoSizeColumnsMode.Fill;

        // Configure the top left header cell as a reset button.
        dataGridView1.TopLeftHeaderCell.Value = "reset";
        dataGridView1.TopLeftHeaderCell.Style.ForeColor =
            System.Drawing.Color.Blue;

        // Add handlers to DataGridView events.
        dataGridView1.CellClick +=
            new DataGridViewEventHandler(dataGridView1_CellClick);
        dataGridView1.ColumnWidthChanged += new
            DataGridViewColumnEventHandler(dataGridView1_ColumnWidthChanged);
        dataGridView1.CurrentCellDirtyStateChanged +=
```

```

        new EventHandler(dataGridView1_CurrentCellDirtyStateChanged);
        dataGridView1.DataError +=
            new DataGridViewDataErrorEventHandler(dataGridView1_DataError);
        dataGridView1.CellEndEdit +=
            new DataGridViewCellEventHandler(dataGridView1_CellEndEdit);
        dataGridView1.CellValueChanged +=
            new DataGridViewCellEventHandler(dataGridView1_CellValueChanged);
    }

    private void AddReadOnlyColumn(String dataPropertyName, String columnName)
    {
        AddColumn(typeof(DataGridViewColumn), dataPropertyName, true,
                  columnName);
    }

    private void AddColumn(String dataPropertyName)
    {
        AddColumn(typeof(DataGridViewColumn), dataPropertyName, false,
                  dataPropertyName);
    }

    // Adds a column to the DataGridView control, binding it to specified
    // property of the specified type and optionally making it read-only.
    private void AddColumn()
    {
        Type type,
        String dataPropertyName,
        Boolean readOnly,
        String columnName)
    {
        // Retrieve information about the property through reflection.
        PropertyInfo property = type.GetProperty(dataPropertyName);

        // Confirm that the property exists and is accessible.
        if (property == null) throw new ArgumentException("No accessible " +
                                                       dataPropertyName + " property was found in the " + type.Name + " type.");

        // Confirm that the property is browsable.
       BrowsableAttribute[] browsables = (BrowsableAttribute[])
            property.GetCustomAttributes(typeof(BrowsableAttribute), false);
        if (browsables.Length > 0 && !browsables[0].Browsable)
        {
            throw new ArgumentException("The " + dataPropertyName + " property has a " +
                                       "Browsable(false) attribute, and therefore cannot be bound.");
        }

        // Create and initialize a column, using a combo box column for
        // enumeration properties, a check box column for Boolean properties,
        // and a text box column otherwise.
        DataGridViewColumn column;
        Type valueType = property.PropertyType;
        if (valueType.IsEnum)
        {
            column = new DataGridViewComboBoxColumn();

            // Populate the drop-down list with the enumeration values.
            ((DataGridViewComboBoxColumn)column).DataSource
                = Enum.GetValues(valueType);
        }
        else if (valueType.Equals(typeof(Boolean)))
        {
            column = new DataGridViewCheckBoxColumn();
        }
        else
        {
            column = new DataGridViewTextBoxColumn();
        }

        // Initialize and bind the column.
        column.ValueType = valueType;
    }
}

```

```

        column.Name = columnName;
        column.DataPropertyName = dataPropertyName;
        column.ReadOnly = readOnly;

        // Add the column to the control.
        dataGridView1.Columns.Add(column);
    }

private void ResetDataGridView()
{
    dataGridView1.CancelEdit();
    dataGridView1.Columns.Clear();
    dataGridView1.DataSource = null;
    InitializeDataGridView();
}

private void dataGridView1_CellClick(
    object sender, DataGridViewCellEventArgs e)
{
    if (e.ColumnIndex == -1 && e.RowIndex == -1)
    {
        ResetDataGridView();
    }
}

private void dataGridView1_ColumnWidthChanged(
    object sender, DataGridViewColumnEventArgs e)
{
    // Invalidate the row corresponding to the column that changed
    // to ensure that the FillWeight and Width entries are updated.
    dataGridView1.InvalidateRow(e.Column.Index);
}

private void dataGridView1_CurrentCellDirtyStateChanged(
    object sender, EventArgs e)
{
    // For combo box and check box cells, commit any value change as soon
    // as it is made rather than waiting for the focus to leave the cell.
    if (!dataGridView1.CurrentCell.OwningColumn.GetType()
        .Equals(typeof(DataGridViewTextBoxColumn)))
    {
        dataGridView1.CommitEdit(DataGridViewDataErrorContexts.Commit);
    }
}

private void dataGridView1_DataError(
    object sender, DataGridViewDataErrorEventArgs e)
{
    if (e.Exception == null) return;

    // If the user-specified value is invalid, cancel the change
    // and display the error icon in the row header.
    if ((e.Context & DataGridViewDataErrorContexts.Commit) != 0 &&
        (typeof(FormatException).IsAssignableFrom(e.Exception.GetType()) ||
        typeof(ArgumentException).IsAssignableFrom(e.Exception.GetType())))
    {
        dataGridView1.Rows[e.RowIndex].ErrorText =
            "The specified value is invalid.";
        e.Cancel = true;
    }
    else
    {
        // Rethrow any exceptions that aren't related to the user input.
        e.ThrowException = true;
    }
}

private void dataGridView1_CellEndEdit(
    object sender, DataGridViewCellEventArgs e)

```

```

{
    // Ensure that the error icon in the row header is hidden.
    dataGridView1.Rows[e.RowIndex].ErrorText = "";
}

private void dataGridView1_CellValueChanged(
    object sender, DataGridViewCellEventArgs e)
{
    // Retrieve the property to change.
    String nameOfPropertyToChange =
        dataGridView1.Columns[e.ColumnIndex].Name;
    PropertyInfo propertyToChange =
        typeof(DataGridViewColumn).GetProperty(nameOfPropertyToChange);

    // Retrieve the column to change.
    String nameOfColumnToChange =
        (String)dataGridView1["Column", e.RowIndex].Value;
    DataGridViewColumn columnToChange =
        dataGridView1.Columns[nameOfColumnToChange];

    // Use reflection to update the value of the column property.
    propertyToChange.SetValue(columnToChange,
        dataGridView1[nameOfPropertyToChange, e.RowIndex].Value, null);
}
}

```

```

Imports System
Imports System.ComponentModel
Imports System.Reflection
Imports System.Windows.Forms

Public Class Form1
    Inherits Form

    <STAThread()> _
    Public Shared Sub Main()
        Application.Run(New Form1())
    End Sub

    Private WithEvents dataGridView1 As New DataGridView()

    Public Sub New()
        dataGridView1.Dock = DockStyle.Fill
        Controls.Add(dataGridView1)
        InitializeDataGridView()
        Width = Width * 2
        Text = "Column Fill-Mode Demo"
    End Sub

    Private Sub InitializeDataGridView()

        ' Add columns to the DataGridView, binding them to the
        ' specified DataGridViewColumn properties.
        AddReadOnlyColumn("HeaderText", "Column")
        AddColumn("AutoSizeMode")
        AddColumn("FillWeight")
        AddColumn("MinimumWidth")
        AddColumn("Width")

        ' Bind the DataGridView to its own Columns collection.
        dataGridView1.AutoGenerateColumns = False
        dataGridView1.DataSource = dataGridView1.Columns

        ' Configure the DataGridView so that users can manually change
        ' only the column widths, which are set to fill mode.
        dataGridView1.AllowUserToAddRows = False
    End Sub

```

```

        dataGridView1.AllowUserToDeleteRows = False
        dataGridView1.AllowUserToResizeRows = False
        dataGridView1.RowHeadersWidthSizeMode = _
            DataGridViewRowHeadersWidthSizeMode.DisableResizing
        dataGridView1.ColumnHeadersHeightSizeMode = _
            DataGridViewColumnHeadersHeightSizeMode.DisableResizing
        dataGridView1.AutoSizeColumnsMode = _
            DataGridViewAutoSizeColumnsMode.Fill

        ' Configure the top left header cell as a reset button.
        dataGridView1.TopLeftHeaderCell.Value = "reset"
        dataGridView1.TopLeftHeaderCell.Style.ForeColor = _
            System.Drawing.Color.Blue

    End Sub

Private Sub AddReadOnlyColumn(ByVal dataPropertyName As String, _
    ByVal columnName As String)

    AddColumn(GetType(DataGridViewColumn), dataPropertyName, True, _
        columnName)
End Sub

Private Sub AddColumn(ByVal dataPropertyName As String)
    AddColumn(GetType(DataGridViewColumn), dataPropertyName, False, _
        dataPropertyName)
End Sub

' Adds a column to the DataGridView control, binding it to specified
' property of the specified type and optionally making it read-only.
Private Sub AddColumn( _
    ByVal type As Type, _
    ByVal dataPropertyName As String, _
    ByVal isReadOnly As Boolean, _
    ByVal columnName As String)

    ' Retrieve information about the property through reflection.
    Dim PropertyInfo1 As PropertyInfo = type.GetProperty(dataPropertyName)

    ' Confirm that the property exists and is accessible.
    If PropertyInfo1 Is Nothing Then
        Throw New ArgumentException("No accessible " & dataPropertyName & _
            " property was found in the " & type.Name & " type.")
    End If

    ' Confirm that the property is browsable.
    Dim browsables As BrowsableAttribute() = CType( _
        PropertyInfo1.GetCustomAttributes(GetType(BrowsableAttribute), _ _
        False), BrowsableAttribute())
    If browsables.Length > 0 AndAlso Not browsables(0).Browsable Then
        Throw New ArgumentException("The " & dataPropertyName & " property has a " & _
            "Browsable(false) attribute, and therefore cannot be bound.")
    End If

    ' Create and initialize a column, using a combo box column for
    ' enumeration properties, a check box column for Boolean properties,
    ' and a text box column otherwise.
    Dim column As DataGridViewColumn
    Dim valueType As Type = PropertyInfo1.PropertyType

    If valueType.IsEnum Then

        column = New DataGridViewComboBoxColumn()

        ' Populate the drop-down list with the enumeration values.
        CType(column, DataGridViewComboBoxColumn).DataSource = _
            [Enum].GetValues(valueType)

    ElseIf valueType.Equals(GetType(Boolean)) Then

```

```

        column = New DataGridViewCheckBoxColumn()
    Else
        column = New DataGridViewTextBoxColumn()
    End If

        ' Initialize and bind the column.
        column.ValueType = valueType
        column.Name = columnName
        column.DataPropertyName = dataPropertyName
        column.ReadOnly = isReadOnly

        ' Add the column to the control.
        dataGridView1.Columns.Add(column)

End Sub

Private Sub ResetDataGridView()
    dataGridView1.CancelEdit()
    dataGridView1.Columns.Clear()
    dataGridView1.DataSource = Nothing
    InitializeDataGridView()
End Sub

Private Sub dataGridView1_CellClick( _
    ByVal sender As Object, ByVal e As DataGridViewCellEventArgs) _
Handles dataGridView1.CellClick

    If e.ColumnIndex = -1 AndAlso e.RowIndex = -1 Then
        ResetDataGridView()
    End If

End Sub

Private Sub dataGridView1_ColumnWidthChanged( _
    ByVal sender As Object, ByVal e As DataGridViewColumnEventArgs) _
Handles dataGridView1.ColumnWidthChanged

    ' Invalidate the row corresponding to the column that changed
    ' to ensure that the FillWeight and Width entries are updated.
    dataGridView1.InvalidateRow(e.Column.Index)

End Sub

Private Sub dataGridView1_CurrentCellDirtyStateChanged( _
    ByVal sender As Object, ByVal e As EventArgs) _
Handles dataGridView1.CurrentCellDirtyStateChanged

    ' For combo box and check box cells, commit any value change as soon
    ' as it is made rather than waiting for the focus to leave the cell.
    If Not dataGridView1.CurrentCell.OwningColumn.GetType() _
        .Equals(GetType(DataGridViewTextBoxColumn)) Then

        dataGridView1.CommitEdit(DataGridViewDataErrorContexts.Commit)

    End If

End Sub

Private Sub dataGridView1_DataError( _
    ByVal sender As Object, ByVal e As DataGridViewDataErrorEventArgs) _
Handles dataGridView1.DataError

    If e.Exception Is Nothing Then Return

        ' If the user-specified value is invalid, cancel the change
        ' and display the error icon in the row header.
        If Not (e.Context And DataGridViewDataErrorContexts.Commit) = 0 AndAlso _
            (GetType(FormatException).IsAssignableFrom(e.Exception.GetType()) Or _
            GetType(ArgumentException).IsAssignableFrom(e.Exception.GetType())) Then

```

```

        dataGridView1.Rows(e.RowIndex).ErrorText = e.Exception.Message
        e.Cancel = True

    Else
        ' Rethrow any exceptions that aren't related to the user input.
        e.ThrowException = True
    End If

End Sub

Private Sub dataGridView1_CellEndEdit( _
    ByVal sender As Object, ByVal e As DataGridViewCellEventArgs) _
Handles dataGridView1.CellEndEdit

    ' Ensure that the error icon in the row header is hidden.
    dataGridView1.Rows(e.RowIndex).ErrorText = ""

End Sub

Private Sub dataGridView1_CellValueChanged( _
    ByVal sender As Object, ByVal e As DataGridViewCellEventArgs) _
Handles dataGridView1.CellValueChanged

    ' Ignore the change to the top-left header cell.
    If e.ColumnIndex < 0 Then Return

    ' Retrieve the property to change.
    Dim nameOfPropertyToChange As String = _
        dataGridView1.Columns(e.ColumnIndex).Name
    Dim propertyToChange As PropertyInfo = _
        GetType(DataGridViewColumn).GetProperty(nameOfPropertyToChange)

    ' Retrieve the column to change.
    Dim nameOfColumnToChange As String = _
        CStr(dataGridView1("Column", e.RowIndex).Value)
    Dim columnToChange As DataGridViewColumn = _
        dataGridView1.Columns(nameOfColumnToChange)

    ' Use reflection to update the value of the column property.
    propertyToChange.SetValue(columnToChange, _
        dataGridView1(nameOfPropertyToChange, e.RowIndex).Value, Nothing)

End Sub

End Class

```

Comments

To use this demonstration application:

- Change the size of the form. Observe how columns change their widths while retaining the proportions indicated by the [FillWeight](#) property values.
- Change the column sizes by dragging the column dividers with the mouse. Observe how the [FillWeight](#) values change.
- Change the [MinimumWidth](#) value for one column, then drag to resize the form. Observe how, when you make the form small enough, the [Width](#) values do not go below the [MinimumWidth](#) values.
- Change the [MinimumWidth](#) values for all columns to large numbers so that the combined values exceed the width of the control. Observe how the horizontal scroll bar appears.
- Change the [AutoSizeMode](#) values for some columns. Observe the effect when you resize columns or the form.

Compiling the Code

This example requires:

- References to the System, System.Drawing, and System.Windows.Forms assemblies.
- For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[DataGridView](#)

[DataGridView.AutoResizeColumns](#)

[DataGridView.AutoSizeColumnsMode](#)

[DataGridViewAutoSizeColumnsMode](#)

[DataGridViewColumn](#)

[DataGridViewColumn.InheritedAutoSizeMode](#)

[DataGridViewColumn.AutoSizeMode](#)

[DataGridViewAutoSizeColumnMode](#)

[DataGridViewColumn.FillWeight](#)

[DataGridViewColumn.MinimumWidth](#)

[DataGridViewColumn.Width](#)

[DataGridViewColumn.Resizable](#)

[Control.RightToLeft](#)

[Resizing Columns and Rows in the Windows Forms DataGridView Control](#)

How to: Set the Sizing Modes of the Windows Forms DataGridView Control

5/4/2018 • 3 min to read • [Edit Online](#)

The following procedures demonstrate some common scenarios that customize or combine the sizing options available for the [DataGridView](#) control and for specific columns in a control.

To create a fixed-width column

- Set the [AutoSizeMode](#) property to [None](#), the [Resizable](#) property to [False](#), the [ReadOnly](#) property to [true](#), and the [Width](#) property to an appropriate value.

```
DataGridViewTextBoxColumn idColumn =
    new DataGridViewTextBoxColumn();
idColumn.HeaderText = "ID";
idColumn.AutoSizeMode = DataGridViewAutoSizeColumnMode.None;
idColumn.Resizable = DataGridViewTriState.False;
idColumn.ReadOnly = true;
idColumn.Width = 20;
```

```
Dim idColumn As New DataGridViewTextBoxColumn()
idColumn.HeaderText = "ID"
idColumn.AutoSizeMode = DataGridViewAutoSizeColumnMode.None
idColumn.Resizable = DataGridViewTriState.False
idColumn.ReadOnly = True
idColumn.Width = 20
```

To create a column that adjusts its size to fit its content

- Set the [AutoSizeMode](#) property to a content-based sizing mode.

```
DataGridViewTextBoxColumn titleColumn =
    new DataGridViewTextBoxColumn();
titleColumn.HeaderText = "Title";
titleColumn.AutoSizeMode =
    DataGridViewAutoSizeColumnMode.AllCellsExceptHeader;
```

```
Dim titleColumn As New DataGridViewTextBoxColumn()
titleColumn.HeaderText = "Title"
titleColumn.AutoSizeMode =
    DataGridViewAutoSizeColumnMode.AllCellsExceptHeader
```

To create fill-mode columns for values of varying size and importance

- Set the [DataGridView.AutoSizeColumnsMode](#) property to [Fill](#) to set the sizing mode for all columns that do not override this value. Set the [FillWeight](#) properties of the columns to values that are proportional to their average content widths. Set the [MinimumWidth](#) properties of important columns to ensure partial content display.

```

dataGridView1.AutoSizeColumnsMode =
    DataGridViewAutoSizeColumnsMode.Fill;

DataGridViewTextBoxColumn subTitleColumn =
    new DataGridViewTextBoxColumn();
subTitleColumn.HeaderText = "Subtitle";
subTitleColumn.MinimumWidth = 50;
subTitleColumn.FillWeight = 100;

DataGridViewTextBoxColumn summaryColumn =
    new DataGridViewTextBoxColumn();
summaryColumn.HeaderText = "Summary";
summaryColumn.MinimumWidth = 50;
summaryColumn.FillWeight = 200;

DataGridViewTextBoxColumn contentColumn =
    new DataGridViewTextBoxColumn();
contentColumn.HeaderText = "Content";
contentColumn.MinimumWidth = 50;
contentColumn.FillWeight = 300;

```

```

dataGridView1.AutoSizeColumnsMode = _
    DataGridViewAutoSizeColumnsMode.Fill

Dim subTitleColumn As new DataGridViewTextBoxColumn()
subTitleColumn.HeaderText = "Subtitle"
subTitleColumn.MinimumWidth = 50
subTitleColumn.FillWeight = 100

Dim summaryColumn As new DataGridViewTextBoxColumn()
summaryColumn.HeaderText = "Summary"
summaryColumn.MinimumWidth = 50
summaryColumn.FillWeight = 200

Dim contentColumn As new DataGridViewTextBoxColumn()
contentColumn.HeaderText = "Content"
contentColumn.MinimumWidth = 50
contentColumn.FillWeight = 300

```

Example

The following complete code example provides a demonstration application that can help you understand the sizing options described in this topic.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

public class Form1 : Form
{
    [STAThreadAttribute()]
    public static void Main()
    {
        Application.Run(new Form1());
    }

    private DataGridView dataGridView1 = new DataGridView();

    public Form1()

```

```

{
    dataGridView1.Dock = DockStyle.Fill;
    Controls.Add(dataGridView1);
    Width *= 2;
    Text = "DataGridView Sizing Scenarios";
}

protected override void OnLoad(EventArgs e)
{
    DataGridViewTextBoxColumn idColumn =
        new DataGridViewTextBoxColumn();
    idColumn.HeaderText = "ID";
    idColumn.AutoSizeMode = DataGridViewAutoSizeColumnMode.None;
    idColumn.Resizable = DataGridViewTriState.False;
    idColumn.ReadOnly = true;
    idColumn.Width = 20;

    DataGridViewTextBoxColumn titleColumn =
        new DataGridViewTextBoxColumn();
    titleColumn.HeaderText = "Title";
    titleColumn.AutoSizeMode =
        DataGridViewAutoSizeColumnMode.AllCellsExceptHeader;

    dataGridView1.AutoSizeColumnsMode =
        DataGridViewAutoSizeColumnsMode.Fill;

    DataGridViewTextBoxColumn subTitleColumn =
        new DataGridViewTextBoxColumn();
    subTitleColumn.HeaderText = "Subtitle";
    subTitleColumn.MinimumWidth = 50;
    subTitleColumn.FillWeight = 100;

    DataGridViewTextBoxColumn summaryColumn =
        new DataGridViewTextBoxColumn();
    summaryColumn.HeaderText = "Summary";
    summaryColumn.MinimumWidth = 50;
    summaryColumn.FillWeight = 200;

    DataGridViewTextBoxColumn contentColumn =
        new DataGridViewTextBoxColumn();
    contentColumn.HeaderText = "Content";
    contentColumn.MinimumWidth = 50;
    contentColumn.FillWeight = 300;

    dataGridView1.Columns.AddRange(new DataGridViewTextBoxColumn[] {
        idColumn, titleColumn, subTitleColumn,
        summaryColumn, contentColumn });
    dataGridView1.Rows.Add(new String[] { "1",
        "A Short Title", "A Longer SubTitle",
        "A short description of the main point.",
        "The full contents of the topic, with detailed examples." });

    base.OnLoad(e);
}
}

```

```

Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Text
Imports System.Windows.Forms

Public Class Form1
    Inherits Form

```

```

<STAThreadAttribute()> _
Public Shared Sub Main()
    Application.Run(New Form1())
End Sub

Private dataGridView1 As New DataGridView()

Public Sub New()
    dataGridView1.Dock = DockStyle.Fill
    Controls.Add(dataGridView1)
    Width *= 2
    Text = "DataGridView Sizing Scenarios"
End Sub

Protected Overrides Sub OnLoad(ByVal e As System.EventArgs)

    Dim idColumn As New DataGridViewTextBoxColumn()
    idColumn.HeaderText = "ID"
    idColumn.AutoSizeMode = DataGridViewAutoSizeColumnsMode.None
    idColumn.Resizable = DataGridViewTriState.False
    idColumn.ReadOnly = True
    idColumn.Width = 20

    Dim titleColumn As New DataGridViewTextBoxColumn()
    titleColumn.HeaderText = "Title"
    titleColumn.AutoSizeMode = _
        DataGridViewAutoSizeColumnsMode.AllCellsExceptHeader

    dataGridView1.AutoSizeColumnsMode = _
        DataGridViewAutoSizeColumnsMode.Fill

    Dim subTitleColumn As New DataGridViewTextBoxColumn()
    subTitleColumn.HeaderText = "Subtitle"
    subTitleColumn.MinimumWidth = 50
    subTitleColumn.FillWeight = 100

    Dim summaryColumn As New DataGridViewTextBoxColumn()
    summaryColumn.HeaderText = "Summary"
    summaryColumn.MinimumWidth = 50
    summaryColumn.FillWeight = 200

    Dim contentColumn As New DataGridViewTextBoxColumn()
    contentColumn.HeaderText = "Content"
    contentColumn.MinimumWidth = 50
    contentColumn.FillWeight = 300

    dataGridView1.Columns.AddRange(New DataGridViewTextBoxColumn() { _
        idColumn, titleColumn, subTitleColumn, _
        summaryColumn, contentColumn})
    dataGridView1.Rows.Add(New String() {"1", _
        "A Short Title", "A Longer SubTitle", _
        "A short description of the main point.", _
        "The full contents of the topic, with detailed examples."})

    MyBase.OnLoad(e)
End Sub
End Class

```

To use this demonstration application:

- Change the size of the form. Observe how the fill-mode columns change their widths while retaining the proportions indicated by the **FillWeight** property values. Observe how a column's **MinimumWidth** prevents it from changing when the form is too small.
- Change the column sizes by dragging the column dividers with the mouse. Observe how some columns cannot be resized, and how resizable columns cannot be made narrower than their minimum widths.

Compiling the Code

This example requires:

- References to the System and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[DataGridView](#)
[DataGridViewColumn.AutoSizeMode](#)
[DataGridViewAutoSizeColumnsMode](#)
[DataGridViewColumn.Resizable](#)
[DataGridViewColumn.ReadOnly](#)
[DataGridViewColumn.Width](#)
[DataGridViewColumn.FillWeight](#)
[DataGridViewColumn.MinimumWidth](#)

How to: Programmatically Resize Cells to Fit Content in the Windows Forms DataGridView Control

5/4/2018 • 13 min to read • [Edit Online](#)

You can use the [DataGridView](#) control methods to resize rows, columns, and headers so that they display their entire values without truncation. You can use these methods to resize [DataGridView](#) elements at times of your choosing. Alternately, you can configure the control to resize these elements automatically whenever content changes. This can be inefficient, however, when you are working with large data sets or when your data changes frequently. For more information, see [Sizing Options in the Windows Forms DataGridView Control](#).

Typically, you will programmatically adjust [DataGridView](#) elements to fit their content only when you load new data from a data source or when the user has edited a value. This is useful to optimize performance, but it is also useful when you want to enable your users to manually resize rows and columns with the mouse.

The following code example demonstrates the options available to you for programmatic resizing.

Example

```
#using <System.Windows.Forms.dll>
#using <System.dll>
#using <System.Drawing.dll>

using namespace System;
using namespace System::Drawing;
using namespace System::Collections;
using namespace System::ComponentModel;
using namespace System::Windows::Forms;
public ref class ProgrammaticSizing: public System::Windows::Forms::Form
{
private:
    FlowLayoutPanel^ flowLayoutPanel1;
    Button^ button1;
    Button^ button2;
    Button^ button3;
    Button^ button4;
    Button^ button5;
    Button^ button6;
    Button^ button7;
    Button^ button8;
    Button^ button9;
    Button^ button10;
    Button^ button11;

public:
    ProgrammaticSizing()
    {
        button1 = gcnew Button;
        button2 = gcnew Button;
        button3 = gcnew Button;
        button4 = gcnew Button;
        button5 = gcnew Button;
        button6 = gcnew Button;
        button7 = gcnew Button;
        button8 = gcnew Button;
        button9 = gcnew Button;
        button10 = gcnew Button;
        button11 = gcnew Button;
        thirdColumnHeader = "Main Ingredients";
    }
}
```

```

boringMeatloaf = "ground beef";
boringMeatloafRanking = "*";
otherRestaurant = "Gomes's Saharan Sushi";
InitializeComponent();
AddDirections();
this->Load += gcnew EventHandler( this, &ProgrammaticSizing::InitializeDataGridView );
AddButton( button1, "Reset", gcnew EventHandler( this, &ProgrammaticSizing::ResetToDisorder ) );
AddButton( button2, "Change Column 3 Header", gcnew EventHandler( this,
&ProgrammaticSizing::ChangeColumn3Header ) );
AddButton( button3, "Change Meatloaf Recipe", gcnew EventHandler( this,
&ProgrammaticSizing::ChangeMeatloafRecipe ) );
AddButton( button10, "Change Restaurant 2", gcnew EventHandler( this,
&ProgrammaticSizing::ChangeRestaurant ) );
AddButtonsForProgrammaticResizing();
}

private:
void InitializeComponent()
{
    this->flowLayoutPanel1 = gcnew FlowLayoutPanel;
    this->flowLayoutPanel1->FlowDirection = FlowDirection::TopDown;
    this->flowLayoutPanel1->Location = Point(492,0);
    this->flowLayoutPanel1->AutoSize = true;
    this->AutoSize = true;
    this->Controls->Add( this->flowLayoutPanel1 );
    this->Text = this->GetType()->Name;
}

void AddDirections()
{
    Label^ directions = gcnew Label;
    directions->AutoSize = true;
    String^ newLine = Environment::NewLine;
    directions->Text = String::Format( "Press the buttons that start {0}with 'Change' to see how different
sizing {1}modes deal with content changes.", newLine, newLine );
    flowLayoutPanel1->Controls->Add( directions );
}

System::Drawing::Size startingSize;
String^ thirdColumnHeader;
String^ boringMeatloaf;
String^ boringMeatloafRanking;
bool boringRecipe;
bool shortMode;
DataGridView^ dataGridView1;
String^ otherRestaurant;
void InitializeDataGridView( Object^ /*sender*/, EventArgs^ /*ignoredToo*/ )
{
    this->dataGridView1 = gcnew DataGridView;
    this->dataGridView1->Location = Point(0,0);
    this->dataGridView1->Size = System::Drawing::Size( 292, 266 );
    this->Controls->Add( this->dataGridView1 );
    startingSize = System::Drawing::Size( 450, 400 );
    dataGridView1->Size = startingSize;
    AddColumns();
    PopulateRows();
    shortMode = false;
    boringRecipe = true;
}

void AddColumns()
{
    dataGridView1->ColumnCount = 4;
    dataGridView1->ColumnHeadersVisible = true;
    DataGridViewCellStyle ^ columnHeaderStyle = gcnew DataGridViewCellStyle;
    columnHeaderStyle->BackColor = Color::Aqua;
    columnHeaderStyle->Font = gcnew System::Drawing::Font( "Verdana",10,FontStyle::Bold );
    dataGridView1->ColumnHeadersDefaultCellStyle = columnHeaderStyle;
}

```

```

    dataGridView1->Columns[ 0 ]->Name = "Recipe";
    dataGridView1->Columns[ 1 ]->Name = "Category";
    dataGridView1->Columns[ 2 ]->Name = thirdColumnHeader;
    dataGridView1->Columns[ 3 ]->Name = "Rating";
}

void PopulateRows()
{
    array<String^>^row1 = {"Meatloaf", "Main Dish", boringMeatloaf, boringMeatloafRanking};
    array<String^>^row2 = {"Key Lime Pie", "Dessert", "lime juice, evaporated milk", "****"};
    array<String^>^row3 = {"Orange-Salsa Pork Chops", "Main Dish", "pork chops, salsa, orange juice", "****"};
    array<String^>^row4 = {"Black Bean and Rice Salad", "Salad", "black beans, brown rice", "****"};
    array<String^>^row5 = {"Chocolate Cheesecake", "Dessert", "cream cheese", "****"};
    array<String^>^row6 = {"Black Bean Dip", "Appetizer", "black beans, sour cream", "***"};
    array<Object^>^rows = {row1, row2, row3, row4, row5, row6};
    IEnumerator^ myEnum = rows->GetEnumerator();
    while ( myEnum->MoveNext() )
    {
        array<String^>^row = safe_cast<array<String^>>(myEnum->Current);
        dataGridView1->Rows->Add( row );
    }

    IEnumerator^ myEnum1 = safe_cast<IEnumerator^>(dataGridView1->Rows)->GetEnumerator();
    while ( myEnum1->MoveNext() )
    {
        DataGridViewRow ^ row = safe_cast<DataGridViewRow ^>(myEnum1->Current);
        if ( row->IsNewRow )
            break;
        row->HeaderCell->Value = String::Format( "Resturant {0}", row->Index );
    }
}

void AddButton( Button^ button, String^ buttonLabel, EventHandler^ handler )
{
    button->Text = buttonLabel;
    button->AutoSize = true;
    flowLayoutPanel1->Controls->Add( button );
    button->Click += handler;
}

void ResetToDisorder( Object^ /*sender*/, EventArgs^ /*e*/ )
{
    Controls->Remove( dataGridView1 );
    dataGridView1->Size = startingSize;
    dataGridView1->DataGridView::~DataGridView();
    InitializeDataGridView( nullptr, nullptr );
}

void ChangeColumn3Header( Object^ /*sender*/, EventArgs^ /*e*/ )
{
    Toggle( &shortMode );
    if ( shortMode )
        dataGridView1->Columns[ 2 ]->HeaderText = "S";
    else
        dataGridView1->Columns[ 2 ]->HeaderText = thirdColumnHeader;
}

void Toggle( interior_ptr<Boolean> toggleThis )
{
    *toggleThis = ! *toggleThis;
}

void ChangeMeatloafRecipe( Object^ /*sender*/, EventArgs^ /*e*/ )
{
    Toggle( &boringRecipe );
    if ( boringRecipe )
        SetMeatloaf( boringMeatloaf, boringMeatloafRanking );
    else
}

```

```

    }
    String^ greatMeatloafRecipe = "1 lb. lean ground beef, "
    "1/2 cup bread crumbs, 1/4 cup ketchup,"
    "1/3 tsp onion powder, "
    "1 clove of garlic, 1/2 pack onion soup mix "
    " dash of your favorite BBQ Sauce";
    SetMeatloaf( greatMeatloafRecipe, "***" );
}
}

void SetMeatloaf( String^ recipe, String^ rating )
{
    dataGridView1->Rows[ 0 ]->Cells[ 2 ]->Value = recipe;
    dataGridView1->Rows[ 0 ]->Cells[ 3 ]->Value = rating;
}

void ChangeRestaurant( Object^ /*sender*/, EventArgs^ /*ignored*/ )
{
    if ( dataGridView1->Rows[ 2 ]->HeaderCell->Value == otherRestaurant )
        dataGridView1->Rows[ 2 ]->HeaderCell->Value = "Restaurant 2";
    else
        dataGridView1->Rows[ 2 ]->HeaderCell->Value = otherRestaurant;
}

void AddButtonsForProgrammaticResizing()
{
    AddButton( button4, "Size Third Column", gcnew EventHandler( this,
&ProgrammaticSizing::SizeThirdColumnHeader ) );
    AddButton( button5, "Size Column Headers", gcnew EventHandler( this,
&ProgrammaticSizing::SizeColumnHeaders ) );
    AddButton( button6, "Size All Columns", gcnew EventHandler( this, &ProgrammaticSizing::SizeAllColumns ) );
    AddButton( button7, "Size Third Row", gcnew EventHandler( this, &ProgrammaticSizing::SizeThirdRow ) );
    AddButton( button8, "Size First Row Header Using All Headers", gcnew EventHandler( this,
&ProgrammaticSizing::SizeFirstRowHeaderToAllHeaders ) );
    AddButton( button9, "Size All Rows and Row Headers", gcnew EventHandler( this,
&ProgrammaticSizing::SizeAllRowsAndTheirHeaders ) );
    AddButton( button11, "Size All Rows ", gcnew EventHandler( this, &ProgrammaticSizing::SizeAllRows ) );
}

void SizeThirdColumnHeader( Object^ /*sender*/, EventArgs^ /*e*/ )
{
    dataGridView1->AutoSizeColumnsMode::ColumnHeader);
}

void SizeColumnHeaders( Object^ /*sender*/, EventArgs^ /*e*/ )
{
    int columnNumber;
    bool dontChangeColumnWidth;
    bool dontChangeRowHeadersWidth;
    dataGridView1->AutoSizeColumnsMode::AllCells);
}

void SizeAllColumns( Object^ /*sender*/, EventArgs^ /*e*/ )
{
    dataGridView1->AutoSizeColumnsMode::AllCells);
}

void SizeThirdRow( Object^ /*sender*/, EventArgs^ /*e*/ )
{
    dataGridView1->AutoSizeColumnsMode::AllCellsExceptHeader);
}

void SizeFirstRowHeaderToAllHeaders( Object^ /*sender*/, EventArgs^ /*e*/ )
{
}

```

```
    dataGridView1->AutoResizeRowHeadersWidth(0, DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders);
}

void SizeAllRowsAndTheirHeaders( Object^ /*sender*/, EventArgs^ /*e*/ )
{
    dataGridView1->AutoResizeRows(DataGridViewAutoSizeRowsMode::AllCells);
}

void SizeAllRows( Object^ /*sender*/, EventArgs^ /*e*/ )
{
    dataGridView1->AutoResizeRows(DataGridViewAutoSizeRowsMode::AllCellsExceptHeaders);
}

};

[STAThread]
int main()
{
    Application::EnableVisualStyles();
    Application::Run( gcnew ProgrammaticSizing );
}
```

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;

public class ProgrammaticSizing : System.Windows.Forms.Form
{
    private FlowLayoutPanel flowLayoutPanel1;
    private Button button1 = new Button();
    private Button button2 = new Button();
    private Button button3 = new Button();
    private Button button4 = new Button();
    private Button button5 = new Button();
    private Button button6 = new Button();
    private Button button7 = new Button();
    private Button button8 = new Button();
    private Button button9 = new Button();
    private Button button10 = new Button();
    private Button button11 = new Button();

    public ProgrammaticSizing()
    {
        InitializeComponent();
        AddDirections();
        this.Load += new EventHandler(InitializeDataGridView);

        addButton(button1, "Reset",
            new EventHandler(ResetToDisorder));
        addButton(button2, "Change Column 3 Header",
            new EventHandler(ChangeColumn3Header));
        addButton(button3, "Change Meatloaf Recipe",
            new EventHandler(ChangeMeatloafRecipe));
        addButton(button10, "Change Restaurant 2",
            new EventHandler(ChangeRestaurant));
        AddButtonsForProgrammaticResizing();
    }

    #region form code
    private void InitializeComponent()
    {
```

```

    }

        this.flowLayoutPanel1 = new FlowLayoutPanel();
        this.flowLayoutPanel1.FlowDirection
            = FlowDirection.TopDown;
        this.flowLayoutPanel1.Location = new Point(492, 0);
        this.flowLayoutPanel1.AutoSize = true;

        this.AutoSize = true;
        this.Controls.Add(this.flowLayoutPanel1);
        this.Text = this.GetType().Name;
    }

private void AddDirections()
{
    Label directions = new Label();
    directions.AutoSize = true;
    String newLine = Environment.NewLine;
    directions.Text = "Press the buttons that start " + newLine
        + "with 'Change' to see how different sizing " + newLine
        + "modes deal with content changes.";

    flowLayoutPanel1.Controls.Add(directions);
}
#endregion

[STAThread]
static void Main()
{
    Application.EnableVisualStyles();
    Application.Run(new ProgrammaticSizing());
}

private Size startingSize;
private string thirdColumnHeader = "Main Ingredients";
private string boringMeatloaf = "ground beef";
private string boringMeatloafRanking = "*";
private bool boringRecipe;
private bool shortMode;
private DataGridView dataGridView1;
private string otherRestaurant = "Gomes's Saharan Sushi";

private void InitializeDataGridView(Object sender,
    EventArgs ignoredToo)
{
    this.dataGridView1 = new DataGridView();
    this.dataGridView1.Location = new Point(0, 0);
    this.dataGridView1.Size = new Size(292, 266);
    this.Controls.Add(this.dataGridView1);
    startingSize = new Size(450, 400);
    dataGridView1.Size = startingSize;

    AddColumns();
    PopulateRows();

    shortMode = false;
    boringRecipe = true;
}

private void AddColumns()
{
    dataGridView1.ColumnCount = 4;
    dataGridView1.ColumnHeadersVisible = true;

    DataGridViewCellStyle columnHeaderStyle =
        new DataGridViewCellStyle();

    columnHeaderStyle.BackColor = Color.Aqua;
    columnHeaderStyle.Font = new Font("Verdana", 10,
        FontStyle.Bold);
}

```

```

dataGridView1.ColumnHeadersDefaultCellStyle =
    columnHeaderStyle;

dataGridView1.Columns[0].Name = "Recipe";
dataGridView1.Columns[1].Name = "Category";
dataGridView1.Columns[2].Name = thirdColumnHeader;
dataGridView1.Columns[3].Name = "Rating";
}

private void PopulateRows()
{
    string[] row1 = {
        "Meatloaf", "Main Dish",
        boringMeatloaf, boringMeatloafRanking
    };
    string[] row2 = {
        "Key Lime Pie", "Dessert",
        "lime juice, evaporated milk", "****"
    };
    string[] row3 = {
        "Orange-Salsa Pork Chops", "Main Dish",
        "pork chops, salsa, orange juice", "****"
    };
    string[] row4 = {
        "Black Bean and Rice Salad", "Salad",
        "black beans, brown rice", "****"
    };
    string[] row5 = {
        "Chocolate Cheesecake", "Dessert",
        "cream cheese", "***"
    };
    string[] row6 = {
        "Black Bean Dip",
        "Appetizer", "black beans, sour cream", "***"
    };
    object[] rows = new object[] {
        row1, row2, row3, row4, row5, row6
    };

    foreach (string[] row in rows)
        dataGridView1.Rows.Add(row);
    foreach (DataGridViewRow row in dataGridView1.Rows)
    {
        if (row.IsNewRow) break;
        row.HeaderCell.Value = "Restaurant " + row.Index;
    }
}

private void AddButton(Button button, string buttonLabel,
    EventHandler handler)
{
    button.Text = buttonLabel;
    button.AutoSize = true;
    flowLayoutPanel1.Controls.Add(button);
    button.Click += handler;
}

private void ResetToDisorder(Object sender, EventArgs e)
{
    Controls.Remove(dataGridView1);
    dataGridView1.Size = startingSize;
    dataGridView1.Dispose();
    InitializeDataGridView(null, null);
}

private void ChangeColumn3Header(Object sender, EventArgs e)
{
    Toggle(ref shortMode);
    if (shortMode)

```

```

        dataGridView1.Columns[2].HeaderText = "S";
    else
        dataGridView1.Columns[2].HeaderText =
            thirdColumnHeader;
    }

private static void Toggle(ref Boolean toggleThis)
{
    toggleThis = !toggleThis;
}

private void ChangeMeatloafRecipe(Object sender,
    EventArgs e)
{
    Toggle(ref boringRecipe);

    if (boringRecipe)
        SetMeatloaf(boringMeatloaf, boringMeatloafRanking);
    else
    {
        string greatMeatloafRecipe =
            "1 lb. lean ground beef, "
            + "1/2 cup bread crumbs, 1/4 cup ketchup,"
            + "1/3 tsp onion powder, "
            + "1 clove of garlic, 1/2 pack onion soup mix "
            + " dash of your favorite BBQ Sauce";

        SetMeatloaf(greatMeatloafRecipe, "***");
    }
}

private void SetMeatloaf(string recipe, string rating)
{
    dataGridView1.Rows[0].Cells[2].Value = recipe;
    dataGridView1.Rows[0].Cells[3].Value = rating;
}

private void ChangeRestaurant(Object sender,
    EventArgs ignored)
{
    if (dataGridView1.Rows[2].HeaderCell.Value.Equals(otherRestaurant))
    {
        dataGridView1.Rows[2].HeaderCell.Value =
            "Restaurant 2";
    }
    else
    {
        dataGridView1.Rows[2].HeaderCell.Value = otherRestaurant;
    }
}

#region "programmatic resizing"

private void AddButtonsForProgrammaticResizing()
{
    AddButton(button4, "Size Third Column",
        new EventHandler(SizeThirdColumnHeader));
    AddButton(button5, "Size Column Headers",
        new EventHandler(SizeColumnHeaders));
    AddButton(button6, "Size All Columns",
        new EventHandler(SizeAllColumns));
    AddButton(button7, "Size Third Row",
        new EventHandler(SizeThirdRow));
    AddButton(button8, "Size First Row Header Using All Headers",
        new EventHandler(SizeFirstRowHeaderToAllHeaders));
    AddButton(button9, "Size All Rows and Row Headers",
        new EventHandler(SizeAllRowsAndTheirHeaders));
    AddButton(button11, "Size All Rows ",
        new EventHandler(SizeAllRows));
}

```

```

}

private void SizeThirdColumnHeader(Object sender,
    EventArgs e)
{
    dataGridView1.AutoResizeColumn(
        2, DataGridViewAutoSizeColumnsMode.ColumnHeader);
}

private void SizeColumnHeaders(Object sender, EventArgs e)
{
    dataGridView1.AutoResizeColumnHeadersHeight(2);
}

private void SizeAllColumns(Object sender, EventArgs e)
{
    dataGridView1.AutoResizeColumns(
        DataGridViewAutoSizeColumnsMode.AllCells);
}

private void SizeThirdRow(Object sender, EventArgs e)
{
    dataGridView1.AutoResizeRow(
        2, DataGridViewAutoSizeRowMode.AllCellsExceptHeader);
}

private void SizeFirstRowHeaderToAllHeaders(Object sender, EventArgs e)
{
    dataGridView1.AutoResizeRowHeadersWidth(
        0, DataGridViewRowHeadersWidthSizeMode.AutoSizeToAllHeaders);
}

private void SizeAllRowsAndTheirHeaders(Object sender, EventArgs e)
{
    dataGridView1.AutoResizeRows(
        DataGridViewAutoSizeRowsMode.AllCells);
}

private void SizeAllRows(Object sender,
    EventArgs e)
{
    dataGridView1.AutoResizeRows(
        DataGridViewAutoSizeRowsMode.AllCellsExceptHeaders);
}
#endregion
}

```

```

Public Class ProgrammaticSizing
    Inherits System.Windows.Forms.Form

#Region " form setup "
    Public Sub New()
        MyBase.New()
        InitializeComponent()
        AddDirections()

        AddButton(Button1, "Reset")
        AddButton(Button2, "Change Column 3 Header")
        AddButton(Button3, "Change Meatloaf Recipe")
        AddButton(Button10, "Change Restaurant 2")
        AddButtonsForProgrammaticResizing()
    End Sub

    Friend WithEvents DataGridView1 As DataGridView
    Friend WithEvents Button1 As Button = New Button()
    Friend WithEvents Button2 As Button = New Button()
    Friend WithEvents Button3 As Button = New Button()

```

```

Friend WithEvents Button4 As Button = New Button()
Friend WithEvents Button5 As Button = New Button()
Friend WithEvents Button6 As Button = New Button()
Friend WithEvents FlowLayoutPanel1 As FlowLayoutPanel
Friend WithEvents Button7 As Button = New Button()
Friend WithEvents Button8 As Button = New Button()
Friend WithEvents Button9 As Button = New Button()
Friend WithEvents Button10 As Button = New Button()
Friend WithEvents Button11 As Button = New Button()

Private Sub InitializeComponent()
    Me.FlowLayoutPanel1 = New FlowLayoutPanel
    Me.FlowLayoutPanel1.FlowDirection = FlowDirection.TopDown
    Me.FlowLayoutPanel1.Location = New Point(454, 0)
    Me.FlowLayoutPanel1.AutoSize = True

    Me.AutoSize = True
    Me.Text = Me.GetType().Name
    Me.Controls.Add(FlowLayoutPanel1)
End Sub

Private Sub AddDirections()
    Dim directions As New Label()
    directions.AutoSize = True
    Dim newLine As String = Environment.NewLine
    directions.Text = "Press the buttons that start " & newLine _
        & "with 'Change' to see how different sizing " & newLine _
        & "modes deal with content changes."
    FlowLayoutPanel1.Controls.Add(directions)
End Sub

Public Shared Sub Main()
    Application.Run(New ProgrammaticSizing())
End Sub
#End Region

Private startingSize As Size
Private thirdColumnHeader As String = "Main Ingredients"
Private boringMeatloaf As String = "ground beef"
Private boringMeatloafRanking As String = "*"
Private boringRecipe As Boolean
Private shortMode As Boolean
Private otherRestaurant As String = "Gomes's Saharan Sushi"

Private Sub InitializeDataGridView(ByVal ignored As Object, _
    ByVal ignoredToo As EventArgs) Handles Me.Load
    Me.DataGridView1 = New System.Windows.Forms.DataGridView
    Me.DataGridView1.Name = "DataGridView1"
    Me.DataGridView1.Size = New System.Drawing.Size(292, 266)
    Me.Controls.Add(Me.DataGridView1)

    startingSize = New Size(450, 400)
    DataGridView1.Size = startingSize

    AddColumns()
    PopulateRows()

    shortMode = False
    boringRecipe = True
End Sub

Private Sub AddColumns()
    DataGridView1.ColumnCount = 4
    DataGridView1.ColumnHeadersVisible = True

    Dim columnHeaderStyle As New DataGridViewCellStyle
    columnHeaderStyle.BackColor = Color.Aqua
    columnHeaderStyle.Font = New Font("Verdana", 10, _

```

```

        FontStyle.Bold)
DataGridView1.ColumnHeadersDefaultCellStyle = _
columnHeaderStyle

DataGridView1.Columns(0).Name = "Recipe"
DataGridView1.Columns(1).Name = "Category"
DataGridView1.Columns(2).Name = thirdColumnHeader
DataGridView1.Columns(3).Name = "Rating"
End Sub

Private Sub PopulateRows()

Dim row1 As String() = New String() _
{"Meatloaf", "Main Dish", boringMeatloaf, _
boringMeatloafRanking}
Dim row2 As String() = New String() _
{"Key Lime Pie", "Dessert", _
"lime juice, evaporated milk", _
"****"}
Dim row3 As String() = New String() _
 {"Orange-Salsa Pork Chops", "Main Dish", _
 "pork chops, salsa, orange juice", "****"}
Dim row4 As String() = New String() _
 {"Black Bean and Rice Salad", "Salad", _
 "black beans, brown rice", _
 "****"}
Dim row5 As String() = New String() _
 {"Chocolate Cheesecake", "Dessert", "cream cheese", _
 "****"}
Dim row6 As String() = New String() _
 {"Black Bean Dip", "Appetizer", _
 "black beans, sour cream", "****"}
Dim rows As Object() = New Object() {row1, row2, row3, _
row4, row5, row6}

Dim rowArray As String()
For Each rowArray In rows
    DataGridView1.Rows.Add(rowArray)
Next

For Each row As DataGridViewRow In DataGridView1.Rows
    If row.IsNewRow Then Continue For
    row.HeaderCell.Value = "Restaurant " & row.Index
Next

End Sub

Private Sub AddButton(ByVal button As Button, _
ByVal buttonLabel As String)

button.Text = buttonLabel
button.AutoSize = True
button.TabIndex = FlowLayoutPanel1.Controls.Count
FlowLayoutPanel1.Controls.Add(button)
End Sub

Private Sub ResetToDisorder(ByVal sender As Object, _
ByVal e As System.EventArgs) _
Handles Button1.Click

Controls.Remove(DataGridView1)
DataGridView1.Size = startingSize
DataGridView1.Dispose()
InitializeDataGridView(Nothing, Nothing)
End Sub

Private Sub ChangeColumn3Header(ByVal sender As Object, _
ByVal e As System.EventArgs) _
Handles Button2.Click

```

```

    Toggle(shortMode)
    If shortMode Then DataGridView1.Columns(2).HeaderText = "S" -
    Else DataGridView1.Columns(2).HeaderText = thirdColumnHeader
End Sub

Private Shared Function Toggle(ByRef toggleThis As Boolean) _
As Boolean

    toggleThis = Not toggleThis
End Function

Private Sub ChangeMeatloafRecipe(ByVal sender As Object, _
 ByVal e As System.EventArgs) _
 Handles Button3.Click

    Toggle(boringRecipe)
    If boringRecipe Then
        SetMeatloaf(boringMeatloaf, boringMeatloafRanking)

    Else
        Dim greatMeatloafRecipe As String = "1 lb. lean ground beef, " -
            & "1/2 cup bread crumbs, 1/4 cup ketchup," -
            & "1/3 tsp onion powder, " -
            & "1 clove of garlic, 1/2 pack onion soup mix " -
            & "dash of your favorite BBQ Sauce"
        SetMeatloaf(greatMeatloafRecipe, "***")
    End If
End Sub

Private Sub SetMeatloaf(ByVal recipe As String, _
 ByVal rating As String)

    DataGridView1.Rows(0).Cells(2).Value = recipe
    DataGridView1.Rows(0).Cells(3).Value = rating
End Sub

Private Sub ChangeRestaurant(ByVal sender As Object, _
 ByVal e As System.EventArgs) Handles Button10.Click

    If DataGridView1.Rows(2).HeaderCell.Value.Equals(otherRestaurant) Then
        DataGridView1.Rows(2).HeaderCell.Value = _
            "Restaurant 2"
    Else
        DataGridView1.Rows(2).HeaderCell.Value = _
            otherRestaurant
    End If

End Sub

#Region "programmatic resizing"

Private Sub AddButtonsForProgrammaticResizing()
    AddButton(Button4, "Size Third Column")
    AddButton(Button5, "Size Column Headers")
    AddButton(Button6, "Size All Columns")
    AddButton(Button7, "Size Third Row")
    AddButton(Button8, "Size First Row Header Using All Headers")
    AddButton(Button9, "Size All Rows and Row Headers")
    AddButton(Button11, "Size All Rows")
End Sub

' The following code example resizes the second column to fit.
Private Sub SizeThirdColumnHeader(ByVal sender As Object, _
 ByVal e As System.EventArgs) Handles Button4.Click

    DataGridView1.AutoResizeColumn( _
        2, DataGridViewAutoSizeColumnsMode.ColumnHeader)

```

```

End Sub

' The following code example resizes the second column to fit
' the header
' text, but it leaves the widths of the
' row headers and columns unchanged.
Private Sub SizeColumnHeaders(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button5.Click

    DataGridView1.AutoResizeColumnHeadersHeight(2)

End Sub

' The following code example resizes all the columns to fit the
' header text and column contents.
Private Sub SizeAllColumns(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button6.Click

    DataGridView1.AutoResizeColumns(DataGridViewAutoSizeColumnsMode.AllCells)

End Sub

' The following code example resizes the third row
' to fit the column contents.
Private Sub SizeThirdRow(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button7.Click

    Dim thirdRow As Integer = 2
    DataGridView1.AutoResizeRow( _
        2, DataGridViewAutoSizeRowMode.AllCellsExceptHeader)

End Sub

' The following code example resizes the first displayed row
' to fit its header.
Private Sub SizeFirstRowHeaderToAllHeaders(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button8.Click

    DataGridView1.AutoResizeRowHeadersWidth( _
        DataGridViewRowHeadersWidthSizeMode.AutoSizeToAllHeaders)

End Sub

' Size all the rows, including their headers and columns.
Private Sub SizeAllRowsAndTheirHeaders(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button9.Click

    DataGridView1.AutoResizeRows(DataGridViewAutoSizeColumnsMode.AllCells)

End Sub

Private Sub SizeAllRows(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button11.Click

    DataGridView1.AutoResizeRows(DataGridViewAutoSizeColumnsMode.AllCellsExceptHeaders)

End Sub
#End Region
End Class

```

Compiling the Code

This example requires:

- References to the System, System.Drawing, and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[DataGridView](#)

[DataGridView.AutoResizeColumn](#)

[DataGridView.AutoResizeColumns](#)

[DataGridView.AutoResizeColumnHeadersHeight](#)

[DataGridView.AutoResizeRow](#)

[DataGridView.AutoResizeRows](#)

[DataGridView.AutoResizeRowHeadersWidth](#)

[DataGridViewAutoSizeRowMode](#)

[DataGridViewAutoSizeRowsMode](#)

[DataGridViewAutoSizeColumnsMode](#)

[DataGridViewAutoSizeColumnsMode](#)

[DataGridViewColumnHeadersHeightSizeMode](#)

[DataGridViewRowHeadersWidthSizeMode](#)

[Resizing Columns and Rows in the Windows Forms DataGridView Control](#)

[Sizing Options in the Windows Forms DataGridView Control](#)

[How to: Automatically Resize Cells When Content Changes in the Windows Forms DataGridView Control](#)

How to: Automatically Resize Cells When Content Changes in the Windows Forms DataGridView Control

5/4/2018 • 14 min to read • [Edit Online](#)

You can configure the [DataGridView](#) control to resize its rows, columns, and headers automatically whenever content changes, so that cells are always large enough to display their values without clipping.

You have many options to restrict which cells are used to determine the new sizes. For example, you can configure the control to automatically resize the width of its columns based only on the values in rows that are currently displayed. With this, you can avoid inefficiency when working with large numbers of rows, although in this case, you might want to use sizing methods such as [AutoResizeColumns](#) to adjust sizes at times of your choosing.

For more information about automatic resizing, see [Sizing Options in the Windows Forms DataGridView Control](#).

The following code example demonstrates the options available for automatic resizing.

Example

```
#using <System.dll>
#using <System.Drawing.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Drawing;
using namespace System::Collections;
using namespace System::ComponentModel;
using namespace System::Windows::Forms;
public ref class AutoSizing: public System::Windows::Forms::Form
{
private:
    FlowLayoutPanel^ flowLayoutPanel1;
    Button^ button1;
    Button^ button2;
    Button^ button3;
    Button^ button4;
    Button^ button5;
    Button^ button6;
    Button^ button7;
    Button^ button8;
    Button^ button9;
    Button^ button10;
    Button^ button11;

public:
    AutoSizing()
    {
        button1 = gcnew Button;
        button2 = gcnew Button;
        button3 = gcnew Button;
        button4 = gcnew Button;
        button5 = gcnew Button;
        button6 = gcnew Button;
        button7 = gcnew Button;
        button8 = gcnew Button;
        button9 = gcnew Button;
        button10 = gcnew Button;
```

```

button11 = gcnew Button;
thirdColumnHeader = "Main Ingredients";
boringMeatloaf = "ground beef";
boringMeatloafRanking = "*";
otherRestaurant = "Gomes's Saharan Sushi";
currentLayoutName = "DataGridView.AutoSizeRowsMode is currently: ";
InitializeComponent();
this->Load += gcnew EventHandler( this, &AutoSizing::InitializeDataGridView );
AddDirections();
AddButton( button1, "Reset", gcnew EventHandler( this, &AutoSizing::ResetToDisorder ) );
AddButton( button2, "Change Column 3 Header", gcnew EventHandler( this, &AutoSizing::ChangeColumn3Header ) );
AddButton( button3, "Change Meatloaf Recipe", gcnew EventHandler( this,
&AutoSizing::ChangeMeatloafRecipe ) );
AddButton( button4, "Change Restaurant 2", gcnew EventHandler( this, &AutoSizing::ChangeRestaurant ) );
AddButtonsForAutomaticResizing();
}

private:
void AddDirections()
{
    Label^ directions = gcnew Label;
    directions->AutoSize = true;
    String^ newLine = Environment::.NewLine;
    directions->Text = String::Format( "Press the buttons that start {0}with 'Change' to see how different
sizing {1}modes deal with content changes.", newLine, newLine );
    flowLayoutPanel1->Controls->Add( directions );
}

void InitializeComponent()
{
    flowLayoutPanel1 = gcnew FlowLayoutPanel;
    flowLayoutPanel1->FlowDirection = FlowDirection::TopDown;
    flowLayoutPanel1->Location = System::Drawing::Point( 492, 0 );
    flowLayoutPanel1->AutoSize = true;
    flowLayoutPanel1->TabIndex = 1;
    ClientSize = System::Drawing::Size( 674, 419 );
    Controls->Add( flowLayoutPanel1 );
    Text = this->GetType()->Name;
    AutoSize = true;
}

System::Drawing::Size startingSize;
String^ thirdColumnHeader;
String^ boringMeatloaf;
String^ boringMeatloafRanking;
bool boringRecipe;
bool shortMode;
DataGridView^ dataGridView1;
String^ otherRestaurant;
void InitializeDataGridView( Object^ /*ignored*/, EventArgs^ /*ignoredToo*/ )
{
    dataGridView1 = gcnew System::Windows::Forms::DataGridView;
    Controls->Add( dataGridView1 );
    startingSize = System::Drawing::Size( 450, 400 );
    dataGridView1->Size = startingSize;
    dataGridView1->AutoSizeRowsModeChanged += gcnew DataGridViewAutoSizeModeEventHandler( this,
&AutoSizing::WatchRowsModeChanges );
    AddLabels();
    SetUpColumns();
    PopulateRows();
    shortMode = false;
    boringRecipe = true;
}

void SetUpColumns()
{
    dataGridView1->ColumnCount = 4;
}

```

```

        dataGridView1->ColumnHeadersVisible = true;
        DataGridViewCellStyle ^ columnHeaderStyle = gcnew DataGridViewCellStyle();
        columnHeaderStyle->BackColor = Color::Aqua;
        columnHeaderStyle->Font = gcnew System::Drawing::Font( "Verdana",10,FontStyle::Bold );
        dataGridView1->ColumnHeadersDefaultCellStyle = columnHeaderStyle;
        dataGridView1->Columns[ 0 ]->Name = "Recipe";
        dataGridView1->Columns[ 1 ]->Name = "Category";
        dataGridView1->Columns[ 2 ]->Name = thirdColumnHeader;
        dataGridView1->Columns[ 3 ]->Name = "Rating";
    }

void PopulateRows()
{
    array<String^>^row1 = {"Meatloaf","Main Dish",boringMeatloaf,boringMeatloafRanking};
    array<String^>^row2 = {"Key Lime Pie","Dessert","lime juice, evaporated milk","****"};
    array<String^>^row3 = {"Orange-Salsa Pork Chops","Main Dish","pork chops, salsa, orange juice","****"};
    array<String^>^row4 = {"Black Bean and Rice Salad","Salad","black beans, brown rice","****"};
    array<String^>^row5 = {"Chocolate Cheesecake","Dessert","cream cheese","****"};
    array<String^>^row6 = {"Black Bean Dip","Appetizer","black beans, sour cream","***"};
    array<Object^>^rows = {row1,row2,row3,row4,row5,row6};
    IEnumerator^ myEnum = rows->GetEnumerator();
    while ( myEnum->MoveNext() )
    {
        array<String^>^row = safe_cast<array<String^>>(myEnum->Current);
        dataGridView1->Rows->Add( row );
    }

    IEnumerator^ myEnum1 = safe_cast<IEnumerable^>(dataGridView1->Rows)->GetEnumerator();
    while ( myEnum1->MoveNext() )
    {
        DataGridViewRow ^ row = safe_cast<DataGridViewRow ^>(myEnum1->Current);
        if ( row->IsNewRow )
            break;
        row->HeaderCell->Value = String::Format( "Restaurant {0}", row->Index );
    }
}

void AddButton( Button^ button, String^ buttonLabel, EventHandler^ handler )
{
    button->Click += handler;
    button->Text = buttonLabel;
    button->AutoSize = true;
    button->TabIndex = flowLayoutPanel1->Controls->Count;
    flowLayoutPanel1->Controls->Add( button );
}

void ResetToDisorder( Object^ /*sender*/, EventArgs^ /*e*/ )
{
    Controls->Remove( dataGridView1 );
    dataGridView1->DataGridview::~DataGridview();
    InitializeDataGridView( nullptr, nullptr );
}

void ChangeColumn3Header( Object^ /*sender*/, EventArgs^ /*e*/ )
{
    Toggle( &shortMode );
    if ( shortMode )
        dataGridView1->Columns[ 2 ]->HeaderText = "S";
    else
        dataGridView1->Columns[ 2 ]->HeaderText = thirdColumnHeader;
}

Boolean Toggle( interior_ptr<Boolean> toggleThis )
{
    *toggleThis = ! *toggleThis;
    return *toggleThis;
}

void ChangeMeatloafRecipe( Object^ /*sender*/, EventArgs^ /*e*/ )

```

```

{
    Toggle( &boringRecipe );
    if ( boringRecipe )
        SetMeatloaf( boringMeatloaf, boringMeatloafRanking );
    else
    {
        String^ greatMeatloafRecipe = "1 lb. lean ground beef, "
        "1/2 cup bread crumbs, 1/4 cup ketchup,"
        "1/3 tsp onion powder, "
        "1 clove of garlic, 1/2 pack onion soup mix,"
        " dash of your favorite BBQ Sauce";
        SetMeatloaf( greatMeatloafRecipe, "****" );
    }
}

void ChangeRestaurant( Object^ /*sender*/, EventArgs^ /*ignored*/ )
{
    if ( dataGridView1->Rows[ 2 ]->HeaderCell->Value->ToString()->Equals( otherRestaurant ) )
        dataGridView1->Rows[ 2 ]->HeaderCell->Value = "Restaurant 2";
    else
        dataGridView1->Rows[ 2 ]->HeaderCell->Value = otherRestaurant;
}

void SetMeatloaf( String^ recipe, String^ rating )
{
    dataGridView1->Rows[ 0 ]->Cells[ 2 ]->Value = recipe;
    dataGridView1->Rows[ 0 ]->Cells[ 3 ]->Value = rating;
}

String^ currentLayoutName;
void AddLabels()
{
    Label^ current = dynamic_cast<Label^>(flowLayoutPanel1->Controls[ currentLayoutName ]);
    if ( current == nullptr )
    {
        current = gcnew Label;
        current->AutoSize = true;
        current->Name = currentLayoutName;
        current->Text = String::Concat( currentLayoutName, dataGridView1->AutoSizeRowsMode );
        flowLayoutPanel1->Controls->Add( current );
    }
}

void AddButtonsForAutomaticResizing()
{
    AddButton( button5, "Keep Column Headers Sized", gcnew EventHandler( this,
&AutoSizing::ColumnHeadersHeightSizeMode ) );
    AddButton( button6, "Keep Row Headers Sized", gcnew EventHandler( this,
&AutoSizing::RowHeadersWidthSizeMode ) );
    AddButton( button7, "Keep Rows Sized", gcnew EventHandler( this, &AutoSizing::AutoSizeRowsMode ) );
    AddButton( button8, "Keep Row Headers Sized with RowsMode", gcnew EventHandler( this,
&AutoSizing::AutoSizeRowHeadersUsingAllHeadersMode ) );
    AddButton( button9, "Disable AutoSizeRowsMode", gcnew EventHandler( this,
&AutoSizing::DisableAutoSizeRowsMode ) );
    AddButton( button10, "AutoSize third column by rows", gcnew EventHandler( this,
&AutoSizing::AutoSizeOneColumn ) );
    AddButton( button11, "AutoSize third column by rows and headers", gcnew EventHandler( this,
&AutoSizing::AutoSizeOneColumnIncludingHeaders ) );
}

void ColumnHeadersHeightSizeMode( Object^ /*sender*/, EventArgs^ /*e*/ )
{
    dataGridView1->ColumnHeadersHeightSizeMode = DataGridViewColumnHeadersHeightSizeMode::AutoSize;
}

void RowHeadersWidthSizeMode( Object^ /*sender*/, EventArgs^ /*e*/ )
{
    dataGridView1->RowHeadersWidthSizeMode = DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders;
}

```

```

}

void AutoSizeRowsMode( Object^ /*sender*/, EventArgs^ /*es*/ )
{
    dataGridView1->AutoSizeRowsMode = DataGridViewAutoSizeColumnsMode::AllCells;
}
void AutoSizeRowHeadersUsingAllHeadersMode( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    dataGridView1->AutoSizeRowsMode = DataGridViewAutoSizeColumnsMode::AllHeaders;
}

void WatchRowsModeChanges( Object^ /*sender*/, DataGridViewAutoSizeModeEventArgs^ modeEvent )
{
    Label^ label = dynamic_cast<Label^>(flowLayoutPanel1->Controls[ currentLayoutName ]);
    if ( modeEvent->PreviousModeAutoSized )
    {
        label->Text = String::Format( "changed to a different {0}{1}", label->Name, dataGridView1-
>AutoSizeRowsMode );
    }
    else
    {
        label->Text = String::Concat( label->Name, dataGridView1->AutoSizeRowsMode );
    }
}

void DisableAutoSizeRowsMode( Object^ /*sender*/, EventArgs^ /*modeEvent*/ )
{
    dataGridView1->AutoSizeRowsMode = DataGridViewAutoSizeColumnsMode::None;
}

void AutoSizeOneColumn( Object^ /*sender*/, EventArgs^ /*theEvent*/ )
{
    DataGridViewColumn^ column = dataGridView1->Columns[ 2 ];
    column->AutoSizeMode = DataGridViewAutoSizeColumnMode::DisplayedCellsExceptHeader;
}

void AutoSizeOneColumnIncludingHeaders( Object^ /*sender*/, EventArgs^ /*theEvent*/ )
{
    DataGridViewColumn^ column = dataGridView1->Columns[ 2 ];
    column->AutoSizeMode = DataGridViewAutoSizeColumnMode::AllCells;
}

};

[STAThread]
int main()
{
    Application::EnableVisualStyles();
    Application::Run( gcnew AutoSizing );
}

```

```

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;

public class AutoSizing : System.Windows.Forms.Form
{
    private FlowLayoutPanel flowLayoutPanel1;
    private Button button1 = new Button();
    private Button button2 = new Button();

```

```

private Button button1 = new Button();
private Button button2 = new Button();
private Button button3 = new Button();
private Button button4 = new Button();
private Button button5 = new Button();
private Button button6 = new Button();
private Button button7 = new Button();
private Button button8 = new Button();
private Button button9 = new Button();
private Button button10 = new Button();
private Button button11 = new Button();
private DataGridView dataGridView1;

public AutoSizing()
{
    InitializeComponent();
    this.Load += new EventHandler(InitializeDataGridView);

    AddDirections();
    AddButton(button1, "Reset",
        new EventHandler(ResetToDisorder));
    AddButton(button2, "Change Column 3 Header",
        new EventHandler(ChangeColumn3Header));
    AddButton(button3, "Change Meatloaf Recipe",
        new EventHandler(ChangeMeatloafRecipe));
    AddButton(button4, "Change Restaurant 2",
        new EventHandler(ChangeRestaurant));
    AddButtonsForAutomaticResizing();
}

private void AddDirections()
{
    Label directions = new Label();
    directions.AutoSize = true;
    String.newLine = Environment.NewLine;
    directions.Text = "Press the buttons that start " + newLine
        + "with 'Change' to see how different sizing " + newLine
        + "modes deal with content changes.";

    flowLayoutPanel1.Controls.Add(directions);
}

private void InitializeComponent()
{
    flowLayoutPanel1 = new FlowLayoutPanel();
    flowLayoutPanel1.FlowDirection = FlowDirection.TopDown;
    flowLayoutPanel1.Location = new System.Drawing.Point(492, 0);
    flowLayoutPanel1.AutoSize = true;
    flowLayoutPanel1.TabIndex = 1;

    ClientSize = new System.Drawing.Size(674, 419);
    Controls.Add(flowLayoutPanel1);
    Text = this.GetType().Name;
    AutoSize = true;
}

[STAThreadAttribute()]
static void Main()
{
    Application.EnableVisualStyles();
    Application.Run(new AutoSizing());
}

private Size startingSize;
private string thirdColumnHeader = "Main Ingredients";
private string boringMeatloaf = "ground beef";
private string boringMeatloafRanking = "*";
private bool boringRecipe;
private bool shortMode;
private string otherRestaurant = "Gomes's Saharan Sushi";

```

```

private void InitializeDataGridView(Object ignored,
    EventArgs ignoredToo)
{
    dataGridView1 = new System.Windows.Forms.DataGridView();
    Controls.Add(dataGridView1);
    startingSize = new Size(450, 400);
    dataGridView1.Size = startingSize;
    dataGridView1.AutoSizeRowsModeChanged +=
        new DataGridViewAutoSizeModeEventHandler
            (WatchRowsModeChanges);
    AddLabels();

    SetUpColumns();
    PopulateRows();

    shortMode = false;
    boringRecipe = true;
}

private void SetUpColumns()
{
    dataGridView1.ColumnCount = 4;
    dataGridView1.ColumnHeadersVisible = true;

    DataGridViewCellStyle columnHeaderStyle =
        new DataGridViewCellStyle();

    columnHeaderStyle.BackColor = Color.Aqua;
    columnHeaderStyle.Font = new Font("Verdana", 10,
        FontStyle.Bold);
    dataGridView1.ColumnHeadersDefaultCellStyle =
        columnHeaderStyle;

    dataGridView1.Columns[0].Name = "Recipe";
    dataGridView1.Columns[1].Name = "Category";
    dataGridView1.Columns[2].Name = thirdColumnHeader;
    dataGridView1.Columns[3].Name = "Rating";
}

private void PopulateRows()
{
    string[] row1 = {
        "Meatloaf", "Main Dish", boringMeatloaf, boringMeatloafRanking
    };
    string[] row2 = {
        "Key Lime Pie", "Dessert", "lime juice, evaporated milk", "****"
    };
    string[] row3 = {
        "Orange-Salsa Pork Chops", "Main Dish",
        "pork chops, salsa, orange juice", "****"
    };
    string[] row4 = {
        "Black Bean and Rice Salad", "Salad",
        "black beans, brown rice", "****"
    };
    string[] row5 = {
        "Chocolate Cheesecake", "Dessert", "cream cheese", "***"
    };
    string[] row6 = {
        "Black Bean Dip", "Appetizer", "black beans, sour cream", "***"
    };
    object[] rows = new object[] {
        row1, row2, row3, row4, row5, row6
    };

    foreach (string[] row in rows)
        dataGridView1.Rows.Add(row);
}

```

```

        foreach (DataGridViewRow row in dataGridView1.Rows)
    {
        if (row.IsNewRow) break;
        row.HeaderCell.Value = "Restaurant " + row.Index;
    }
}

private void AddButton(Button button, string buttonLabel,
    EventHandler handler)
{
    button.Click += handler;
    button.Text = buttonLabel;
    button.AutoSize = true;
    button.TabIndex = flowLayoutPanel1.Controls.Count;
    flowLayoutPanel1.Controls.Add(button);
}

private void ResetToDisorder(Object sender, EventArgs e)
{
    Controls.Remove(dataGridView1);
    dataGridView1.Dispose();
    InitializeDataGridView(null, null);
}

private void ChangeColumn3Header(Object sender, EventArgs e)
{
    Toggle(ref shortMode);
    if (shortMode) dataGridView1.Columns[2].HeaderText = "S";
    else
        dataGridView1.Columns[2].HeaderText = thirdColumnHeader;
}

private static Boolean Toggle(ref Boolean toggleThis)
{
    toggleThis = !toggleThis;
    return toggleThis;
}

private void ChangeMeatloafRecipe(Object sender, EventArgs e)
{
    Toggle(ref boringRecipe);
    if (boringRecipe)
        SetMeatloaf(boringMeatloaf, boringMeatloafRanking);
    else
    {
        string greatMeatloafRecipe = "1 lb. lean ground beef, "
            + "1/2 cup bread crumbs, 1/4 cup ketchup,"
            + "1/3 tsp onion powder,"
            + "1 clove of garlic, 1/2 pack onion soup mix,"
            + " dash of your favorite BBQ Sauce";

        SetMeatloaf(greatMeatloafRecipe, "***");
    }
}

private void ChangeRestaurant(Object sender, EventArgs ignored)
{
    if (dataGridView1.Rows[2].HeaderCell.Value.ToString() ==
        otherRestaurant)
        dataGridView1.Rows[2].HeaderCell.Value =
            "Restaurant 2";
    else
        dataGridView1.Rows[2].HeaderCell.Value =
            otherRestaurant;
}

private void SetMeatloaf(string recipe, string rating)
{
    dataGridView1.Rows[0].Cells[2].Value = recipe;
}

```

```

        dataGridView1.Rows[0].Cells[3].Value = rating;
    }

    private string currentLayoutName =
        "DataGridView.AutoSizeRowsMode is currently: ";
    private void AddLabels()
    {
        Label current = (Label)
            flowLayoutPanel1.Controls[currentLayoutName];
        if (current == null)
        {
            current = new Label();
            current.AutoSize = true;
            current.Name = currentLayoutName;
            current.Text = currentLayoutName +
                dataGridView1.AutoSizeRowsMode.ToString();
            flowLayoutPanel1.Controls.Add(current);
        }
    }

    #region "Automatic Resizing"
    private void AddButtonsForAutomaticResizing()
    {
        AddButton(button5, "Keep Column Headers Sized",
            new EventHandler(ColumnHeadersSizeMode));
        AddButton(button6, "Keep Row Headers Sized",
            new EventHandler.RowHeadersWidthSizeMode);
        AddButton(button7, "Keep Rows Sized",
            new EventHandler(AutoSizeRowsMode));
        AddButton(button8, "Keep Row Headers Sized with RowsMode",
            new EventHandler(AutoSizeRowHeadersUsingAllHeadersMode));
        AddButton(button9, "Disable AutoSizeRowsMode",
            new EventHandler(DisableAutoSizeRowsMode));
        AddButton(button10, "AutoSize third column by rows",
            new EventHandler(AutoSizeOneColumn));
        AddButton(button11, "AutoSize third column by rows and headers",
            new EventHandler(AutoSizeOneColumnIncludingHeaders));
    }

    private void ColumnHeadersSizeMode(Object sender, EventArgs e)
    {
        dataGridView1.ColumnHeadersHeightSizeMode =
            DataGridViewColumnHeadersHeightSizeMode.AutoSize;
    }

    private void RowHeadersWidthSizeMode(Object sender, EventArgs e)
    {
        dataGridView1.RowHeadersWidthSizeMode =
            DataGridViewRowHeadersWidthSizeMode.AutoSizeToAllHeaders;
    }

    private void AutoSizeRowsMode(Object sender, EventArgs es)
    {
        dataGridView1.AutoSizeRowsMode =
            DataGridViewAutoSizeRowsMode.AllCells;
    }

    private void AutoSizeRowHeadersUsingAllHeadersMode(
        Object sender, System.EventArgs e)
    {
        dataGridView1.AutoSizeRowsMode =
            DataGridViewAutoSizeRowsMode.AllHeaders;
    }

    private void WatchRowsModeChanges(object sender,
        DataGridViewAutoSizeModeEventArgs modeEvent)
    {
        Label label =
            (Label)flowLayoutPanel1.Controls[currentLayoutName];

```

```

        if (modeEvent.PreviousModeAutoSize)
    {
        label.Text = "changed to a different " +
            label.Name +
            dataGridView1.AutoSizeRowsMode.ToString();
    }
    else
    {
        label.Text = label.Name +
            dataGridView1.AutoSizeRowsMode.ToString();
    }
}

private void DisableAutoSizeRowsMode(object sender,
    EventArgs modeEvent)
{
    dataGridView1.AutoSizeRowsMode = DataGridViewAutoSizeRowsMode.None;
}

private void AutoSizeOneColumn(object sender,
    EventArgs theEvent)
{
    DataGridViewColumn column = dataGridView1.Columns[2];
    column.AutoSizeMode =
        DataGridViewAutoSizeColumnMode.DisplayedCellsExceptHeader;
}

private void AutoSizeOneColumnIncludingHeaders(
    object sender, EventArgs theEvent)
{
    DataGridViewColumn column = dataGridView1.Columns[2];
    column.AutoSizeMode = DataGridViewAutoSizeColumnMode.AllCells;
}
#endregion
}

```

```

Public Class AutoSizing
    Inherits System.Windows.Forms.Form

    Friend WithEvents FlowLayoutPanel1 As FlowLayoutPanel
    Friend WithEvents Button1 As Button = New Button()
    Friend WithEvents Button2 As Button = New Button()
    Friend WithEvents Button3 As Button = New Button()
    Friend WithEvents Button4 As Button = New Button()
    Friend WithEvents Button5 As Button = New Button()
    Friend WithEvents Button6 As Button = New Button()
    Friend WithEvents Button7 As Button = New Button()
    Friend WithEvents Button8 As Button = New Button()
    Friend WithEvents Button9 As Button = New Button()
    Friend WithEvents Button10 As Button = New Button()
    Friend WithEvents Button11 As Button = New Button()
    Friend WithEvents DataGridView1 As DataGridView

    Public Sub New()
        MyBase.New()
        InitializeComponent()
        AddDirections()

        AddButton(Button1, "Reset")
        AddButton(Button2, "Change Column 3 Header")
        AddButton(Button3, "Change Meatloaf Recipe")
        AddButton(Button4, "Change Restaurant 2")

        AddButtonsForAutomaticResizing()
    End Sub

```

```

Private Sub AddDirections()
    Dim directions As New Label()
    directions.AutoSize = True
    Dim newLine As String = Environment.NewLine
    directions.Text = "Press the buttons that start " & newLine _
        & "with 'Change' to see how different sizing " & newLine _
        & "modes deal with content changes."
    FlowLayoutPanel1.Controls.Add(directions)
End Sub

Private Sub InitializeComponent()
    Me.FlowLayoutPanel1 = New FlowLayoutPanel
    Me.FlowLayoutPanel1.FlowDirection = FlowDirection.TopDown
    Me.FlowLayoutPanel1.Location = _
        New System.Drawing.Point(454, 0)
    Me.FlowLayoutPanel1.AutoSize = True
    Me.FlowLayoutPanel1.TabIndex = 7

    Me.Controls.Add(Me.FlowLayoutPanel1)
    Me.Text = Me.GetType().Name
    Me.AutoSize = True
End Sub

Private startingSize As Size
Private thirdColumnHeader As String = "Main Ingredients"
Private boringMeatloaf As String = "ground beef"
Private boringMeatloafRanking As String = "*"
Private boringRecipe As Boolean
Private shortMode As Boolean
Private otherRestaurant As String = "Gomes's Saharan Sushi"

Private Sub InitializeDataGridView(ByVal ignored As Object, _
ByVal ignoredToo As EventArgs) Handles Me.Load
    DataGridView1 = New System.Windows.Forms.DataGridView
    Controls.Add(DataGridView1)
    startingSize = New Size(450, 400)
    DataGridView1.Size = startingSize

    SetUpColumns()
    PopulateRows()

    shortMode = False
    boringRecipe = True
    AddLabels()
End Sub

Private Sub SetUpColumns()
    DataGridView1.ColumnCount = 4
    DataGridView1.ColumnHeadersVisible = True

    Dim columnHeaderStyle As New DataGridViewCellStyle
    columnHeaderStyle.BackColor = Color.Aqua
    columnHeaderStyle.Font = New Font("Verdana", 10, _
        FontStyle.Bold)
    DataGridView1.ColumnHeadersDefaultCellStyle = _
        columnHeaderStyle

    DataGridView1.Columns(0).Name = "Recipe"
    DataGridView1.Columns(1).Name = "Category"
    DataGridView1.Columns(2).Name = thirdColumnHeader
    DataGridView1.Columns(3).Name = "Rating"
End Sub

Private Sub PopulateRows()
    Dim row1 As String() = New String() _
        {"Meatloaf", "Main Dish", boringMeatloaf, _
        boringMeatloafRanking}
    Dim row2 As String() = New String() _

```

```

        {"Key Lime Pie", "Dessert", _
        "lime juice, evaporated milk", _
        "****"}
Dim row3 As String() = New String() _
    {"Orange-Salsa Pork Chops", "Main Dish", _
    "pork chops, salsa, orange juice", "****"}
Dim row4 As String() = New String() _
    {"Black Bean and Rice Salad", "Salad", _
    "black beans, brown rice", _
    "****"}
Dim row5 As String() = New String() _
    {"Chocolate Cheesecake", "Dessert", "cream cheese", _
    "****"}
Dim row6 As String() = New String() _
    {"Black Bean Dip", "Appetizer", "black beans, sour cream", _
    "****"}
Dim rows As Object() = New Object() {row1, row2, row3, _
    row4, row5, row6}

Dim rowArray As String()
For Each rowArray In rows
    DataGridView1.Rows.Add(rowArray)
Next

For Each row As DataGridViewRow In DataGridView1.Rows
    If row.IsNewRow Then Continue For
    row.HeaderCell.Value = "Restaurant " & row.Index
Next
End Sub

Private Sub AddButton(ByVal button As Button, _
    ByVal buttonLabel As String)

    button.Text = buttonLabel
    button.AutoSize = True
    FlowLayoutPanel1.Controls.Add(button)
End Sub

Private Sub resetToDisorder(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles Button1.Click

    DataGridView1.Size = startingSize
    Controls.Remove(DataGridView1)
    DataGridView1.Dispose()
    InitializeDataGridView(Nothing, Nothing)
End Sub

Private Sub ChangeColumn3Header(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles Button2.Click

    Toggle(shortMode)
    If shortMode Then DataGridView1.Columns(2).HeaderText = "S" _
        Else DataGridView1.Columns(2).HeaderText = _
        thirdColumnHeader
End Sub

Private Shared Function Toggle(ByRef toggleThis As Boolean) _
As Boolean
    toggleThis = Not toggleThis
    Return toggleThis
End Function

Private Sub ChangeMeatloafRecipe(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles Button3.Click

    Toggle(boringRecipe)

```

```

If boringRecipe Then
    SetMeatloaf(boringMeatloaf, boringMeatloafRanking)
Else
    Dim greatMeatloafRecipe As String = "1 lb. lean ground beef, "
        & "1/2 cup bread crumbs, 1/4 cup ketchup, "
        & "1/3 tsp onion powder, "
        & "1 clove of garlic, 1/2 pack onion soup mix, "
        & "dash of your favorite BBQ Sauce"
    SetMeatloaf(greatMeatloafRecipe, "***")
End If
End Sub

Private Sub ChangeRestaurant(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button4.Click

    If Not DataGridView1.Rows(2).HeaderCell.Value =
        .Equals(otherRestaurant) Then

        DataGridView1.Rows(2).HeaderCell.Value = _
            otherRestaurant
    Else
        DataGridView1.Rows(2).HeaderCell.Value = _
            "Restaurant 2"
    End If
End Sub

Private Sub SetMeatloaf(ByVal recipe As String, _
    ByVal rating As String)
    DataGridView1.Rows(0).Cells(2).Value = recipe
    DataGridView1.Rows(0).Cells(3).Value = rating
End Sub

Private currentLayoutName As String = _
    "DataGridView.AutoSizeRowsMode is currently: "
Private Sub AddLabels()
    Dim current As Label = CType( _
        FlowLayoutPanel1.Controls(currentLayoutName), Label)

    If current Is Nothing Then
        current = New Label()
        current.AutoSize = True
        current.Name = currentLayoutName
        FlowLayoutPanel1.Controls.Add(current)
        current.Text = currentLayoutName & _
            DataGridView1.AutoSizeRowsMode.ToString()
    End If
End Sub

#Region "Automatic Resizing"
Private Sub AddButtonsForAutomaticResizing()
    AddButton(Button5, "Keep Column Headers Sized")
    AddButton(Button6, "Keep Row Headers Sized")
    AddButton(Button7, "Keep Rows Sized")
    AddButton(Button8, "Keep Row Headers Sized with RowsMode")
    AddButton(Button9, "Disable AutoSizeRowsMode")
    AddButton(Button10, "AutoSize third column by rows")
    AddButton(Button11, "AutoSize third column by rows and headers")
End Sub

Private Sub ColumnHeadersHeightSizeMode(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button5.Click

    DataGridView1.ColumnHeadersHeightSizeMode = _
        DataGridViewColumnHeadersHeightSizeMode.AutoSize
End Sub

Private Sub RowHeadersWidthSizeMode(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button6.Click

```

```

    DataGridView1.RowHeadersWidthSizeMode = _
        DataGridViewRowHeadersWidthSizeMode.AutoSizeToAllHeaders

End Sub

Private Sub AutoSizeRowsMode(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button7.Click

    DataGridView1.AutoSizeRowsMode = _
        DataGridViewAutoSizeRowsMode.AllCells

End Sub

Private Sub AutoSizeRowHeadersUsingAllHeadersMode _
    (ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Button8.Click

    DataGridView1.AutoSizeRowsMode = _
        DataGridViewAutoSizeRowsMode.AllHeaders

End Sub

Private Sub WatchRowsModeChanges(ByVal sender As Object, _
    ByVal modeEvent As DataGridViewAutoSizeModeEventArgs) _
    Handles DataGridView1.AutoSizeModeChanged

    Dim label As Label = CType(FlowLayoutPanel1.Controls _
        (currentLayoutName), Label)

    If modeEvent.PreviousModeAutoSized Then
        label.Text = "changed to different " & label.Name & _
            DataGridView1.AutoSizeRowsMode.ToString()
    Else
        label.Text = label.Name & _
            DataGridView1.AutoSizeRowsMode.ToString()
    End If
End Sub

Private Sub DisableAutoSizeRowsMode(ByVal sender As Object, _
    ByVal modeEvent As EventArgs) Handles Button9.Click

    DataGridView1.AutoSizeRowsMode = _
        DataGridViewAutoSizeRowsMode.None
End Sub

Private Sub AutoSizeOneColumn(ByVal sender As Object, _
    ByVal theEvent As EventArgs) Handles Button10.Click

    Dim column As DataGridViewColumn = DataGridView1.Columns(2)
    column.AutoSizeMode = _
        DataGridViewAutoSizeColumnMode.DisplayedCells

End Sub

Private Sub AutoSizeOneColumnIncludingHeaders(ByVal sender As Object, _
    ByVal theEvent As EventArgs) Handles Button11.Click

    Dim column As DataGridViewColumn = DataGridView1.Columns(2)
    column.AutoSizeMode = DataGridViewAutoSizeColumnMode.AllCells

End Sub
#End Region

<STAThreadAttribute()> _
Public Shared Sub Main()
    Application.Run(New AutoSizing())
End Sub

```

End Class

Compiling the Code

This example requires:

- References to the System, System.Drawing, and System.Windows.Forms assemblies.
- For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[DataGridView](#)

[DataGridView.ColumnHeadersHeightSizeMode](#)

[DataGridView.RowHeadersWidthSizeMode](#)

[DataGridViewAutoSizeColumnsMode](#)

[DataGridViewAutoSizeRowsMode](#)

[DataGridViewColumn.AutoSizeMode](#)

[DataGridViewColumn.InheritedAutoSizeMode](#)

[DataGridViewAutoSizeRowsMode](#)

[DataGridViewAutoSizeColumnsMode](#)

[DataGridViewColumnHeadersHeightSizeMode](#)

[DataGridViewRowHeadersWidthSizeMode](#)

[Resizing Columns and Rows in the Windows Forms DataGridView Control](#)

[Sizing Options in the Windows Forms DataGridView Control](#)

[How to: Programmatically Resize Cells to Fit Content in the Windows Forms DataGridView Control](#)

Sorting data in the Windows Forms DataGridView control

5/4/2018 • 1 min to read • [Edit Online](#)

By default, users can sort the data in a [DataGridView](#) control by clicking the header of a text box column (or by pressing F3 when a text box cell is focused on .NET Framework 4.7.2 and later versions). You can modify the [SortMode](#) property of specific columns to allow users to sort by other column types when it makes sense to do so. You can also sort the data programmatically by any column, or by multiple columns.

In this section

[Column Sort Modes in the Windows Forms DataGridView Control](#)

Describes the options for sorting data in the control.

[How to: Set the Sort Modes for Columns in the Windows Forms DataGridView Control](#)

Describes how to enable users to sort by columns that are not sortable by default.

[How to: Customize Sorting in the Windows Forms DataGridView Control](#)

Describes how to sort data programmatically and how to customize sorting by using the [DataGridView.SortCompare](#) event or by implementing the [IComparer](#) interface.

Reference

[DataGridView](#)

Provides reference documentation for the [DataGridView](#) control.

[DataGridView.Sort](#)

Provides reference documentation for the [Sort](#) method.

[DataGridViewColumn.SortMode](#)

Provides reference documentation for the [SortMode](#) property.

[DataGridViewColumnSortMode](#)

Provides reference documentation for the [DataGridViewColumnSortMode](#) enumeration.

See also

[DataGridView Control](#)

[Column Types in the Windows Forms DataGridView Control](#)

Column Sort Modes in the Windows Forms DataGridView Control

5/4/2018 • 5 min to read • [Edit Online](#)

[DataGridView](#) columns have three sort modes. The sort mode for each column is specified through the [SortMode](#) property of the column, which can be set to one of the following [DataGridViewColumnSortMode](#) enumeration values.

DATAGRIDVIEWCOLUMNSMODE	VALUE	DESCRIPTION
Automatic		Default for text box columns. Unless column headers are used for selection, clicking the column header automatically sorts the DataGridView by this column and displays a glyph indicating the sort order.
NotSortable		Default for non-text box columns. You can sort this column programmatically; however, it is not intended for sorting, so no space is reserved for the sorting glyph.
Programmatic		You can sort this column programmatically, and space is reserved for the sorting glyph.

You might want to change the sort mode for a column that defaults to [NotSortable](#) if it contains values that can be meaningfully ordered. For example, if you have a database column containing numbers that represent item states, you can display these numbers as corresponding icons by binding an image column to the database column. You can then change the numerical cell values into image display values in a handler for the [DataGridView.CellFormatting](#) event. In this case, setting the [SortMode](#) property to [Automatic](#) will enable your users to sort the column. Automatic sorting will enable your users to group items that have the same state even if the states corresponding to the numbers do not have a natural sequence. Check box columns are another example where automatic sorting is useful for grouping items in the same state.

You can sort a [DataGridView](#) programmatically by the values in any column or in multiple columns, regardless of the [SortMode](#) settings. Programmatic sorting is useful when you want to provide your own user interface (UI) for sorting or when you want to implement custom sorting. Providing your own sorting UI is useful, for example, when you set the [DataGridView](#) selection mode to enable column header selection. In this case, although the column headers cannot be used for sorting, you still want the headers to display the appropriate sorting glyph, so you would set the [SortMode](#) property to [Programmatic](#).

Columns set to programmatic sort mode do not automatically display a sorting glyph. For these columns, you must display the glyph yourself by setting the [DataGridViewColumnHeaderCell.SortGlyphDirection](#) property. This is necessary if you want flexibility in custom sorting. For example, if you sort the [DataGridView](#) by multiple columns, you might want to display multiple sorting glyphs or no sorting glyph.

Although you can programmatically sort a [DataGridView](#) by any column, some columns, such as button columns, might not contain values that can be meaningfully ordered. For these columns, a [SortMode](#) property setting of [NotSortable](#) indicates that it will never be used for sorting, so there is no need to reserve space in the header for the sorting glyph.

When a [DataGridView](#) is sorted, you can determine both the sort column and the sort order by checking the values of the [SortedColumn](#) and [SortOrder](#) properties. These values are not meaningful after a custom sorting operation. For more information about custom sorting, see the Custom Sorting section later in this topic.

When a [DataGridView](#) control containing both bound and unbound columns is sorted, the values in the unbound columns cannot be maintained automatically. To maintain these values, you must implement virtual mode by setting the [VirtualMode](#) property to `true` and handling the [CellValueNeeded](#) and [CellValuePushed](#) events. For more information, see [How to: Implement Virtual Mode in the Windows Forms DataGridView Control](#). Sorting by unbound columns in bound mode is not supported.

Programmatic Sorting

You can sort a [DataGridView](#) programmatically by calling its [Sort](#) method.

The `Sort(DataGridViewColumn, ListSortDirection)` overload of the [Sort](#) method takes a [DataGridViewColumn](#) and a [ListSortDirection](#) enumeration value as parameters. This overload is useful when sorting by columns with values that can be meaningfully ordered, but which you do not want to configure for automatic sorting. When you call this overload and pass in a column with a [SortMode](#) property value of [DataGridViewColumnSortMode.Automatic](#), the [SortedColumn](#) and [SortOrder](#) properties are set automatically and the appropriate sorting glyph appears in the column header.

NOTE

When the [DataGridView](#) control is bound to an external data source by setting the [DataSource](#) property, the `Sort(DataGridViewColumn, ListSortDirection)` method overload does not work for unbound columns. Additionally, when the [VirtualMode](#) property is `true`, you can call this overload only for bound columns. To determine whether a column is data-bound, check the [IsDataBound](#) property value. Sorting unbound columns in bound mode is not supported.

Custom Sorting

You can customize [DataGridView](#) by using the `Sort(IComparer)` overload of the [Sort](#) method or by handling the [SortCompare](#) event.

The `Sort(IComparer)` method overload takes an instance of a class that implements the [IComparer](#) interface as a parameter. This overload is useful when you want to provide custom sorting; for example, when the values in a column do not have a natural sort order or when the natural sort order is inappropriate. In this case, you cannot use automatic sorting, but you might still want your users to sort by clicking the column headers. You can call this overload in a handler for the [ColumnHeaderMouseClick](#) event if you do not use column headers for selection.

NOTE

The `Sort(IComparer)` method overload works only when the [DataGridView](#) control is not bound to an external data source and the [VirtualMode](#) property value is `false`. To customize sorting for columns bound to an external data source, you must use the sorting operations provided by the data source. In virtual mode, you must provide your own sorting operations for unbound columns.

To use the `Sort(IComparer)` method overload, you must create your own class that implements the [IComparer](#) interface. This interface requires your class to implement the [IComparer.Compare](#) method, to which the [DataGridView](#) passes [DataGridViewRow](#) objects as input when the `Sort(IComparer)` method overload is called. With this, you can calculate the correct row ordering based on the values in any column.

The `Sort(IComparer)` method overload does not set the [SortedColumn](#) and [SortOrder](#) properties, so you must always set the [DataGridViewColumnHeaderCell.SortGlyphDirection](#) property to display the sorting glyph.

As an alternative to the `Sort(IComparer)` method overload, you can provide custom sorting by implementing a handler for the [SortCompare](#) event. This event occurs when users click the headers of columns configured for automatic sorting or when you call the `Sort(DataGridViewColumn, ListSortDirection)` overload of the [Sort](#) method.

The event occurs for each pair of rows in the control, enabling you to calculate their correct order.

NOTE

The [SortCompare](#) event does not occur when the [DataSource](#) property is set or when the [VirtualMode](#) property value is `true`.

See Also

[DataGridView](#)

[DataGridView.Sort](#)

[DataGridView.SortedColumn](#)

[DataGridView.SortOrder](#)

[DataGridViewColumn.SortMode](#)

[DataGridViewColumnHeaderCell.SortGlyphDirection](#)

[Sorting Data in the Windows Forms DataGridView Control](#)

[How to: Set the Sort Modes for Columns in the Windows Forms DataGridView Control](#)

[How to: Customize Sorting in the Windows Forms DataGridView Control](#)

How to: Set the Sort Modes for Columns in the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

In the [DataGridView](#) control, text box columns use automatic sorting by default, while other column types are not sorted automatically. Sometimes you will want to override these defaults. For example, you can display images in place of text, numbers, or enumeration cell values. While the images cannot be sorted, the underlying values that they represent can be sorted.

In the [DataGridView](#) control, the [SortMode](#) property value of a column determines its sorting behavior.

The following procedure shows the `Priority` column from [How to: Customize Data Formatting in the Windows Forms DataGridView Control](#). This column is an image column and is not sortable by default. It contains actual cell values that are strings, however, so it can be sorted automatically.

To set the sort mode for a column

- Set the [DataGridViewColumn.SortMode](#) property.

```
this.dataGridView1.Columns["Priority"].SortMode =  
    DataGridViewColumnSortMode.Automatic;
```

```
Me.dataGridView1.Columns("Priority").SortMode = _  
    DataGridViewColumnSortMode.Automatic
```

Compiling the Code

This example requires:

- A [DataGridView](#) control named `dataGridView1` that contains a column named `Priority`.
- References to the [System](#) and [System.Windows.Forms](#) assemblies.

See Also

[DataGridView](#)

[DataGridViewColumn.SortMode](#)

[Sorting Data in the Windows Forms DataGridView Control](#)

[Column Sort Modes in the Windows Forms DataGridView Control](#)

[How to: Customize Sorting in the Windows Forms DataGridView Control](#)

How to: Customize Sorting in the Windows Forms DataGridView Control

5/4/2018 • 10 min to read • [Edit Online](#)

The [DataGridView](#) control provides automatic sorting but, depending on your needs, you might need to customize sort operations. For example, you can use programmatic sorting to create an alternate user interface (UI). Alternatively, you can handle the [SortCompare](#) event or call the `Sort(IComparer)` overload of the [Sort](#) method for greater sorting flexibility, such as sorting multiple columns.

The following code examples demonstrate these three approaches to custom sorting. For more information, see [Column Sort Modes in the Windows Forms DataGridView Control](#).

Programmatic Sorting

The following code example demonstrates a programmatic sort using the [SortOrder](#) and [SortedColumn](#) properties to determine the direction of the sort, and the [SortGlyphDirection](#) property to manually set the sort glyph. The `Sort(DataGridViewColumn, ListSortDirection)` overload of the [Sort](#) method is used to sort data only in a single column.

```
using System;
using System.ComponentModel;
using System.Windows.Forms;

class Form1 : Form
{
    private Button sortButton = new Button();
    private DataGridView dataGridView1 = new DataGridView();

    // Initializes the form.
    // You can replace this code with designer-generated code.
    public Form1()
    {
        dataGridView1.Dock = DockStyle.Fill;
        dataGridView1.AllowUserToAddRows = false;
        dataGridView1.SelectionMode =
            DataGridViewSelectionMode.ColumnHeaderSelect;
        dataGridView1.MultiSelect = false;

        sortButton.Dock = DockStyle.Bottom;
        sortButton.Text = "Sort";

        Controls.Add(dataGridView1);
        Controls.Add(sortButton);
        Text = "DataGridView programmatic sort demo";
    }

    // Establishes the main entry point for the application.
    [STAThreadAttribute()]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.Run(new Form1());
    }

    // Populates the DataGridView.
    // Replace this with your own code to populate the DataGridView.
    public void PopulateDataGridView()
    {
```

```

    // Add columns to the DataGridView.
    dataGridView1.ColumnCount = 2;
    dataGridView1.Columns[0].HeaderText = "Last Name";
    dataGridView1.Columns[1].HeaderText = "City";
    // Put the new columns into programmatic sort mode
    dataGridView1.Columns[0].SortMode =
        DataGridViewColumnSortMode.Programmatic;
    dataGridView1.Columns[1].SortMode =
        DataGridViewColumnSortMode.Programmatic;

    // Populate the DataGridView.
    dataGridView1.Rows.Add(new string[] { "Parker", "Seattle" });
    dataGridView1.Rows.Add(new string[] { "Watson", "Seattle" });
    dataGridView1.Rows.Add(new string[] { "Osborn", "New York" });
    dataGridView1.Rows.Add(new string[] { "Jameson", "New York" });
    dataGridView1.Rows.Add(new string[] { "Brock", "New Jersey" });
}

protected override void OnLoad(EventArgs e)
{
    sortButton.Click += new EventHandler(sortButton_Click);

    PopulateDataGridView();
    base.OnLoad(e);
}

private void sortButton_Click(object sender, System.EventArgs e)
{
    // Check which column is selected, otherwise set NewColumn to null.
    DataGridViewColumn newColumn =
        dataGridView1.Columns.GetColumnCount(
            DataGridViewElementStates.Selected) == 1 ?
        dataGridView1.SelectedColumns[0] : null;

    DataGridViewColumn oldColumn = dataGridView1.SortedColumn;
    ListSortDirection direction;

    // If oldColumn is null, then the DataGridView is not currently sorted.
    if (oldColumn != null)
    {
        // Sort the same column again, reversing the SortOrder.
        if (oldColumn == newColumn &&
            dataGridView1.SortOrder == SortOrder.Ascending)
        {
            direction = ListSortDirection.Descending;
        }
        else
        {
            // Sort a new column and remove the old SortGlyph.
            direction = ListSortDirection.Ascending;
            oldColumn.HeaderCell.SortGlyphDirection = SortOrder.None;
        }
    }
    else
    {
        direction = ListSortDirection.Ascending;
    }

    // If no column has been selected, display an error dialog box.
    if (newColumn == null)
    {
        MessageBox.Show("Select a single column and try again.",
            "Error: Invalid Selection", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }
    else
    {
        dataGridView1.Sort(newColumn, direction);
    }
}

```

```
        newColumn.HeaderCell.SortGlyphDirection =
            direction == ListSortDirection.Ascending ?
                SortOrder.Ascending : SortOrder.Descending;
    }
}
}
```

```
Imports System
Imports System.ComponentModel
Imports System.Windows.Forms

Public Class Form1
    Inherits Form

    Private WithEvents sortButton As New Button()
    Private WithEvents dataGridView1 As New DataGridView()

    ' Initializes the form.
    ' You can replace this code with designer-generated code.
    Public Sub New()
        With dataGridView1
            .Dock = DockStyle.Fill
            .AllowUserToAddRows = False
            .SelectionMode = DataGridViewSelectionMode.ColumnHeaderSelect
            .MultiSelect = False
        End With

        sortButton.Dock = DockStyle.Bottom
        sortButton.Text = "Sort"

        Controls.Add(dataGridView1)
        Controls.Add(sortButton)
        Text = "DataGridView programmatic sort demo"

        PopulateDataGridView()
    End Sub

    ' Establish the main entry point for the application.
<STAThreadAttribute()> _
Public Shared Sub Main()
    Application.Run(New Form1())
End Sub

    ' Populates the DataGridView.
    ' Replace this with your own code to populate the DataGridView.
    Public Sub PopulateDataGridView()

        ' Add columns to the DataGridView.
        dataGridView1.ColumnCount = 2
        dataGridView1.Columns(0).HeaderText = "Last Name"
        dataGridView1.Columns(1).HeaderText = "City"
        ' Put the new columns into programmatic sort mode
        dataGridView1.Columns(0).SortMode = _
            DataGridViewColumnSortMode.Programmatic
        dataGridView1.Columns(1).SortMode = _
            DataGridViewColumnSortMode.Programmatic

        ' Populate the DataGridView.
        dataGridView1.Rows.Add(New String() {"Parker", "Seattle"})
        dataGridView1.Rows.Add(New String() {"Watson", "Seattle"})
        dataGridView1.Rows.Add(New String() {"Osborn", "New York"})
        dataGridView1.Rows.Add(New String() {"Jameson", "New York"})
        dataGridView1.Rows.Add(New String() {"Brock", "New Jersey"})
    End Sub

    Private Sub SortButton_Click(ByVal sender As Object, _
```

```

    ByVal e As EventArgs) Handles sortButton.Click

    ' Check which column is selected, otherwise set NewColumn to Nothing.
    Dim newColumn As DataGridViewColumn
    If dataGridView1.Columns.GetColumnCount(DataGridViewElementStates_
        .Selected) = 1 Then
        newColumn = dataGridView1.SelectedColumns(0)
    Else
        newColumn = Nothing
    End If

    Dim oldColumn As DataGridViewColumn = dataGridView1.SortedColumn
    Dim direction As ListSortDirection

    ' If oldColumn is null, then the DataGridView is not currently sorted.
    If oldColumn IsNot Nothing Then

        ' Sort the same column again, reversing the SortOrder.
        If oldColumn Is newColumn AndAlso dataGridView1.SortOrder = _
            SortOrder.Ascending Then
            direction = ListSortDirection.Descending
        Else

            ' Sort a new column and remove the old SortGlyph.
            direction = ListSortDirection.Ascending
            oldColumn.HeaderCell.SortGlyphDirection = SortOrder.None
        End If
    Else
        direction = ListSortDirection.Ascending
    End If

    ' If no column has been selected, display an error dialog box.
    If newColumn Is Nothing Then
        MessageBox.Show("Select a single column and try again.", _
            "Error: Invalid Selection", MessageBoxButtons.OK, _
            MessageBoxIcon.Error)
    Else
        dataGridView1.Sort(newColumn, direction)
        If direction = ListSortDirection.Ascending Then
            newColumn.HeaderCell.SortGlyphDirection = SortOrder.Ascending
        Else
            newColumn.HeaderCell.SortGlyphDirection = SortOrder.Descending
        End If
    End If

    End Sub

End Class

```

Custom Sorting Using the SortCompare Event

The following code example demonstrates custom sorting using a [SortCompare](#) event handler. The selected [DataGridViewColumn](#) is sorted and, if there are duplicate values in the column, the ID column is used to determine the final order.

```

#region Using directives

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Windows.Forms;

#endregion

```

```
class Form1 : Form
{
    private DataGridView dataGridView1 = new DataGridView();

    // Establish the main entry point for the application.
    [STAThreadAttribute()]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.Run(new Form1());
    }

    public Form1()
    {
        // Initialize the form.
        // This code can be replaced with designer generated code.
        dataGridView1.AllowUserToAddRows = false;
        dataGridView1.Dock = DockStyle.Fill;
        dataGridView1.SortCompare += new DataGridViewSortCompareEventHandler(
            this.dataGridView1_SortCompare);
        Controls.Add(this.dataGridView1);
        this.Text = "DataGridView.SortCompare demo";

        PopulateDataGridView();
    }

    // Replace this with your own population code.
    public void PopulateDataGridView()
    {
        // Add columns to the DataGridView.
        dataGridView1.ColumnCount = 3;

        // Set the properties of the DataGridView columns.
        dataGridView1.Columns[0].Name = "ID";
        dataGridView1.Columns[1].Name = "Name";
        dataGridView1.Columns[2].Name = "City";
        dataGridView1.Columns["ID"].HeaderText = "ID";
        dataGridView1.Columns["Name"].HeaderText = "Name";
        dataGridView1.Columns["City"].HeaderText = "City";

        // Add rows of data to the DataGridView.
        dataGridView1.Rows.Add(new string[] { "1", "Parker", "Seattle" });
        dataGridView1.Rows.Add(new string[] { "2", "Parker", "New York" });
        dataGridView1.Rows.Add(new string[] { "3", "Watson", "Seattle" });
        dataGridView1.Rows.Add(new string[] { "4", "Jameson", "New Jersey" });
        dataGridView1.Rows.Add(new string[] { "5", "Brock", "New York" });
        dataGridView1.Rows.Add(new string[] { "6", "Conner", "Portland" });

        // Autosize the columns.
        dataGridView1.AutoResizeColumns();
    }

    private void dataGridView1_SortCompare(object sender,
        DataGridViewSortCompareEventArgs e)
    {
        // Try to sort based on the cells in the current column.
        e.SortResult = System.String.Compare(
            e.CellValue1.ToString(), e.CellValue2.ToString());

        // If the cells are equal, sort based on the ID column.
        if (e.SortResult == 0 && e.Column.Name != "ID")
        {
            e.SortResult = System.String.Compare(
                dataGridView1.Rows[e.RowIndex1].Cells["ID"].Value.ToString(),
                dataGridView1.Rows[e.RowIndex2].Cells["ID"].Value.ToString());
        }
        e.Handled = true;
    }
}
```

```

Imports System
Imports System.Windows.Forms

Public Class Form1
    Inherits Form

    Private WithEvents DataGridView1 As New DataGridView()

    ' Establish the main entry point for the application.
    <STAThreadAttribute()> _
    Public Shared Sub Main()
        Application.Run(New Form1())
    End Sub

    Public Sub New()
        ' Initialize the form.
        ' This code can be replaced with designer generated code.
        Me.DataGridView1.AllowUserToAddRows = False
        Me.DataGridView1.Dock = DockStyle.Fill
        Me.Controls.Add(Me.DataGridView1)
        Me.Text = "DataGridView.SortCompare demo"

        Me.PopulateDataGridView()
    End Sub

    '<snippet20>
    ' Replace this with your own population code.
    Private Sub PopulateDataGridView()
        With Me.DataGridView1
            ' Add columns to the DataGridView.
            .ColumnCount = 3

            ' Set the properties of the DataGridView columns.
            .Columns(0).Name = "ID"
            .Columns(1).Name = "Name"
            .Columns(2).Name = "City"
            .Columns("ID").HeaderText = "ID"
            .Columns("Name").HeaderText = "Name"
            .Columns("City").HeaderText = "City"
        End With

        ' Add rows of data to the DataGridView.
        With Me.DataGridView1.Rows
            .Add(New String() {"1", "Parker", "Seattle"})
            .Add(New String() {"2", "Parker", "New York"})
            .Add(New String() {"3", "Watson", "Seattle"})
            .Add(New String() {"4", "Jameson", "New Jersey"})
            .Add(New String() {"5", "Brock", "New York"})
            .Add(New String() {"6", "Conner", "Portland"})
        End With

        ' Autosize the columns.
        Me.DataGridView1.AutoResizeColumns()
    End Sub

    Private Sub DataGridView1_SortCompare( _
        ByVal sender As Object, ByVal e As DataGridViewSortCompareEventArgs) _
        Handles DataGridView1.SortCompare

        ' Try to sort based on the contents of the cell in the current column.
        e.SortResult = System.String.Compare(e.CellValue1.ToString(), _
            e.CellValue2.ToString())

        ' If the cells are equal, sort based on the ID column.
        If (e.SortResult = 0) AndAlso Not (e.Column.Name = "ID") Then

```

```

    e.SortResult = System.String.Compare(
        DataGridView1.Rows(e.RowIndex1).Cells("ID").Value.ToString(), _
        DataGridView1.Rows(e.RowIndex2).Cells("ID").Value.ToString())
End If

e.Handled = True

End Sub
End Class

```

Custom Sorting Using the [IComparer](#) Interface

The following code example demonstrates custom sorting using the [Sort\(IComparer\)](#) overload of the [Sort](#) method, which takes an implementation of the [IComparer](#) interface to perform a multiple-column sort.

```

#region Using directives

using System;
using System.Drawing;
using System.Windows.Forms;

#endregion

class Form1 : Form
{
    private DataGridView DataGridView1 = new DataGridView();
    private FlowLayoutPanel FlowLayoutPanel1 = new FlowLayoutPanel();
    private Button Button1 = new Button();
    private RadioButton RadioButton1 = new RadioButton();
    private RadioButton RadioButton2 = new RadioButton();

    // Establish the main entry point for the application.
    [STAThreadAttribute()]
    public static void Main()
    {
        Application.Run(new Form1());
    }

    public Form1()
    {
        // Initialize the form.
        // This code can be replaced with designer generated code.
        AutoSize = true;
        Text = "DataGridView IComparer sort demo";

        FlowLayoutPanel1.FlowDirection = FlowDirection.TopDown;
        FlowLayoutPanel1.Location = new System.Drawing.Point( 304, 0 );
        FlowLayoutPanel1.AutoSize = true;

        FlowLayoutPanel1.Controls.Add( RadioButton1 );
        FlowLayoutPanel1.Controls.Add( RadioButton2 );
        FlowLayoutPanel1.Controls.Add( Button1 );

        Button1.Text = "Sort";
        RadioButton1.Text = "Ascending";
        RadioButton2.Text = "Descending";
        RadioButton1.Checked = true;

        Controls.Add( FlowLayoutPanel1 );
        Controls.Add( DataGridView1 );
    }

    protected override void OnLoad( EventArgs e )
    {
        PopulateDataGridView();
    }
}

```

```

        Button1.Click += new EventHandler(Button1_Click);

        base.OnLoad( e );
    }

// Replace this with your own code to populate the DataGridView.
private void PopulateDataGridView()
{
    DataGridView1.Size = new Size(300, 300);

    // Add columns to the DataGridView.
    DataGridView1.ColumnCount = 2;

    // Set the properties of the DataGridView columns.
    DataGridView1.Columns[0].Name = "First";
    DataGridView1.Columns[1].Name = "Last";
    DataGridView1.Columns["First"].HeaderText = "First Name";
    DataGridView1.Columns["Last"].HeaderText = "Last Name";
    DataGridView1.Columns["First"].SortMode =
        DataGridViewColumnSortMode.Programmatic;
    DataGridView1.Columns["Last"].SortMode =
        DataGridViewColumnSortMode.Programmatic;

    // Add rows of data to the DataGridView.
    DataGridView1.Rows.Add(new string[] { "Peter", "Parker" });
    DataGridView1.Rows.Add(new string[] { "James", "Jameson" });
    DataGridView1.Rows.Add(new string[] { "May", "Parker" });
    DataGridView1.Rows.Add(new string[] { "Mary", "Watson" });
    DataGridView1.Rows.Add(new string[] { "Eddie", "Brock" });
}

private void Button1_Click( object sender, EventArgs e )
{
    if ( RadioButton1.Checked == true )
    {
        DataGridView1.Sort( new RowComparer( SortOrder.Ascending ) );
    }
    else if ( RadioButton2.Checked == true )
    {
        DataGridView1.Sort( new RowComparer( SortOrder.Descending ) );
    }
}

private class RowComparer : System.Collections.IComparer
{
    private static int sortOrderModifier = 1;

    public RowComparer(SortOrder sortOrder)
    {
        if (sortOrder == SortOrder.Descending)
        {
            sortOrderModifier = -1;
        }
        else if (sortOrder == SortOrder.Ascending)
        {
            sortOrderModifier = 1;
        }
    }

    public int Compare(object x, object y)
    {
        DataGridViewRow DataGridViewRow1 = (DataGridViewRow)x;
        DataGridViewRow DataGridViewRow2 = (DataGridViewRow)y;

        // Try to sort based on the Last Name column.
        int CompareResult = System.String.Compare(
            DataGridViewRow1.Cells[1].Value.ToString(),
            DataGridViewRow2.Cells[1].Value.ToString());
    }
}

```

```

        // If the Last Names are equal, sort based on the First Name.
        if ( CompareResult == 0 )
        {
            CompareResult = System.String.Compare(
                DataGridViewRow1.Cells[0].Value.ToString(),
                DataGridViewRow2.Cells[0].Value.ToString());
        }
        return CompareResult * sortOrderModifier;
    }
}
}

```

```

Imports System
Imports System.Drawing
Imports System.Windows.Forms

Public Class Form1
Inherits Form

Private WithEvents DataGridView1 As New DataGridView()
Private FlowLayoutPanel1 As New FlowLayoutPanel()
Private WithEvents Button1 As New Button()
Private RadioButton1 As New RadioButton()
Private RadioButton2 As New RadioButton()

' Establish the main entry point for the application.
<STAThreadAttribute()> _
Public Shared Sub Main()
    Application.Run(New Form1())
End Sub

Public Sub New()
    ' Initialize the form.
    ' This code can be replaced with designer generated code.
    AutoSize = True
    Text = "DataGridView IComparer sort demo"

    FlowLayoutPanel1.FlowDirection = FlowDirection.TopDown
    FlowLayoutPanel1.Location = New System.Drawing.Point(304, 0)
    FlowLayoutPanel1.AutoSize = True

    FlowLayoutPanel1.Controls.Add(RadioButton1)
    FlowLayoutPanel1.Controls.Add(RadioButton2)
    FlowLayoutPanel1.Controls.Add(Button1)

    Button1.Text = "Sort"
    RadioButton1.Text = "Ascending"
    RadioButton2.Text = "Descending"
    RadioButton1.Checked = True

    Controls.Add(FlowLayoutPanel1)
    Controls.Add(DataGridView1)

    PopulateDataGridView()
End Sub

' Replace this with your own code to populate the DataGridView.
Private Sub PopulateDataGridView()

    DataGridView1.Size = New Size(300, 300)

    ' Add columns to the DataGridView.
    DataGridView1.ColumnCount = 2

    ' Set the properties of the DataGridView columns.
    DataGridView1.Columns(0).Name = "First"

```

```

    DataGridView1.Columns(1).Name = "Last"
    DataGridView1.Columns("First").HeaderText = "First Name"
    DataGridView1.Columns("Last").HeaderText = "Last Name"
    DataGridView1.Columns("First").SortMode = _
        DataGridViewColumnSortMode.Programmatic
    DataGridView1.Columns("Last").SortMode = _
        DataGridViewColumnSortMode.Programmatic

    ' Add rows of data to the DataGridView.
    DataGridView1.Rows.Add(New String() {"Peter", "Parker"})
    DataGridView1.Rows.Add(New String() {"James", "Jameson"})
    DataGridView1.Rows.Add(New String() {"May", "Parker"})
    DataGridView1.Rows.Add(New String() {"Mary", "Watson"})
    DataGridView1.Rows.Add(New String() {"Eddie", "Brock"})
End Sub

Private Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs) _
Handles Button1.Click
    If RadioButton1.Checked = True Then
        DataGridView1.Sort(New RowComparer(SortOrder.Ascending))
    ElseIf RadioButton2.Checked = True Then
        DataGridView1.Sort(New RowComparer(SortOrder.Descending))
    End If
End Sub

Private Class RowComparer
    Implements System.Collections.IComparer

    Private sortOrderModifier As Integer = 1

    Public Sub New(ByVal sortOrder As SortOrder)
        If sortOrder = sortOrder.Descending Then
            sortOrderModifier = -1
        ElseIf sortOrder = sortOrder.Ascending Then

            sortOrderModifier = 1
        End If
    End Sub

    Public Function Compare(ByVal x As Object, ByVal y As Object) As Integer _
        Implements System.Collections.IComparer.Compare

        Dim DataGridViewRow1 As DataGridViewRow = CType(x, DataGridViewRow)
        Dim DataGridViewRow2 As DataGridViewRow = CType(y, DataGridViewRow)

        ' Try to sort based on the Last Name column.
        Dim CompareResult As Integer = System.String.Compare( _
            DataGridView1.Cells(1).Value.ToString(), _
            DataGridView2.Cells(1).Value.ToString())

        ' If the Last Names are equal, sort based on the First Name.
        If CompareResult = 0 Then
            CompareResult = System.String.Compare( _
                DataGridView1.Cells(0).Value.ToString(), _
                DataGridView2.Cells(0).Value.ToString())
        End If
        Return CompareResult * sortOrderModifier
    End Function
End Class
End Class

```

Compiling the Code

These examples require:

- References to the System, System.Drawing, and System.Windows.Forms assemblies.

For information about building these examples from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[DataGridView](#)

[Sorting Data in the Windows Forms DataGridView Control](#)

[Column Sort Modes in the Windows Forms DataGridView Control](#)

[How to: Set the Sort Modes for Columns in the Windows Forms DataGridView Control](#)

Data Entry in the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

The `DataGridView` control provides several features that let you change how users add or modify data in the control. For example, you can make data entry more efficient by providing default values for new rows and by alerting users when errors occur.

In This Section

[How to: Specify the Edit Mode for the Windows Forms DataGridView Control](#)

Describes how to change the way users start editing cells.

[How to: Specify Default Values for New Rows in the Windows Forms DataGridView Control](#)

Describes how to prepopulate the row for new records to save data-entry time.

[Using the Row for New Records in the Windows Forms DataGridView Control](#)

Describes the row for new records in detail, including information on hiding it, on customizing its appearance, and on how it relates to the `Rows` collection.

[Walkthrough: Validating Data in the Windows Forms DataGridView Control](#)

Describes how to validate user input to prevent data-entry formatting errors.

[Walkthrough: Handling Errors that Occur During Data Entry in the Windows Forms DataGridView Control](#)

Describes how to handle data-entry errors that originate from the data source when the user attempts to commit a new value.

Reference

[DataGridView](#)

Provides reference documentation for the `DataGridView` control.

[DataGridViewEditMode](#)

Provides reference documentation for the `EditMode` property.

[DataGridView.DefaultValuesNeeded](#)

Provides reference documentation for the `DefaultValuesNeeded` event.

[DataGridView.DataError](#)

Provides reference documentation for the `DataError` event.

[DataGridView.CellValidating](#)

Provides reference documentation for the `CellValidating` event.

Related Sections

[Displaying Data in the Windows Forms DataGridView Control](#)

Provides topics that describe how to populate the control with data either manually or from an external data source.

See Also

[DataGridView Control](#)

[Column Types in the Windows Forms DataGridView Control](#)

How to: Specify the Edit Mode for the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

By default, users can edit the contents of the current [DataGridView](#) text box cell by typing in it or pressing F2. This puts the cell in edit mode if all of the following conditions are met:

- The underlying data source supports editing.
- The [DataGridView](#) control is enabled.
- The [EditMode](#) property value is not [EditProgrammatically](#).
- The `ReadOnly` properties of the cell, row, column, and control are all set to `false`.

In edit mode, the user can change the cell value and press ENTER to commit the change or ESC to revert the cell to its original value.

You can configure a [DataGridView](#) control so that a cell enters edit mode as soon as it becomes the current cell. The behavior of the ENTER and ESC keys is unchanged in this case, but the cell remains in edit mode after the value is committed or reverted. You can also configure the control so that cells enter edit mode only when users type in the cell or only when users press F2. Finally, you can prevent cells from entering edit mode except when you call the [BeginEdit](#) method.

To change the edit mode of a DataGridView control

- Set the [DataGridView.EditMode](#) property to the appropriate [DataGridViewEditMode](#) enumeration.

```
this.dataGridView1EditMode = DataGridViewEditMode.EditOnEnter;
```

```
Me.dataGridView1EditMode = DataGridViewEditMode.EditOnEnter
```

Compiling the Code

This example requires:

- A [DataGridView](#) control named `dataGridView1`.
- References to the [System](#) and [System.Windows.Forms](#) assemblies.

See Also

[DataGridView](#)

[DataGridViewEditMode](#)

[Data Entry in the Windows Forms DataGridView Control](#)

How to: Specify Default Values for New Rows in the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

You can make data entry more convenient when the application fills in default values for newly added rows. With the [DataGridView](#) class, you can fill in default values with the [DefaultValuesNeeded](#) event. This event is raised when the user enters the row for new records. When your code handles this event, you can populate desired cells with values of your choosing.

The following code example demonstrates how to specify default values for new rows using the [DefaultValuesNeeded](#) event.

Example

```
private void dataGridView1_DefaultValuesNeeded(object sender,
    System.Windows.Forms.DataGridViewRowEventArgs e)
{
    e.Row.Cells["Region"].Value = "WA";
    e.Row.Cells["City"].Value = "Redmond";
    e.Row.Cells["PostalCode"].Value = "98052-6399";
    e.Row.Cells["Country"].Value = "USA";
    e.Row.Cells["CustomerID"].Value = NewCustomerId();
}
```

```
Private Sub dataGridView1_DefaultValuesNeeded(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DataGridViewRowEventArgs) _
    Handles dataGridView1.DefaultValuesNeeded

    With e.Row
        .Cells("Region").Value = "WA"
        .Cells("City").Value = "Redmond"
        .Cells("PostalCode").Value = "98052-6399"
        .Cells("Country").Value = "USA"
        .Cells("CustomerID").Value = NewCustomerId()
    End With

End Sub
```

Compiling the Code

This example requires:

- A [DataGridView](#) control named `dataGridView1`.
- A `NewCustomerId` function for generating unique `CustomerID` values.
- References to the [System](#) and [System.Windows.Forms](#) assemblies.

See Also

[DataGridView](#)

[DataGridView.DefaultValuesNeeded](#)

Data Entry in the Windows Forms DataGridView Control

Using the Row for New Records in the Windows Forms DataGridView Control

Using the Row for New Records in the Windows Forms DataGridView Control

5/4/2018 • 3 min to read • [Edit Online](#)

When you use a [DataGridView](#) for editing data in your application, you will often want to give your users the ability to add new rows of data to the data store. The [DataGridView](#) control supports this functionality by providing a row for new records, which is always shown as the last row. It is marked with an asterisk (*) symbol in its row header. The following sections discuss some of the things you should consider when you program with the row for new records enabled.

Displaying the Row for New Records

Use the [AllowUserToAddRows](#) property to indicate whether the row for new records is displayed. The default value of this property is `true`.

For the data bound case, the row for new records will be shown if the [AllowUserToAddRows](#) property of the control and the [IBindingList.AllowNew](#) property of the data source are both `true`. If either is `false` then the row will not be shown.

Populating the Row for New Records with Default Data

When the user selects the row for new records as the current row, the [DataGridView](#) control raises the [DefaultValuesNeeded](#) event.

This event provides access to the new [DataGridViewRow](#) and enables you to populate the new row with default data. For more information, see [How to: Specify Default Values for New Rows in the Windows Forms DataGridView Control](#)

The Rows Collection

The row for new records is contained in the [DataGridView](#) control's [Rows](#) collection but behaves differently in two respects:

- The row for new records cannot be removed from the [Rows](#) collection programmatically. An [InvalidOperationException](#) is thrown if this is attempted. The user also cannot delete the row for new records. The [DataGridViewRowCollection.Clear](#) method does not remove this row from the [Rows](#) collection.
- No row can be added after the row for new records. An [InvalidOperationException](#) is raised if this is attempted. As a result, the row for new records is always the last row in the [DataGridView](#) control. The methods on [DataGridViewRowCollection](#) that add rows—[Add](#), [AddCopy](#), and [AddCopies](#)—all call insertion methods internally when the row for new records is present.

Visual Customization of the Row for New Records

When the row for new records is created, it is based on the row specified by the [RowTemplate](#) property. Any cell styles that are not specified for this row are inherited from other properties. For more information about cell style inheritance, see [Cell Styles in the Windows Forms DataGridView Control](#).

The initial values displayed by cells in the row for new records are retrieved from each cell's [DefaultNewRowValue](#)

property. For cells of type [DataGridViewImageCell](#), this property returns a placeholder image. Otherwise, this property returns `null`. You can override this property to return a custom value. However, these initial values can be replaced by a [DefaultValuesNeeded](#) event handler when focus enters the row for new records.

The standard icons for this row's header, which are an arrow or an asterisk, are not exposed publicly. If you want to customize the icons, you will need to create a custom [DataGridViewRowHeaderCell](#) class.

The standard icons use the [ForeColor](#) property of the [DataGridViewCellStyle](#) in use by the row header cell. The standard icons are not rendered if there is not enough space to display them completely.

If the row header cell has a string value set, and if there is not enough room for both the text and icon, the icon is dropped first.

Sorting

In unbound mode, new records will always be added to the end of the [DataGridView](#) even if the user has sorted the content of the [DataGridView](#). The user will need to apply the sort again in order to sort the row to the correct position; this behavior is similar to that of the [ListView](#) control.

In data bound and virtual modes, the insertion behavior when a sort is applied will be dependent on the implementation of the data model. For ADO.NET, the row is immediately sorted into the correct position.

Other Notes on the Row for New Records

You cannot set the [Visible](#) property of this row to `false`. An [InvalidOperationException](#) is raised if this is attempted.

The row for new records is always created in the unselected state.

Virtual Mode

If you are implementing virtual mode, you will need to track when a row for new records is needed in the data model and when to roll back the addition of the row. The exact implementation of this functionality depends on the implementation of the data model and its transaction semantics, for example, whether commit scope is at the cell or row level. For more information, see [Virtual Mode in the Windows Forms DataGridView Control](#).

See Also

[DataGridView](#)

[DataGridView.DefaultValuesNeeded](#)

[Data Entry in the Windows Forms DataGridView Control](#)

[How to: Specify Default Values for New Rows in the Windows Forms DataGridView Control](#)

Walkthrough: Validating Data in the Windows Forms DataGridView Control

5/4/2018 • 5 min to read • [Edit Online](#)

When you display data entry functionality to users, you frequently have to validate the data entered into your form. The [DataGridView](#) class provides a convenient way to perform validation before data is committed to the data store. You can validate data by handling the [CellValidating](#) event, which is raised by the [DataGridView](#) when the current cell changes.

In this walkthrough, you will retrieve rows from the `Customers` table in the Northwind sample database and display them in a [DataGridView](#) control. When a user edits a cell in the `CompanyName` column and tries to leave the cell, the [CellValidating](#) event handler will examine new company name string to make sure it is not empty; if the new value is an empty string, the [DataGridView](#) will prevent the user's cursor from leaving the cell until a non-empty string is entered.

To copy the code in this topic as a single listing, see [How to: Validate Data in the Windows Forms DataGridView Control](#).

Prerequisites

In order to complete this walkthrough, you will need:

- Access to a server that has the Northwind SQL Server sample database.

Creating the Form

To validate data entered in a DataGridView

1. Create a class that derives from [Form](#) and contains a [DataGridView](#) control and a [BindingSource](#) component.

The following code example provides basic initialization and includes a `Main` method.

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Windows.Forms;

public class Form1 : System.Windows.Forms.Form
{
    private DataGridView dataGridView1 = new DataGridView();
    private BindingSource bindingSource1 = new BindingSource();

    public Form1()
    {
        // Initialize the form.
        this.dataGridView1.Dock = DockStyle.Fill;
        this.Controls.Add(dataGridView1);
        this.Load += new EventHandler(Form1_Load);
        this.Text = "DataGridView validation demo (disallows empty CompanyName)";
    }
}
```

```

Imports System
Imports System.Data
Imports System.Data.SqlClient
Imports System.Windows.Forms

Public Class Form1
    Inherits System.Windows.Forms.Form

    Private WithEvents dataGridView1 As New DataGridView()
    Private bindingSource1 As New BindingSource()

    Public Sub New()

        ' Initialize the form.
        Me.dataGridView1.Dock = DockStyle.Fill
        Me.Controls.Add(dataGridView1)
        Me.Text = "DataGridView validation demo (disallows empty CompanyName)"

    End Sub

```

```

[STAThread]
static void Main()
{
    Application.EnableVisualStyles();
    Application.Run(new Form1());
}

```

```

<STAThread()> _
Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())
End Sub

End Class

```

2. Implement a method in your form's class definition for handling the details of connecting to the database.

This code example uses a `GetData` method that returns a populated `DataTable` object. Be sure that you set the `connectionString` variable to a value that is appropriate for your database.

IMPORTANT

Storing sensitive information, such as a password, within the connection string can affect the security of your application. Using Windows Authentication, also known as integrated security, is a more secure way to control access to a database. For more information, see [Protecting Connection Information](#).

```

private static DataTable GetData(string selectCommand)
{
    string connectionString =
        "Integrated Security=SSPI;Persist Security Info=False;" +
        "Initial Catalog=Northwind;Data Source=localhost;Packet Size=4096";

    // Connect to the database and fill a data table.
    SqlDataAdapter adapter =
        new SqlDataAdapter(selectCommand, connectionString);
    DataTable data = new DataTable();
    data.Locale = System.Globalization.CultureInfo.InvariantCulture;
    adapter.Fill(data);

    return data;
}

```

```

Private Shared Function GetData(ByVal selectCommand As String) As DataTable

    Dim connectionString As String = _
        "Integrated Security=SSPI;Persist Security Info=False;" + _
        "Initial Catalog=Northwind;Data Source=localhost;Packet Size=4096"

    ' Connect to the database and fill a data table.
    Dim adapter As New SqlDataAdapter(selectCommand, connectionString)
    Dim data As New DataTable()
    data.Locale = System.Globalization.CultureInfo.InvariantCulture
    adapter.Fill(data)

    Return data

End Function

```

3. Implement a handler for your form's [Load](#) event that initializes the [DataGridView](#) and [BindingSource](#) and sets up the data binding.

```

private void Form1_Load(System.Object sender, System.EventArgs e)
{
    // Attach DataGridView events to the corresponding event handlers.
    this.dataGridView1.CellValidating += new
        DataGridViewCellValidatingEventHandler(dataGridView1_CellValidating);
    this.dataGridView1.CellEndEdit += new
        DataGridViewCellEventHandler(dataGridView1_CellEndEdit);

    // Initialize the BindingSource and bind the DataGridView to it.
    bindingSource1.DataSource = GetData("select * from Customers");
    this.dataGridView1.DataSource = bindingSource1;
    this.dataGridView1.AutoResizeColumns(
        DataGridViewAutoSizeColumnsMode.AllCellsExceptHeader);
}

```

```

Private Sub Form1_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Me.Load

    ' Initialize the BindingSource and bind the DataGridView to it.
    bindingSource1.DataSource = GetData("select * from Customers")
    Me.dataGridView1.DataSource = bindingSource1
    Me.dataGridView1.AutoResizeColumns(
        DataGridViewAutoSizeColumnsMode.AllCellsExceptHeader)

End Sub

```

4. Implement handlers for the `DataGridView` control's `CellValidating` and `CellEndEdit` events.

The `CellValidating` event handler is where you determine whether the value of a cell in the `CompanyName` column is empty. If the cell value fails validation, set the `Cancel` property of the `System.Windows.Forms.DataGridViewCellValidatingEventArgs` class to `true`. This causes the `DataGridView` control to prevent the cursor from leaving the cell. Set the `ErrorText` property on the row to an explanatory string. This displays an error icon with a ToolTip that contains the error text. In the `CellEndEdit` event handler, set the `ErrorText` property on the row to the empty string. The `CellEndEdit` event occurs only when the cell exits edit mode, which it cannot do if it fails validation.

```
private void dataGridView1_CellValidating(object sender,
    DataGridViewCellValidatingEventArgs e)
{
    string headerText =
        dataGridView1.Columns[e.ColumnIndex].HeaderText;

    // Abort validation if cell is not in the CompanyName column.
    if (!headerText.Equals("CompanyName")) return;

    // Confirm that the cell is not empty.
    if (string.IsNullOrEmpty(e.FormattedValue.ToString()))
    {
        dataGridView1.Rows[e.RowIndex].ErrorText =
            "Company Name must not be empty";
        e.Cancel = true;
    }
}

void dataGridView1_CellEndEdit(object sender, DataGridViewCellEventArgs e)
{
    // Clear the row error in case the user presses ESC.
    dataGridView1.Rows[e.RowIndex].ErrorText = String.Empty;
}
```

```
Private Sub dataGridView1_CellValidating(ByVal sender As Object, _
    ByVal e As DataGridViewCellValidatingEventArgs) _
Handles dataGridView1.CellValidating

    Dim headerText As String = _
        dataGridView1.Columns(e.ColumnIndex).HeaderText

    ' Abort validation if cell is not in the CompanyName column.
    If Not headerText.Equals("CompanyName") Then Return

    ' Confirm that the cell is not empty.
    If (String.IsNullOrEmpty(e.FormattedValue.ToString())) Then
        dataGridView1.Rows(e.RowIndex).ErrorText = _
            "Company Name must not be empty"
        e.Cancel = True
    End If
End Sub

Private Sub dataGridView1_CellEndEdit(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DataGridViewCellEventArgs) _
Handles dataGridView1.CellEndEdit

    ' Clear the row error in case the user presses ESC.
    dataGridView1.Rows(e.RowIndex).ErrorText = String.Empty

End Sub
```

Testing the Application

You can now test the form to make sure it behaves as expected.

To test the form

- Compile and run the application.

You will see a [DataGridView](#) filled with data from the `Customers` table. When you double-click a cell in the `CompanyName` column, you can edit the value. If you delete all the characters and hit the TAB key to exit the cell, the [DataGridView](#) prevents you from exiting. When you type a non-empty string into the cell, the [DataGridView](#) control lets you exit the cell.

Next Steps

This application gives you a basic understanding of the [DataGridView](#) control's capabilities. You can customize the appearance and behavior of the [DataGridView](#) control in several ways:

- Change border and header styles. For more information, see [How to: Change the Border and Gridline Styles in the Windows Forms DataGridView Control](#).
- Enable or restrict user input to the [DataGridView](#) control. For more information, see [How to: Prevent Row Addition and Deletion in the Windows Forms DataGridView Control](#), and [How to: Make Columns Read-Only in the Windows Forms DataGridView Control](#).
- Check user input for database-related errors. For more information, see [Walkthrough: Handling Errors that Occur During Data Entry in the Windows Forms DataGridView Control](#).
- Handle very large data sets using virtual mode. For more information, see [Walkthrough: Implementing Virtual Mode in the Windows Forms DataGridView Control](#).
- Customize the appearance of cells. For more information, see [How to: Customize the Appearance of Cells in the Windows Forms DataGridView Control](#) and [How to: Set Font and Color Styles in the Windows Forms DataGridView Control](#).

See Also

[DataGridView](#)

[BindingSource](#)

[Data Entry in the Windows Forms DataGridView Control](#)

[How to: Validate Data in the Windows Forms DataGridView Control](#)

[Walkthrough: Handling Errors that Occur During Data Entry in the Windows Forms DataGridView Control](#)

[Protecting Connection Information](#)

How to: Validate Data in the Windows Forms DataGridView Control

5/4/2018 • 3 min to read • [Edit Online](#)

The following code example demonstrates how to validate data entered by a user into a [DataGridView](#) control. In this example, the [DataGridView](#) is populated with rows from the `Customers` table of the Northwind sample database. When the user edits a cell in the `CompanyName` column, its value is tested for validity by checking that it is not empty. If the event handler for the [CellValidating](#) event finds that the value is an empty string, the [DataGridView](#) prevents the user from exiting the cell until a non-empty string is entered.

For a complete explanation of this code example, see [Walkthrough: Validating Data in the Windows Forms DataGridView Control](#).

Example

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Windows.Forms;

public class Form1 : System.Windows.Forms.Form
{
    private DataGridView dataGridView1 = new DataGridView();
    private BindingSource bindingSource1 = new BindingSource();

    public Form1()
    {
        // Initialize the form.
        this.dataGridView1.Dock = DockStyle.Fill;
        this.Controls.Add(dataGridView1);
        this.Load += new EventHandler(Form1_Load);
        this.Text = "DataGridView validation demo (disallows empty CompanyName)";
    }

    private void Form1_Load(System.Object sender, System.EventArgs e)
    {
        // Attach DataGridView events to the corresponding event handlers.
        this.dataGridView1.CellValidating += new
            DataGridViewCellValidatingEventHandler(dataGridView1_CellValidating);
        this.dataGridView1.CellEndEdit += new
            DataGridViewCellEventHandler(dataGridView1_CellEndEdit);

        // Initialize the BindingSource and bind the DataGridView to it.
        bindingSource1.DataSource = GetData("select * from Customers");
        this.dataGridView1.DataSource = bindingSource1;
        this.dataGridView1.AutoResizeColumns(
            DataGridViewAutoSizeColumnsMode.AllCellsExceptHeader);
    }

    private void dataGridView1_CellValidating(object sender,
        DataGridViewCellValidatingEventArgs e)
    {
        string headerText =
            dataGridView1.Columns[e.ColumnIndex].HeaderText;

        // Abort validation if cell is not in the CompanyName column.
        if (!headerText.Equals("CompanyName")) return;
    }
}
```

```

// Confirm that the cell is not empty.
if (string.IsNullOrEmpty(e.FormattedValue.ToString()))
{
    dataGridView1.Rows[e.RowIndex].ErrorText =
        "Company Name must not be empty";
    e.Cancel = true;
}

void dataGridView1_CellEndEdit(object sender, DataGridViewCellEventArgs e)
{
    // Clear the row error in case the user presses ESC.
    dataGridView1.Rows[e.RowIndex].ErrorText = String.Empty;
}

private static DataTable GetData(string selectCommand)
{
    string connectionString =
        "Integrated Security=SSPI;Persist Security Info=False;" +
        "Initial Catalog=Northwind;Data Source=localhost;Packet Size=4096";

    // Connect to the database and fill a data table.
    SqlDataAdapter adapter =
        new SqlDataAdapter(selectCommand, connectionString);
    DataTable data = new DataTable();
    data.Locale = System.Globalization.CultureInfo.InvariantCulture;
    adapter.Fill(data);

    return data;
}

[STAThread]
static void Main()
{
    Application.EnableVisualStyles();
    Application.Run(new Form1());
}
}

```

```

Imports System
Imports System.Data
Imports System.Data.SqlClient
Imports System.Windows.Forms

Public Class Form1
    Inherits System.Windows.Forms.Form

    Private WithEvents dataGridView1 As New DataGridView()
    Private bindingSource1 As New BindingSource()

    Public Sub New()

        ' Initialize the form.
        Me.dataGridView1.Dock = DockStyle.Fill
        Me.Controls.Add(dataGridView1)
        Me.Text = "DataGridView validation demo (disallows empty CompanyName)"

    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles Me.Load

        ' Initialize the BindingSource and bind the DataGridView to it.
        bindingSource1.DataSource = GetData("select * from Customers")
        Me.dataGridView1.DataSource = bindingSource1
        Me.dataGridView1.AutoResizeColumns( _

```

```

        DataGridViewAutoSizeColumnsMode.AllCellsExceptHeader)

End Sub

Private Sub dataGridView1_CellValidating(ByVal sender As Object, _
    ByVal e As DataGridViewCellValidatingEventArgs) _
Handles dataGridView1.CellValidating

    Dim headerText As String = _
        dataGridView1.Columns(e.ColumnIndex).HeaderText

    ' Abort validation if cell is not in the CompanyName column.
    If Not headerText.Equals("CompanyName") Then Return

    ' Confirm that the cell is not empty.
    If (String.IsNullOrEmpty(e.FormattedValue.ToString())) Then
        dataGridView1.Rows(e.RowIndex).ErrorText = _
            "Company Name must not be empty"
        e.Cancel = True
    End If
End Sub

Private Sub dataGridView1_CellEndEdit(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DataGridViewCellEventArgs) _
Handles dataGridView1.CellEndEdit

    ' Clear the row error in case the user presses ESC.
    dataGridView1.Rows(e.RowIndex).ErrorText = String.Empty

End Sub

Private Shared Function GetData(ByVal selectCommand As String) As DataTable

    Dim connectionString As String = _
        "Integrated Security=SSPI;Persist Security Info=False;" + _
        "Initial Catalog=Northwind;Data Source=localhost;Packet Size=4096"

    ' Connect to the database and fill a data table.
    Dim adapter As New SqlDataAdapter(selectCommand, connectionString)
    Dim data As New DataTable()
    data.Locale = System.Globalization.CultureInfo.InvariantCulture
    adapter.Fill(data)

    Return data
End Function

<STAThread()> _
Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())
End Sub

End Class

```

Compiling the Code

This example requires:

- References to the System, System.Data, System.Windows.Forms and System.XML assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

.NET Framework Security

Storing sensitive information, such as a password, within the connection string can affect the security of your application. Using Windows Authentication (also known as integrated security) is a more secure way to control access to a database. For more information, see [Protecting Connection Information](#).

See Also

[DataGridView](#)

[BindingSource](#)

[Walkthrough: Validating Data in the Windows Forms DataGridView Control](#)

[Data Entry in the Windows Forms DataGridView Control](#)

[Walkthrough: Handling Errors that Occur During Data Entry in the Windows Forms DataGridView Control](#)

[Protecting Connection Information](#)

Walkthrough: Handling Errors that Occur During Data Entry in the Windows Forms DataGridView Control

5/4/2018 • 4 min to read • [Edit Online](#)

Handling errors from the underlying data store is a required feature for a data-entry application. The Windows Forms [DataGridView](#) control makes this easy by exposing the [DataError](#) event, which is raised when the data store detects a constraint violation or a broken business rule.

In this walkthrough, you will retrieve rows from the `Customers` table in the Northwind sample database and display them in a [DataGridView](#) control. When a duplicate `CustomerID` value is detected in a new row or an edited existing row, the [DataError](#) event will occur, which will be handled by displaying a [MessageBox](#) that describes the exception.

To copy the code in this topic as a single listing, see [How to: Handle Errors That Occur During Data Entry in the Windows Forms DataGridView Control](#).

Prerequisites

In order to complete this walkthrough, you will need:

- Access to a server that has the Northwind SQL Server sample database.

Creating the Form

To handle data-entry errors in the DataGridView control

1. Create a class that derives from [Form](#) and contains a [DataGridView](#) control and a [BindingSource](#) component.

The following code example provides basic initialization and includes a `Main` method.

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Windows.Forms;

public class Form1 : System.Windows.Forms.Form
{
    private DataGridView dataGridView1 = new DataGridView();
    private BindingSource bindingSource1 = new BindingSource();

    public Form1()
    {
        // Initialize the form.
        this.dataGridView1.Dock = DockStyle.Fill;
        this.Controls.Add(dataGridView1);
        this.Load += new EventHandler(Form1_Load);
    }
}
```

```

Imports System
Imports System.Data
Imports System.Data.SqlClient
Imports System.Windows.Forms

Public Class Form1
    Inherits System.Windows.Forms.Form

    Private WithEvents dataGridView1 As New DataGridView()
    Private bindingSource1 As New BindingSource()

    Public Sub New()

        ' Initialize the form.
        Me.dataGridView1.Dock = DockStyle.Fill
        Me.Controls.Add(dataGridView1)

    End Sub

```

```

[STAThread]
static void Main()
{
    Application.EnableVisualStyles();
    Application.Run(new Form1());
}

```

```

<STAThread()> _
Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())
End Sub

End Class

```

2. Implement a method in your form's class definition for handling the details of connecting to the database.

This code example uses a `GetData` method that returns a populated `DataTable` object. Be sure that you set the `connectionString` variable to a value that is appropriate for your database.

IMPORTANT

Storing sensitive information, such as a password, within the connection string can affect the security of your application. Using Windows Authentication (also known as integrated security) is a more secure way to control access to a database. For more information, see [Protecting Connection Information](#).

```

private static DataTable GetData(string selectCommand)
{
    string connectionString =
        "Integrated Security=SSPI;Persist Security Info=False;" +
        "Initial Catalog=Northwind;Data Source=localhost;Packet Size=4096";

    // Connect to the database and fill a data table, including the
    // schema information that contains the CustomerID column
    // constraint.
    SqlDataAdapter adapter =
        new SqlDataAdapter(selectCommand, connectionString);
    DataTable data = new DataTable();
    data.Locale = System.Globalization.CultureInfo.InvariantCulture;
    adapter.Fill(data);
    adapter.FillSchema(data, SchemaType.Source);

    return data;
}

```

```

Private Shared Function GetData(ByVal selectCommand As String) As DataTable

    Dim connectionString As String = _
        "Integrated Security=SSPI;Persist Security Info=False;" + _
        "Initial Catalog=Northwind;Data Source=localhost;Packet Size=4096"

    ' Connect to the database and fill a data table, including the
    ' schema information that contains the CustomerID column
    ' constraint.
    Dim adapter As New SqlDataAdapter(selectCommand, connectionString)
    Dim data As New DataTable()
    data.Locale = System.Globalization.CultureInfo.InvariantCulture
    adapter.Fill(data)
    adapter.FillSchema(data, SchemaType.Source)

    Return data

End Function

```

3. Implement a handler for your form's `Load` event that initializes the `DataGridView` and `BindingSource` and sets up the data binding.

```

private void Form1_Load(System.Object sender, System.EventArgs e)
{
    // Attach the DataError event to the corresponding event handler.
    this.dataGridView1.DataError +=
        new DataGridViewDataErrorEventHandler(dataGridView1_DataError);

    // Initialize the BindingSource and bind the DataGridView to it.
    bindingSource1.DataSource = GetData("select * from Customers");
    this.dataGridView1.DataSource = bindingSource1;
    this.dataGridView1.AutoResizeColumns(
        DataGridViewAutoSizeColumnsMode.AllCellsExceptHeader);
}

```

```

Private Sub Form1_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Me.Load

    ' Initialize the BindingSource and bind the DataGridView to it.
    bindingSource1.DataSource = GetData("select * from Customers")
    Me.dataGridView1.DataSource = bindingSource1
    Me.dataGridView1.AutoResizeColumns( _
        DataGridViewAutoSizeColumnsMode.AllCellsExceptHeader)

End Sub

```

4. Handle the [DataError](#) event on the [DataGridView](#).

If the context for the error is a commit operation, display the error in a [MessageBox](#).

```

private void dataGridView1_DataError(object sender,
    DataGridViewDataErrorEventArgs e)
{
    // If the data source raises an exception when a cell value is
    // committed, display an error message.
    if (e.Exception != null &&
        e.Context == DataGridViewDataErrorContexts.Commit)
    {
        MessageBox.Show("CustomerID value must be unique.");
    }
}

```

```

Private Sub dataGridView1_DataError(ByVal sender As Object, _
    ByVal e As DataGridViewDataErrorEventArgs) _
    Handles dataGridView1.DataError

    ' If the data source raises an exception when a cell value is
    ' committed, display an error message.
    If e.Exception IsNot Nothing AndAlso _
        e.Context = DataGridViewDataErrorContexts.Commit Then

        MessageBox.Show("CustomerID value must be unique.")

    End If
End Sub

```

Testing the Application

You can now test the form to make sure it behaves as expected.

To test the form

- Press F5 to run the application.

You will see a [DataGridView](#) control filled with data from the Customers table. If you enter a duplicate value for `CustomerID` and commit the edit, the cell value will revert automatically and you will see a [MessageBox](#) that displays the data entry error.

Next Steps

This application gives you a basic understanding of the [DataGridView](#) control's capabilities. You can customize the appearance and behavior of the [DataGridView](#) control in several ways:

- Change border and header styles. For more information, see [How to: Change the Border and Gridline](#)

[Styles in the Windows Forms DataGridView Control.](#)

- Enable or restrict user input to the [DataGridView](#) control. For more information, see [How to: Prevent Row Addition and Deletion in the Windows Forms DataGridView Control](#), and [How to: Make Columns Read-Only in the Windows Forms DataGridView Control](#).
- Validate user input to the [DataGridView](#) control. For more information, see [Walkthrough: Validating Data in the Windows Forms DataGridView Control](#).
- Handle very large data sets using virtual mode. For more information, see [Walkthrough: Implementing Virtual Mode in the Windows Forms DataGridView Control](#).
- Customize the appearance of cells. For more information, see [How to: Customize the Appearance of Cells in the Windows Forms DataGridView Control](#) and [How to: Set Default Cell Styles for the Windows Forms DataGridView Control](#).

See Also

[DataGridView](#)

[BindingSource](#)

[Data Entry in the Windows Forms DataGridView Control](#)

[How to: Handle Errors That Occur During Data Entry in the Windows Forms DataGridView Control](#)

[Walkthrough: Validating Data in the Windows Forms DataGridView Control](#)

[Protecting Connection Information](#)

How to: Handle Errors That Occur During Data Entry in the Windows Forms DataGridView Control

5/4/2018 • 3 min to read • [Edit Online](#)

The following code example demonstrates how to use the `DataGridView` control to report data entry errors to the user.

For a complete explanation of this code example, see [Walkthrough: Handling Errors that Occur During Data Entry in the Windows Forms DataGridView Control](#).

Example

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Windows.Forms;

public class Form1 : System.Windows.Forms.Form
{
    private DataGridView dataGridView1 = new DataGridView();
    private BindingSource bindingSource1 = new BindingSource();

    public Form1()
    {
        // Initialize the form.
        this.dataGridView1.Dock = DockStyle.Fill;
        this.Controls.Add(dataGridView1);
        this.Load += new EventHandler(Form1_Load);
    }

    private void Form1_Load(System.Object sender, System.EventArgs e)
    {
        // Attach the DataError event to the corresponding event handler.
        this.dataGridView1.DataError +=
            new DataGridViewDataErrorEventHandler(dataGridView1_DataError);

        // Initialize the BindingSource and bind the DataGridView to it.
        bindingSource1.DataSource = GetData("select * from Customers");
        this.dataGridView1.DataSource = bindingSource1;
        this.dataGridView1.AutoResizeColumns(
            DataGridViewAutoSizeColumnsMode.AllCellsExceptHeader);
    }

    private void dataGridView1_DataError(object sender,
        DataGridViewDataErrorEventArgs e)
    {
        // If the data source raises an exception when a cell value is
        // committed, display an error message.
        if (e.Exception != null &&
            e.Context == DataGridViewDataErrorContexts.Commit)
        {
            MessageBox.Show("CustomerID value must be unique.");
        }
    }

    private static DataTable GetData(string selectCommand)
    {
        string connectionString =
            "Integrated Security=SSPI;Persist Security Info=False;" +
            "Initial Catalog=Northwind;Data Source=.\\" + selectCommand;
    }
}
```

```

        "Initial Catalog=Northwind;Data Source=localhost;Packet Size=4096";

    // Connect to the database and fill a data table, including the
    // schema information that contains the CustomerID column
    // constraint.
    SqlDataAdapter adapter =
        new SqlDataAdapter(selectCommand, connectionString);
    DataTable data = new DataTable();
    data.Locale = System.Globalization.CultureInfo.InvariantCulture;
    adapter.Fill(data);
    adapter.FillSchema(data, SchemaType.Source);

    return data;
}

[STAThread]
static void Main()
{
    Application.EnableVisualStyles();
    Application.Run(new Form1());
}

```

```

Imports System
Imports System.Data
Imports System.Data.SqlClient
Imports System.Windows.Forms

Public Class Form1
    Inherits System.Windows.Forms.Form

    Private WithEvents dataGridView1 As New DataGridView()
    Private bindingSource1 As New BindingSource()

    Public Sub New()

        ' Initialize the form.
        Me.dataGridView1.Dock = DockStyle.Fill
        Me.Controls.Add(dataGridView1)

    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles Me.Load

        ' Initialize the BindingSource and bind the DataGridView to it.
        bindingSource1.DataSource = GetData("select * from Customers")
        Me.dataGridView1.DataSource = bindingSource1
        Me.dataGridView1.AutoResizeColumns( _
            DataGridViewAutoSizeColumnsMode.AllCellsExceptHeader)

    End Sub

    Private Sub dataGridView1_DataError(ByVal sender As Object, _
        ByVal e As DataGridViewDataErrorEventArgs) _
        Handles dataGridView1.DataError

        ' If the data source raises an exception when a cell value is
        ' committed, display an error message.
        If e.Exception IsNot Nothing AndAlso _
            e.Context = DataGridViewDataErrorContexts.Commit Then

            MessageBox.Show("CustomerID value must be unique.")

        End If
    End Sub

```

```

End Sub

Private Shared Function GetData(ByVal selectCommand As String) As DataTable

    Dim connectionString As String = _
        "Integrated Security=SSPI;Persist Security Info=False;" + _
        "Initial Catalog=Northwind;Data Source=localhost;Packet Size=4096"

    ' Connect to the database and fill a data table, including the
    ' schema information that contains the CustomerID column
    ' constraint.
    Dim adapter As New SqlDataAdapter(selectCommand, connectionString)
    Dim data As New DataTable()
    data.Locale = System.Globalization.CultureInfo.InvariantCulture
    adapter.Fill(data)
    adapter.FillSchema(data, SchemaType.Source)

    Return data
End Function

<STAThread()> _
Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())
End Sub

End Class

```

Compiling the Code

This example requires:

- References to the System, System.Data, System.Windows.Forms, and System.XML assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

.NET Framework Security

Storing sensitive information, such as a password, within the connection string can affect the security of your application. Using Windows Authentication (also known as integrated security) is a more secure way to control access to a database. For more information, see [Protecting Connection Information](#).

See Also

[DataGridView](#)

[BindingSource](#)

[Walkthrough: Handling Errors that Occur During Data Entry in the Windows Forms DataGridView Control](#)

[Data Entry in the Windows Forms DataGridView Control](#)

[Walkthrough: Validating Data in the Windows Forms DataGridView Control](#)

[Protecting Connection Information](#)

Selection and Clipboard Use with the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

The `DataGridView` control provides you with a variety of options for configuring how users can select cells, rows, and columns. For example, you can enable single or multiple selection, selection of whole rows or columns when users click cells, or selection of whole rows or columns only when users click their headers, which enables cell selection as well. If you want to provide your own user interface for selection, you can disable ordinary selection and handle all selection programmatically. Additionally, you can enable users to copy the selected values to the Clipboard.

In This Section

[Selection Modes in the Windows Forms DataGridView Control](#)

Describes the options for user and programmatic selection in the control.

[How to: Set the Selection Mode of the Windows Forms DataGridView Control](#)

Describes how to configure the control for single-row selection when a user clicks a cell.

[How to: Get the Selected Cells, Rows, and Columns in the Windows Forms DataGridView Control](#)

Describes how to work with the selected cell, row, and column collections.

[How to: Enable Users to Copy Multiple Cells to the Clipboard from the Windows Forms DataGridView Control](#)

Describes how to enable Clipboard support in the control.

Reference

[DataGridView](#)

Provides reference documentation for the `DataGridView` control.

[DataGridViewSelectionMode](#)

Provides reference documentation for the `SelectionMode` property.

[ClipboardCopyMode](#)

Provides reference documentation for the `ClipboardCopyMode` property.

[DataGridViewSelectedCellCollection](#)

Provides reference documentation for the `DataGridViewSelectedCellCollection` class.

[DataGridViewSelectedRowCollection](#)

Provides reference documentation for the `DataGridViewSelectedRowCollection` class.

[DataGridViewSelectedColumnCollection](#)

Provides reference documentation for the `DataGridViewSelectedColumnCollection` class.

See Also

[DataGridView Control](#)

[Default Keyboard and Mouse Handling in the Windows Forms DataGridView Control](#)

Selection Modes in the Windows Forms DataGridView Control

5/4/2018 • 2 min to read • [Edit Online](#)

Sometimes you want your application to perform actions based on user selections within a [DataGridView](#) control. Depending on the actions, you may want to restrict the kinds of selection that are possible. For example, suppose your application can print a report for the currently selected record. In this case, you may want to configure the [DataGridView](#) control so that clicking anywhere within a row always selects the entire row, and so that only one row at a time can be selected.

You can specify the selections allowed by setting the [DataGridView.SelectionMode](#) property to one of the following [DataGridViewSelectionMode](#) enumeration values.

DataGridViewSelectionMode Value	Description
CellSelect	Clicking a cell selects it. Row and column headers cannot be used for selection.
ColumnHeaderSelect	Clicking a cell selects it. Clicking a column header selects the entire column. Column headers cannot be used for sorting.
FullColumnSelect	Clicking a cell or a column header selects the entire column. Column headers cannot be used for sorting.
FullRowSelect	Clicking a cell or a row header selects the entire row.
RowHeaderSelect	Default selection mode. Clicking a cell selects it. Clicking a row header selects the entire row.

NOTE

Changing the selection mode at run time automatically clears the current selection.

By default, users can select multiple rows, columns, or cells by dragging with the mouse, pressing CTRL or SHIFT while selecting to extend or modify a selection, or clicking the top-left header cell to select all cells in the control. To prevent this behavior, set the [MultiSelect](#) property to `false`.

The [FullRowSelect](#) and [RowHeaderSelect](#) modes allow users to delete rows by selecting them and pressing the DELETE key. Users can delete rows only when the current cell is not in edit mode, the [AllowUserToDeleteRows](#) property is set to `true`, and the underlying data source supports user-driven row deletion. Note that these settings do not prevent programmatic row deletion.

Programmatic Selection

The current selection mode restricts the behavior of programmatic selection as well as user selection. You can change the current selection programmatically by setting the [Selected](#) property of any cells, rows, or columns present in the [DataGridView](#) control. You can also select all cells in the control through the [SelectAll](#) method, depending on the selection mode. To clear the selection, use the [ClearSelection](#) method.

If the [MultiSelect](#) property is set to `true`, you can add [DataGridView](#) elements to or remove them from the selection by changing the `Selected` property of the element. Otherwise, setting the `Selected` property to `true` for one element automatically removes other elements from the selection.

Note that changing the value of the [CurrentCell](#) property does not alter the current selection.

You can retrieve a collection of the currently selected cells, rows, or columns through the [SelectedCells](#), [SelectedRows](#), and [SelectedColumns](#) properties of the [DataGridView](#) control. Accessing these properties is inefficient when every cell in the control is selected. To avoid a performance penalty in this case, use the [AreAllCellsSelected](#) method first. Additionally, accessing these collections to determine the number of selected cells, rows, or columns can be inefficient. Instead, you should use the [GetCellCount](#), [GetRowCount](#), or [GetColumnCount](#) method, passing in the [Selected](#) value.

TIP

Example code that demonstrates the programmatic use of selected cells can be found in the [DataGridView](#) class overview.

See Also

[DataGridView](#)

[MultiSelect](#)

[SelectionMode](#)

[DataGridViewSelectionMode](#)

[Selection and Clipboard Use with the Windows Forms DataGridView Control](#)

[How to: Set the Selection Mode of the Windows Forms DataGridView Control](#)

How to: Set the Selection Mode of the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

The following code example demonstrates how to configure a [DataGridView](#) control so that clicking anywhere within a row automatically selects the entire row, and so that only one row at a time can be selected.

Example

```
this.dataGridView1.SelectionMode =  
    DataGridViewSelectionMode.FullRowSelect;  
this.dataGridView1.MultiSelect = false;
```

```
With Me.dataGridView1  
    .SelectionMode = DataGridViewSelectionMode.FullRowSelect  
    .MultiSelect = False  
End With
```

Compiling the Code

This example requires:

- A [DataGridView](#) control named `dataGridView1`.
- References to the [System](#) and [System.Windows.Forms](#) assemblies.

See Also

[DataGridView](#)

[MultiSelect](#)

[SelectionMode](#)

[DataGridViewSelectionMode](#)

[Selection and Clipboard Use with the Windows Forms DataGridView Control](#)

[Selection Modes in the Windows Forms DataGridView Control](#)

How to: Get the Selected Cells, Rows, and Columns in the Windows Forms DataGridView Control

5/4/2018 • 2 min to read • [Edit Online](#)

You can get the selected cells, rows, or columns from a [DataGridView](#) control by using the corresponding properties: [SelectedCells](#), [SelectedRows](#), and [SelectedColumns](#). In the following procedures, you will get the selected cells and display their row and column indexes in a [MessageBox](#).

To get the selected cells in a DataGridView control

- Use the [SelectedCells](#) property.

NOTE

Use the [AreAllCellsSelected](#) method to avoid showing a potentially large number of cells.

```
private void selectedCellsButton_Click(object sender, System.EventArgs e)
{
    Int32 selectedCellCount =
        dataGridView1.GetCellCount(DataGridViewElementStates.Selected);
    if (selectedCellCount > 0)
    {
        if (dataGridView1.AreAllCellsSelected(true))
        {
            MessageBox.Show("All cells are selected", "Selected Cells");
        }
        else
        {
            System.Text.StringBuilder sb =
                new System.Text.StringBuilder();

            for (int i = 0;
                i < selectedCellCount; i++)
            {
                sb.Append("Row: ");
                sb.Append(dataGridView1.SelectedCells[i].RowIndex
                    .ToString());
                sb.Append(", Column: ");
                sb.Append(dataGridView1.SelectedCells[i].ColumnIndex
                    .ToString());
                sb.Append(Environment.NewLine);
            }

            sb.Append("Total: " + selectedCellCount.ToString());
            MessageBox.Show(sb.ToString(), "Selected Cells");
        }
    }
}
```

```

Private Sub selectedCellsButton_Click( _
    ByVal sender As Object, ByVal e As System.EventArgs) _
Handles selectedCellsButton.Click

    Dim selectedCellCount As Integer = _
        dataGridView1.GetCellCount(DataGridViewElementStates.Selected)

    If selectedCellCount > 0 Then

        If dataGridView1.AreAllCellsSelected(True) Then

            MessageBox.Show("All cells are selected", "Selected Cells")

        Else

            Dim sb As New System.Text.StringBuilder()

            Dim i As Integer
            For i = 0 To selectedCellCount - 1

                sb.Append("Row: ")
                sb.Append(dataGridView1.SelectedCells(i).RowIndex _
                    .ToString())
                sb.Append(", Column: ")
                sb.Append(dataGridView1.SelectedCells(i).ColumnIndex _
                    .ToString())
                sb.Append(Environment.NewLine)

            Next i

            sb.Append("Total: " + selectedCellCount.ToString())
            MessageBox.Show(sb.ToString(), "Selected Cells")

        End If

    End If

End Sub

```

To get the selected rows in a DataGridView control

- Use the [SelectedRows](#) property. To enable users to select rows, you must set the [SelectionMode](#) property to [FullRowSelect](#) or [RowHeaderSelect](#).

```

private void selectedRowsButton_Click(object sender, System.EventArgs e)
{
    Int32 selectedRowCount =
        dataGridView1.Rows.GetRowCount(DataGridViewElementStates.Selected);
    if (selectedRowCount > 0)
    {
        System.Text.StringBuilder sb = new System.Text.StringBuilder();

        for (int i = 0; i < selectedRowCount; i++)
        {
            sb.Append("Row: ");
            sb.Append(dataGridView1.SelectedRows[i].Index.ToString());
            sb.Append(Environment.NewLine);
        }

        sb.Append("Total: " + selectedRowCount.ToString());
        MessageBox.Show(sb.ToString(), "Selected Rows");
    }
}

```

```

Private Sub selectedRowsButton_Click( _
    ByVal sender As Object, ByVal e As System.EventArgs) _
Handles selectedRowsButton.Click

    Dim selectedRowCount As Integer = _
        dataGridView1.Rows.GetRowCount(DataGridViewElementStates.Selected)

    If selectedRowCount > 0 Then

        Dim sb As New System.Text.StringBuilder()

        Dim i As Integer
        For i = 0 To selectedRowCount - 1

            sb.Append("Row: ")
            sb.Append(dataGridView1.SelectedRows(i).Index.ToString())
            sb.Append(Environment.NewLine)

        Next i

        sb.Append("Total: " + selectedRowCount.ToString())
        MessageBox.Show(sb.ToString(), "Selected Rows")

    End If

End Sub

```

To get the selected columns in a DataGridView control

- Use the [SelectedColumns](#) property. To enable users to select columns, you must set the [SelectionMode](#) property to [FullColumnSelect](#) or [ColumnHeaderSelect](#).

```

private void selectedColumnsButton_Click(object sender, System.EventArgs e)
{
    Int32 selectedColumnCount = dataGridView1.Columns
        .GetColumnCount(DataGridViewElementStates.Selected);
    if (selectedColumnCount > 0)
    {
        System.Text.StringBuilder sb = new System.Text.StringBuilder();

        for (int i = 0; i < selectedColumnCount; i++)
        {
            sb.Append("Column: ");
            sb.Append(dataGridView1.SelectedColumns[i].Index
                .ToString());
            sb.Append(Environment.NewLine);
        }

        sb.Append("Total: " + selectedColumnCount.ToString());
        MessageBox.Show(sb.ToString(), "Selected Columns");
    }
}

```

```

Private Sub selectedColumnsButton_Click( _
    ByVal sender As Object, ByVal e As System.EventArgs) _
Handles selectedColumnsButton.Click

    Dim selectedColumnCount As Integer = dataGridView1.Columns _
        .GetColumnCount(DataGridViewElementStates.Selected)

    If selectedColumnCount > 0 Then

        Dim sb As New System.Text.StringBuilder()

        Dim i As Integer
        For i = 0 To selectedColumnCount - 1

            sb.Append("Column: ")
            sb.Append(dataGridView1.SelectedColumns(i).Index.ToString())
            sb.Append(Environment.NewLine)

        Next i

        sb.Append("Total: " + selectedColumnCount.ToString())
        MessageBox.Show(sb.ToString(), "Selected Columns")

    End If

End Sub

```

Compiling the Code

This example requires:

- [Button](#) controls named `selectedCellsButton`, `selectedRowsButton`, and `selectedColumnsButton`, each with handlers for the [Click](#) event attached.
- A [DataGridView](#) control named `dataGridView1`.
- References to the [System](#), [System.Windows.Forms](#), and [System.Text](#) assemblies.

Robust Programming

The collections described in this topic do not perform efficiently when large numbers of cells, rows, or columns are selected. For more information about using these collections with large amounts of data, see [Best Practices for Scaling the Windows Forms DataGridView Control](#).

See Also

[DataGridView](#)
[SelectionMode](#)
[AreAllCellsSelected](#)
[SelectedCells](#)
[SelectedRows](#)
[SelectedColumns](#)
[Selection and Clipboard Use with the Windows Forms DataGridView Control](#)

How to: Enable Users to Copy Multiple Cells to the Clipboard from the Windows Forms DataGridView Control

5/4/2018 • 3 min to read • [Edit Online](#)

When you enable cell copying, you make the data in your [DataGridView](#) control easily accessible to other applications through the [Clipboard](#). The values of the selected cells are converted to strings and added to the Clipboard as tab-delimited text values for pasting into applications like Notepad and Excel, and as an HTML-formatted table for pasting into applications like Word.

You can configure cell copying to copy cell values only, to include row and column header text in the Clipboard data, or to include header text only when users select entire rows or columns.

Depending on the selection mode, users can select multiple disconnected groups of cells. When a user copies cells to the Clipboard, rows and columns with no selected cells are not copied. All other rows or columns become rows and columns in the table of data copied to the Clipboard. Unselected cells in these rows or columns are copied as blank placeholders to the Clipboard.

To enable cell copying

- Set the [DataGridView.ClipboardCopyMode](#) property.

```
this.DataGridView1.ClipboardCopyMode =  
    DataGridViewClipboardCopyMode.EnableWithoutHeaderText;
```

```
Me.DataGridView1.ClipboardCopyMode = _  
    DataGridViewClipboardCopyMode.EnableWithoutHeaderText
```

Example

The following complete code example demonstrates how cells are copied to the Clipboard. This example includes a button that copies the selected cells to the Clipboard using the [DataGridView.GetClipboardContent](#) method and displays the Clipboard contents in a text box.

```
using System;  
using System.Windows.Forms;  
  
public class Form1 : Form  
{  
    private DataGridView DataGridView1 = new DataGridView();  
    private Button CopyPasteButton = new Button();  
    private TextBox TextBox1 = new TextBox();  
  
    [STAThreadAttribute()]  
    public static void Main()  
    {  
        Application.Run(new Form1());  
    }  
  
    public Form1()  
    {  
        this.DataGridView1.AllowUserToAddRows = false;
```

```

        this.DataGridView1.Dock = DockStyle.Fill;
        this.Controls.Add(this.DataGridView1);

        this.CopyPasteButton.Text = "copy/paste selected cells";
        this.CopyPasteButton.Dock = DockStyle.Top;
        this.CopyPasteButton.Click += new EventHandler(CopyPasteButton_Click);
        this.Controls.Add(this.CopyPasteButton);

        this.TextBox1.Multiline = true;
        this.TextBox1.Height = 100;
        this.TextBox1.Dock = DockStyle.Bottom;
        this.Controls.Add(this.TextBox1);

        this.Load += new EventHandler(Form1_Load);
        this.Text = "DataGridView Clipboard demo";
    }

    private void Form1_Load(object sender, System.EventArgs e)
    {
        // Initialize the DataGridView control.
        this.DataGridView1.ColumnCount = 5;
        this.DataGridView1.Rows.Add(new string[] { "A", "B", "C", "D", "E" });
        this.DataGridView1.Rows.Add(new string[] { "F", "G", "H", "I", "J" });
        this.DataGridView1.Rows.Add(new string[] { "K", "L", "M", "N", "O" });
        this.DataGridView1.Rows.Add(new string[] { "P", "Q", "R", "S", "T" });
        this.DataGridView1.Rows.Add(new string[] { "U", "V", "W", "X", "Y" });
        this.DataGridView1.AutoResizeColumns();
        this.DataGridView1.ClipboardCopyMode =
            DataGridViewClipboardCopyMode.EnableWithoutHeaderText;
    }

    private void CopyPasteButton_Click(object sender, System.EventArgs e)
    {
        if (this.DataGridView1
            .GetCellCount(DataGridViewElementStates.Selected) > 0)
        {
            try
            {
                // Add the selection to the clipboard.
                Clipboard.SetDataObject(
                    this.DataGridView1.GetClipboardContent());

                // Replace the text box contents with the clipboard text.
                this.TextBox1.Text = Clipboard.GetText();
            }
            catch (System.Runtime.InteropServices.ExternalException)
            {
                this.TextBox1.Text =
                    "The Clipboard could not be accessed. Please try again.";
            }
        }
    }
}

```

```

Imports System
Imports System.Windows.Forms

Public Class Form1
Inherits Form

Private WithEvents DataGridView1 As New DataGridView()
Private WithEvents CopyPasteButton As New Button()
Private TextBox1 As New TextBox()

<STAThreadAttribute()> _
Public Shared Sub Main()

```

```

Application.Run(New Form1())
End Sub

Public Sub New()

    Me.DataGridView1.AllowUserToAddRows = False
    Me.DataGridView1.Dock = DockStyle.Fill
    Me.Controls.Add(Me.DataGridView1)

    Me.CopyPasteButton.Text = "copy/paste selected cells"
    Me.CopyPasteButton.Dock = DockStyle.Top
    Me.Controls.Add(Me.CopyPasteButton)

    Me.TextBox1.Multiline = True
    Me.TextBox1.Height = 100
    Me.TextBox1.Dock = DockStyle.Bottom
    Me.Controls.Add(Me.TextBox1)

    Me.Text = "DataGridView Clipboard demo"

End Sub

Private Sub Form1_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load

    ' Initialize the DataGridView control.
    Me.DataGridView1.ColumnCount = 5
    Me.DataGridView1.Rows.Add(New String() {"A", "B", "C", "D", "E"})
    Me.DataGridView1.Rows.Add(New String() {"F", "G", "H", "I", "J"})
    Me.DataGridView1.Rows.Add(New String() {"K", "L", "M", "N", "O"})
    Me.DataGridView1.Rows.Add(New String() {"P", "Q", "R", "S", "T"})
    Me.DataGridView1.Rows.Add(New String() {"U", "V", "W", "X", "Y"})
    Me.DataGridView1.AutoResizeColumns()
    Me.DataGridView1.ClipboardCopyMode = _
        DataGridViewClipboardCopyMode.EnableWithoutHeaderText

End Sub

Private Sub CopyPasteButton_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles CopyPasteButton.Click

    If Me.DataGridView1.GetCellCount( _
        DataGridViewElementStates.Selected) > 0 Then

        Try

            ' Add the selection to the clipboard.
            Clipboard.SetDataObject( _
                Me.DataGridView1.GetClipboardContent())

            ' Replace the text box contents with the clipboard text.
            Me.TextBox1.Text = Clipboard.GetText()

        Catch ex As System.Runtime.InteropServices.ExternalException
            Me.TextBox1.Text = _
                "The Clipboard could not be accessed. Please try again."
        End Try

    End If

End Sub

End Class

```

Compiling the Code

This code requires:

- References to the N:System and N:System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[DataGridView](#)

[ClipboardCopyMode](#)

[GetClipboardContent](#)

[Selection and Clipboard Use with the Windows Forms DataGridView Control](#)

Programming with Cells, Rows, and Columns in the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

This section provides topics that demonstrate various programming tasks involving cell, row, and column objects.

In This Section

[How to: Add ToolTips to Individual Cells in a Windows Forms DataGridView Control](#)

Describes how to handle the [CellFormatting](#) event to provide different ToolTips for individual cells.

[How to: Perform a Custom Action Based on Changes in a Cell of a Windows Forms DataGridView Control](#)

Describes how to handle the [CellValueChanged](#) and [CellStateChanged](#) events.

[How to: Manipulate Bands in the Windows Forms DataGridView Control](#)

Describes how to program with objects of type [DataGridViewBand](#), which is the base type for rows and columns.

[How to: Manipulate Rows in the Windows Forms DataGridView Control](#)

Describes how to program with objects of type [DataGridViewRow](#).

[How to: Manipulate Columns in the Windows Forms DataGridView Control](#)

Describes how to program with objects of type [DataGridViewColumn](#).

[How to: Work with Image Columns in the Windows Forms DataGridView Control](#)

Describes how to program with the [DataGridViewImageColumn](#) class.

Reference

[DataGridView](#)

Provides reference documentation for the [DataGridView](#) control.

[DataGridViewCell](#)

Provides reference documentation for the [DataGridViewCell](#) class.

[DataGridViewRow](#)

Provides reference documentation for the [DataGridViewRow](#) class.

[DataGridViewColumn](#)

Provides reference documentation for the [DataGridViewColumn](#) class.

Related Sections

[Basic Column, Row, and Cell Features in the Windows Forms DataGridView Control](#)

Provides topics that describe commonly used cell, row, and column properties.

See Also

[DataGridView Control](#)

[Column Types in the Windows Forms DataGridView Control](#)

How to: Add ToolTips to Individual Cells in a Windows Forms DataGridView Control

5/4/2018 • 2 min to read • [Edit Online](#)

By default, ToolTips are used to display the values of [DataGridView](#) cells that are too small to show their entire contents. You can override this behavior, however, to set ToolTip-text values for individual cells. This is useful to display to users additional information about a cell or to provide to users an alternate description of the cell contents. For example, if you have a row that displays status icons, you may want to provide text explanations using ToolTips.

You can also disable the display of cell-level ToolTips by setting the [DataGridView.ShowCellToolTips](#) property to `false`.

To add a ToolTip to a cell

- Set the [DataGridViewCell.ToolTipText](#) property.

```
// Sets the ToolTip text for cells in the Rating column.
void dataGridView1_CellFormatting(Object^ /*sender*/,
    DataGridViewCellFormattingEventArgs^ e)
{
    if ( (e->ColumnIndex == this->dataGridView1->Columns["Rating"]->Index)
        && e->Value != nullptr )
    {
        DataGridViewCell^ cell =
            this->dataGridView1->Rows[e->RowIndex]->Cells[e->ColumnIndex];
        if (e->Value->Equals("*"))
        {
            cell->ToolTipText = "very bad";
        }
        else if (e->Value->Equals("**"))
        {
            cell->ToolTipText = "bad";
        }
        else if (e->Value->Equals("/**"))
        {
            cell->ToolTipText = "good";
        }
        else if (e->Value->Equals("****"))
        {
            cell->ToolTipText = "very good";
        }
    }
}
```

```

// Sets the ToolTip text for cells in the Rating column.
void dataGridView1_CellFormatting(object sender,
    DataGridViewCellFormattingEventArgs e)
{
    if ( (e.ColumnIndex == this.dataGridView1.Columns["Rating"].Index)
        && e.Value != null )
    {
        DataGridViewCell cell =
            this.dataGridView1.Rows[e.RowIndex].Cells[e.ColumnIndex];
        if (e.Value.Equals("*"))
        {
            cell.ToolTipText = "very bad";
        }
        else if (e.Value.Equals("**"))
        {
            cell.ToolTipText = "bad";
        }
        else if (e.Value.Equals("/**"))
        {
            cell.ToolTipText = "good";
        }
        else if (e.Value.Equals("****"))
        {
            cell.ToolTipText = "very good";
        }
    }
}

```

```

' Sets the ToolTip text for cells in the Rating column.
Sub dataGridView1_CellFormatting(ByVal sender As Object, _
    ByVal e As DataGridViewCellFormattingEventArgs) _
Handles dataGridView1.CellFormatting

If e.ColumnIndex = Me.dataGridView1.Columns("Rating").Index _
    AndAlso (e.Value IsNot Nothing) Then

    With Me.dataGridView1.Rows(e.RowIndex).Cells(e.ColumnIndex)

        If e.Value.Equals("*") Then
            .ToolTipText = "very bad"
        ElseIf e.Value.Equals("**") Then
            .ToolTipText = "bad"
        ElseIf e.Value.Equals("/**") Then
            .ToolTipText = "good"
        ElseIf e.Value.Equals("****") Then
            .ToolTipText = "very good"
        End If

    End With

End If

End Sub 'dataGridView1_CellFormatting

```

Compiling the Code

- This example requires:
- A [DataGridView](#) control named `dataGridView1` that contains a column named `Rating` for displaying string values of one through four asterisk ("*") symbols. The [CellFormatting](#) event of the control must be associated with the event handler method shown in the example.
- References to the [System](#) and [System.Windows.Forms](#) assemblies.

Robust Programming

When you bind the [DataGridView](#) control to an external data source or provide your own data source by implementing virtual mode, you might encounter performance issues. To avoid a performance penalty when working with large amounts of data, handle the [CellToolTipTextNeeded](#) event rather than setting the [ToolTipText](#) property of multiple cells. When you handle this event, getting the value of a cell [ToolTipText](#) property raises the event and returns the value of the [DataGridViewCell.ToolTipTextNeededEventArgs.ToolTipText](#) property as specified in the event handler.

See Also

[DataGridView](#)

[DataGridView.ShowCellToolTips](#)

[DataGridView.CellToolTipTextNeeded](#)

[DataGridViewCell](#)

[DataGridViewCell.ToolTipText](#)

[Programming with Cells, Rows, and Columns in the Windows Forms DataGridView Control](#)

How to: Perform a Custom Action Based on Changes in a Cell of a Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

The [DataGridView](#) control has a number of events you can use to detect changes in the state of [DataGridView](#) cells. Two of the most commonly used are the [CellValueChanged](#) and [CellStateChanged](#) events.

To detect changes in the values of DataGridView cells

- Write a handler for the [CellValueChanged](#) event.

```
private void dataGridView1_CellValueChanged(object sender,
    DataGridViewCellEventArgs e)
{
    string msg = String.Format(
        "Cell at row {0}, column {1} value changed",
        e.RowIndex, e.ColumnIndex);
    MessageBox.Show(msg, "Cell Value Changed");
}
```

```
Private Sub dataGridView1_CellValueChanged(ByVal sender As Object, _
    ByVal e As DataGridViewCellEventArgs) _
Handles dataGridView1.CellValueChanged

    Dim msg As String = String.Format( _
        "Cell at row {0}, column {1} value changed", _
        e.RowIndex, e.ColumnIndex)
    MessageBox.Show(msg, "Cell Value Changed")

End Sub
```

To detect changes in the states of DataGridView cells

- Write a handler for the [CellStateChanged](#) event.

```
private void dataGridView1_CellStateChanged(object sender,
    DataGridViewCellStateChangedEventArgs e)
{
    DataGridViewElementStates state = e.StateChanged;
    string msg = String.Format("Row {0}, Column {1}, {2}",
        e.Cell.RowIndex, e.Cell.ColumnIndex, e.StateChanged);
    MessageBox.Show(msg, "Cell State Changed");
}
```

```
Private Sub dataGridView1_CellStateChanged(ByVal sender As Object, _
    ByVal e As DataGridViewCellStateChangedEventArgs) _
Handles dataGridView1.CellStateChanged

    Dim state As DataGridViewElementStates = e.StateChanged
    Dim msg As String = String.Format( _
        "Row {0}, Column {1}, {2}", _
        e.Cell.RowIndex, e.Cell.ColumnIndex, e.StateChanged)
    MessageBox.Show(msg, "Cell State Changed")

End Sub
```

Compiling the Code

This example requires:

- A [DataGridView](#) control named `dataGridView1`. For C#, the event handlers must be connected to the corresponding events.
- References to the [System](#) and [System.Windows.Forms](#) assemblies.

See Also

[DataGridView](#)

[DataGridViewCellValueChanged](#)

[DataGridView.CellStateChanged](#)

[Programming with Cells, Rows, and Columns in the Windows Forms DataGridView Control](#)

[Walkthrough: Validating Data in the Windows Forms DataGridView Control](#)

How to: Manipulate Bands in the Windows Forms DataGridView Control

5/4/2018 • 13 min to read • [Edit Online](#)

The following code example shows various ways to manipulate [DataGridView](#) rows and columns using properties of the [DataGridViewBand](#) class from which the [DataGridViewRow](#) and [DataGridViewColumn](#) classes derive.

Example

```
#using <System.Drawing.dll>
#using <System.dll>
#using <system.windows.forms.dll>
#using <system.drawing.dll>

using namespace System::Drawing;
using namespace System::Windows::Forms;
using namespace System;
using namespace System::Collections;
public ref class DataGridViewBandDemo: public Form
{
private:

#pragma region S " form setup "

public:
    DataGridViewBandDemo()
    {
        Button1 = gcnew Button;
        Button2 = gcnew Button;
        Button3 = gcnew Button;
        Button4 = gcnew Button;
        Button5 = gcnew Button;
        Button6 = gcnew Button;
        Button7 = gcnew Button;
        Button8 = gcnew Button;
        Button9 = gcnew Button;
        Button10 = gcnew Button;
        FlowLayoutPanel1 = gcnew FlowLayoutPanel;
        InitializeComponent();
        thirdColumnHeader = L"Main Ingredients";
        boringMeatloaf = L"ground beef";
        boringMeatloafRanking = L"**";
        AddButton( Button1, L"Reset", gcnew EventHandler( this, &DataGridViewBandDemo::Button1_Click ) );
        AddButton( Button2, L"Change Column 3 Header", gcnew EventHandler( this,
&DataGridViewBandDemo::Button2_Click ) );
        AddButton( Button3, L"Change Meatloaf Recipe", gcnew EventHandler( this,
&DataGridViewBandDemo::Button3_Click ) );
        AddAdditionalButtons();
        InitializeDataGridView();
    }

    DataGridView^ dataGridView;
    Button^ Button1;
    Button^ Button2;
    Button^ Button3;
    Button^ Button4;
    Button^ Button5;
    Button^ Button6;
    Button^ Button7;
    Button^ Button8;
```

```

button^ Button8;
Button^ Button9;
Button^ Button10;
FlowLayoutPanel^ FlowLayoutPanel1;

private:
void InitializeComponent()
{
    FlowLayoutPanel1->Location = Point(454,0);
    FlowLayoutPanel1->AutoSize = true;
    FlowLayoutPanel1->FlowDirection = FlowDirection::TopDown;
    AutoSize = true;
    ClientSize = System::Drawing::Size( 614, 360 );
    FlowLayoutPanel1->Name = L"flowlayoutpanel";
    Controls->Add( this->FlowLayoutPanel1 );
    Text = this->GetType()->Name;
}

#pragma endregion
#pragma region S " setup DataGridView "
String^ thirdColumnHeader;
String^ boringMeatloaf;
String^ boringMeatloafRanking;
bool boringRecipe;
Boolean shortMode;
void InitializeDataGridView()
{
    dataGridView = gcnew System::Windows::Forms::DataGridView;
    Controls->Add( dataGridView );
    dataGridView->Size = System::Drawing::Size( 300, 200 );

    // Create an unbound DataGridView by declaring a
    // column count.
    dataGridView->ColumnCount = 4;
    AdjustDataGridViewSizing();

    // Set the column header style.
    DataGridViewCellStyle^ columnHeaderStyle = gcnew DataGridViewCellStyle;
    columnHeaderStyle->BackColor = Color::Aqua;
    columnHeaderStyle->Font = gcnew System::Drawing::Font( L"Verdana",10,FontStyle::Bold );
    dataGridView->ColumnHeadersDefaultCellStyle = columnHeaderStyle;

    // Set the column header names.
    dataGridView->Columns[ 0 ]->Name = L"Recipe";
    dataGridView->Columns[ 1 ]->Name = L"Category";
    dataGridView->Columns[ 2 ]->Name = thirdColumnHeader;
    dataGridView->Columns[ 3 ]->Name = L"Rating";

    // Populate the rows.
    array<String^>^row1 = gcnew array<String^>{
        L"Meatloaf",L"Main Dish",boringMeatloaf,boringMeatloafRanking
    };
    array<String^>^row2 = gcnew array<String^>{
        L"Key Lime Pie",L"Dessert",L"lime juice, evaporated milk",L"****"
    };
    array<String^>^row3 = gcnew array<String^>{
        L"Orange-Salsa Pork Chops",L"Main Dish",L"pork chops, salsa, orange juice",L"****"
    };
    array<String^>^row4 = gcnew array<String^>{
        L"Black Bean and Rice Salad",L"Salad",L"black beans, brown rice",L"****"
    };
    array<String^>^row5 = gcnew array<String^>{
        L"Chocolate Cheesecake",L"Dessert",L"cream cheese",L"***"
    };
    array<String^>^row6 = gcnew array<String^>{
        L"Black Bean Dip",L"Appetizer",L"black beans, sour cream",L"***"
    };
    array<Object^>^rows = gcnew array<Object^>{

```

```

    row1, row2, row3, row4, row5, row6
};

System::Collections::IEnumerator^ myEnum = rows->GetEnumerator();
while ( myEnum->MoveNext() )
{
    array<String^>^rowArray = safe_cast<array<String^>^>(myEnum->Current);
    dataGridView->Rows->Add( rowArray );
}

PostRowCreation();
shortMode = false;
boringRecipe = true;
}

void AddButton( Button^ button, String^ buttonLabel, EventHandler^ handler )
{
    FlowLayoutPanel1->Controls->Add( button );
    button->TabIndex = FlowLayoutPanel1->Controls->Count;
    button->Text = buttonLabel;
    button->AutoSize = true;
    button->Click += handler;
}

// Reset columns to initial disorderly arrangement.
void Button1_Click( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    Controls->Remove( dataGridView );
    dataGridView->~DataGridView();
    InitializeDataGridView();
}

// Change the header in column three.
void Button2_Click( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    Toggle( &shortMode );
    if ( shortMode )
    {
        dataGridView->Columns[ 2 ]->HeaderText = L"S";
    }
    else
    {
        dataGridView->Columns[ 2 ]->HeaderText = thirdColumnHeader;
    }
}

void Toggle( interior_ptr<Boolean> toggleThis )
{
    *toggleThis = ! *toggleThis;
}

// Change the meatloaf recipe.
void Button3_Click( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    Toggle( &boringRecipe );
    if ( boringRecipe )
    {
        SetMeatloaf( boringMeatloaf, boringMeatloafRanking );
    }
    else
    {
        String^ greatMeatloafRecipe = L"1 lb. lean ground beef, "
        L"1/2 cup bread crumbs, 1/4 cup ketchup,"
        L"1/3 tsp onion powder,"
        L"1 clove of garlic, 1/2 pack onion soup mix "
        L" dash of your favorite BBQ Sauce";
        SetMeatloaf( greatMeatloafRecipe, L"***" );
    }
}

```

```

    }

}

void SetMeatloaf( String^ recipe, String^ rating )
{
    dataGridView->Rows[ 0 ]->Cells[ 2 ]->Value = recipe;
    dataGridView->Rows[ 0 ]->Cells[ 3 ]->Value = rating;
}

#pragma endregion
#pragma region S " demonstration code "
void AddAdditionalButtons()
{
    AddButton( Button4, L"Freeze First Row", gcnew EventHandler( this, &DataGridViewBandDemo::Button4_Click ) );
    AddButton( Button5, L"Freeze Second Column", gcnew EventHandler( this,
&DataGridViewBandDemo::Button5_Click ) );
    AddButton( Button6, L"Hide Salad Row", gcnew EventHandler( this, &DataGridViewBandDemo::Button6_Click ) );
    AddButton( Button7, L"Disable First Column Resizing", gcnew EventHandler( this,
&DataGridViewBandDemo::Button7_Click ) );
    AddButton( Button8, L"Make ReadOnly", gcnew EventHandler( this, &DataGridViewBandDemo::Button8_Click ) );
    AddButton( Button9, L"Style Using Tag", gcnew EventHandler( this, &DataGridViewBandDemo::Button9_Click ) );
}
}

void AdjustDataGridViewSizing()
{
    dataGridView->AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode::AllCells;
    dataGridView->ColumnHeadersHeightSizeMode = DataGridViewColumnHeadersHeightSizeMode::AutoSize;
}

// Freeze the first row.
void Button4_Click( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    FreezeBand( dataGridView->Rows[ 0 ] );
}

void Button5_Click( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    FreezeBand( dataGridView->Columns[ 1 ] );
}

void FreezeBand( DataGridViewBand^ band )
{
    band->Frozen = true;
    DataGridViewCellStyle^ style = gcnew DataGridViewCellStyle;
    style->BackColor = Color::WhiteSmoke;
    band->DefaultCellStyle = style;
}

// Hide a band of cells.
void Button6_Click( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    DataGridViewBand^ band = dataGridView->Rows[ 3 ];
    band->Visible = false;
}

// Turn off user's ability to resize a column.
void Button7_Click( Object^ /*sender*/, EventArgs^ /*e*/ )
{
    DataGridViewBand^ band = dataGridView->Columns[ 0 ];
    band->Resizable = DataGridViewTriState::False;
}

```

```

// Make the entire DataGridView read only.
void Button8_Click( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    System::Collections::IEnumerator^ myEnum = dataGridView->Columns->GetEnumerator();
    while ( myEnum->MoveNext() )
    {
        DataGridViewBand^ band = safe_cast<DataGridViewBand^>(myEnum->Current);
        band->ReadOnly = true;
    }
}

void PostRowCreation()
{
    SetBandColor( dataGridView->Columns[ 0 ], Color::CadetBlue );
    SetBandColor( dataGridView->Rows[ 1 ], Color::Coral );
    SetBandColor( dataGridView->Columns[ 2 ], Color::DodgerBlue );
}

void SetBandColor( DataGridViewBand^ band, Color color )
{
    band->Tag = color;
}

// Color the bands by the value stored in their tag.
void Button9_Click( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    IEnumerator^ myEnum1 = dataGridView->Columns->GetEnumerator();
    while ( myEnum1->MoveNext() )
    {
        DataGridViewBand^ band = static_cast<DataGridViewBand^>(myEnum1->Current);
        if ( band->Tag != nullptr )
        {
            band->DefaultCellStyle->BackColor = *dynamic_cast<Color^>(band->Tag);
        }
    }

    IEnumerator^ myEnum2 = safe_cast<IEnumerable^>(dataGridView->Rows)->GetEnumerator();
    while ( myEnum2->MoveNext() )
    {
        DataGridViewBand^ band = safe_cast<DataGridViewBand^>(myEnum2->Current);
        if ( band->Tag != nullptr )
        {
            band->DefaultCellStyle->BackColor = *dynamic_cast<Color^>(band->Tag);
        }
    }
}

#pragma endregion

public:
    static void Main()
    {
        Application::Run( gcnew DataGridViewBandDemo );
    }
};

int main()
{
    DataGridViewBandDemo::Main();
}

```

```
using System.Drawing;
```

```

using System.Windows.Forms;
using System;

public class DataGridViewBandDemo : Form
{
    #region "form setup"
    public DataGridViewBandDemo()
    {
        InitializeComponent();

        AddButton(Button1, "Reset",
            new EventHandler(Button1_Click));
        AddButton(Button2, "Change Column 3 Header",
            new EventHandler(Button2_Click));
        AddButton(Button3, "Change Meatloaf Recipe",
            new EventHandler(Button3_Click));
        AddAdditionalButtons();

        InitializeDataGridView();
    }

    DataGridView dataGridView;
    Button Button1 = new Button();
    Button Button2 = new Button();
    Button Button3 = new Button();
    Button Button4 = new Button();
    Button Button5 = new Button();
    Button Button6 = new Button();
    Button Button7 = new Button();
    Button Button8 = new Button();
    Button Button9 = new Button();
    Button Button10 = new Button();
    FlowLayoutPanel FlowLayoutPanel1 = new FlowLayoutPanel();

    private void InitializeComponent()
    {
        FlowLayoutPanel1.Location = new Point(454, 0);
        FlowLayoutPanel1.AutoSize = true;
        FlowLayoutPanel1.FlowDirection = FlowDirection.TopDown;
        AutoSize = true;
        ClientSize = new System.Drawing.Size(614, 360);
        FlowLayoutPanel1.Name = "flowlayoutpanel";
        Controls.Add(thisFlowLayout1);
        Text = this.GetType().Name;
    }
    #endregion

    #region "setup DataGridView"

    private string thirdColumnHeader = "Main Ingredients";
    private string boringMeatloaf = "ground beef";
    private string boringMeatloafRanking = "*";
    private bool boringRecipe;
    private bool shortMode;

    private void InitializeDataGridView()
    {
        dataGridView = new System.Windows.Forms.DataGridView();
        Controls.Add(dataGridView);
        dataGridView.Size = new Size(300, 200);

        // Create an unbound DataGridView by declaring a
        // column count.
        dataGridView.ColumnCount = 4;
        AdjustDataGridViewSizing();

        // Set the column header style.
        DataGridViewCellStyle columnHeaderStyle =
            new DataGridViewCellStyle();

```

```

columnHeaderStyle.BackColor = Color.Aqua;
columnHeaderStyle.Font =
    new Font("Verdana", 10, FontStyle.Bold);
dataGridView.ColumnHeadersDefaultCellStyle =
    columnHeaderStyle;

// Set the column header names.
dataGridView.Columns[0].Name = "Recipe";
dataGridView.Columns[1].Name = "Category";
dataGridView.Columns[2].Name = thirdColumnHeader;
dataGridView.Columns[3].Name = "Rating";

// Populate the rows.
string[] row1 = new string[]{"Meatloaf",
    "Main Dish", boringMeatloaf, boringMeatloafRanking};
string[] row2 = new string[]{"Key Lime Pie",
    "Dessert", "lime juice, evaporated milk", "****"};
string[] row3 = new string[]{"Orange-Salsa Pork Chops",
    "Main Dish", "pork chops, salsa, orange juice", "****"};
string[] row4 = new string[]{"Black Bean and Rice Salad",
    "Salad", "black beans, brown rice", "****"};
string[] row5 = new string[]{"Chocolate Cheesecake",
    "Dessert", "cream cheese", "***"};
string[] row6 = new string[]{"Black Bean Dip", "Appetizer",
    "black beans, sour cream", "***"};
object[] rows = new object[] { row1, row2, row3, row4, row5, row6 };

foreach (string[] rowArray in rows)
{
    dataGridView.Rows.Add(rowArray);
}

PostRowCreation();

shortMode = false;
boringRecipe = true;
}

void AddButton(Button button, string buttonLabel,
    EventHandler handler)
{
    FlowLayoutPanel1.Controls.Add(button);
    button.TabIndex = FlowLayoutPanel1.Controls.Count;
    button.Text = buttonLabel;
    button.AutoSize = true;
    button.Click += handler;
}

// Reset columns to initial disorderly arrangement.
private void Button1_Click(object sender, System.EventArgs e)
{
    Controls.Remove(dataGridView);
    dataGridView.Dispose();
    InitializeDataGridView();
}

// Change the header in column three.
private void Button2_Click(object sender,
    System.EventArgs e)
{
    Toggle(ref shortMode);
    if (shortMode)
    { dataGridView.Columns[2].HeaderText = "S"; }
    else
    { dataGridView.Columns[2].HeaderText = thirdColumnHeader; }
}

private static void Toggle(ref bool toggleThis)

```

```

{
    toggleThis = !toggleThis;
}

// Change the meatloaf recipe.
private void Button3_Click(object sender,
    System.EventArgs e)
{
    Toggle(ref boringRecipe);
    if (boringRecipe)
    {
        SetMeatloaf(boringMeatloaf, boringMeatloafRanking);
    }
    else
    {
        string greatMeatloafRecipe =
            "1 lb. lean ground beef, " +
            "1/2 cup bread crumbs, 1/4 cup ketchup," +
            "1/3 tsp onion powder, " +
            "1 clove of garlic, 1/2 pack onion soup mix " +
            " dash of your favorite BBQ Sauce";
        SetMeatloaf(greatMeatloafRecipe, "***");
    }
}

private void SetMeatloaf(string recipe, string rating)
{
    dataGridView.Rows[0].Cells[2].Value = recipe;
    dataGridView.Rows[0].Cells[3].Value = rating;
}
#endregion

#region "demonstration code"
private void AddAdditionalButtons()
{
    AddButton(Button4, "Freeze First Row",
        new EventHandler(Button4_Click));
    AddButton(Button5, "Freeze Second Column",
        new EventHandler(Button5_Click));
    AddButton(Button6, "Hide Salad Row",
        new EventHandler(Button6_Click));
    AddButton(Button7, "Disable First Column Resizing",
        new EventHandler(Button7_Click));
    AddButton(Button8, "Make ReadOnly",
        new EventHandler(Button8_Click));
    AddButton(Button9, "Style Using Tag",
        new EventHandler(Button9_Click));
}
private void AdjustDataGridViewSizing()
{
    dataGridView.AutoSizeRowsMode =
        DataGridViewAutoSizeRowsMode.AllCells;
    dataGridView.ColumnHeadersHeightSizeMode =
        DataGridViewColumnHeadersHeightSizeMode.AutoSize;
}

// Freeze the first row.
private void Button4_Click(object sender, System.EventArgs e)
{
    FreezeBand(dataGridView.Rows[0]);
}

private void Button5_Click(object sender, System.EventArgs e)
{
    FreezeBand(dataGridView.Columns[1]);
}

```

```

private static void FreezeBand(DataGridViewBand band)
{
    band.Frozen = true;
    DataGridViewCellStyle style = new DataGridViewCellStyle();
    style.BackColor = Color.WhiteSmoke;
    band.DefaultCellStyle = style;
}

// Hide a band of cells.
private void Button6_Click(object sender, System.EventArgs e)
{
    DataGridViewBand band = dataGridView.Rows[3];
    band.Visible = false;
}

// Turn off user's ability to resize a column.
private void Button7_Click(object sender, EventArgs e)
{
    DataGridViewBand band = dataGridView.Columns[0];
    band.Resizable = DataGridViewTriState.False;
}

// Make the entire DataGridView read only.
private void Button8_Click(object sender, System.EventArgs e)
{
    foreach (DataGridViewBand band in dataGridView.Columns)
    {
        band.ReadOnly = true;
    }
}

private void PostRowCreation()
{
    SetBandColor(dataGridView.Columns[0], Color.CadetBlue);
    SetBandColor(dataGridView.Rows[1], Color.Coral);
    SetBandColor(dataGridView.Columns[2], Color.DodgerBlue);
}

private static void SetBandColor(DataGridViewBand band, Color color)
{
    band.Tag = color;
}

// Color the bands by the value stored in their tag.
private void Button9_Click(object sender, System.EventArgs e)
{
    foreach (DataGridViewBand band in dataGridView.Columns)
    {
        if (band.Tag != null)
        {
            band.DefaultCellStyle.BackColor = (Color)band.Tag;
        }
    }

    foreach (DataGridViewBand band in dataGridView.Rows)
    {
        if (band.Tag != null)
        {
            band.DefaultCellStyle.BackColor = (Color)band.Tag;
        }
    }
}
#endif

```

```

    [STAThread]
    public static void Main()
    {
        Application.Run(new DataGridViewBandDemo());
    }
}

```

```

Imports System.Windows.Forms
Imports System.Drawing

Public Class DataGridViewBandDemo
    Inherits Form

#Region "Form setup"
    Public Sub New()
        MyBase.New()
        InitializeComponent()

        AddButton(Button1, "Reset")
        AddButton(Button2, "Change Column 3 Header")
        AddButton(Button3, "Change Meatloaf Recipe")
        AddAdditionalButtons()
    End Sub

    Friend WithEvents dataGridView As DataGridView
    Friend WithEvents Button1 As Button = New Button()
    Friend WithEvents Button2 As Button = New Button()
    Friend WithEvents Button3 As Button = New Button()
    Friend WithEvents Button4 As Button = New Button()
    Friend WithEvents Button5 As Button = New Button()
    Friend WithEvents Button6 As Button = New Button()
    Friend WithEvents Button7 As Button = New Button()
    Friend WithEvents Button8 As Button = New Button()
    Friend WithEvents Button9 As Button = New Button()
    Friend WithEvents Button10 As Button = New Button()
    Friend WithEvents FlowLayoutPanel1 As FlowLayoutPanel _
        = New FlowLayoutPanel()

    Private Sub InitializeComponent()
        FlowLayoutPanel1.Location = New Point(454, 0)
        FlowLayoutPanel1.AutoSize = True
        FlowLayoutPanel1.FlowDirection = FlowDirection.TopDown
        FlowLayoutPanel1.Name = "flowlayoutpanel"
        ClientSize = New System.Drawing.Size(614, 360)
        Controls.Add(FlowLayoutPanel1)
        Text = Me.GetType().Name
        AutoSize = True
    End Sub
#End Region

#Region "setup DataGridView"
    Private thirdColumnHeader As String = "Main Ingredients"
    Private boringMeatloaf As String = "ground beef"
    Private boringMeatloafRanking As String = "*"
    Private boringRecipe As Boolean
    Private shortMode As Boolean

    Private Sub InitializeDataGridView(ByVal ignored As Object, _
        ByVal ignoredToo As EventArgs) Handles Me.Load
        dataGridView = New System.Windows.Forms.DataGridView
        Controls.Add(dataGridView)
        dataGridView.Size = New Size(300, 200)

        ' Create an unbound DataGridView by declaring a
        ' column count.
        dataGridView.ColumnCount = 4
        AdjustDataGridViewSizing()
    End Sub
#End Region

```

```

' Set the column header style.
Dim columnHeaderStyle As New DataGridViewCellStyle
columnHeaderStyle.BackColor = Color.Aqua
columnHeaderStyle.Font = _
    New Font("Verdana", 10, FontStyle.Bold)
dataGridView.ColumnHeadersDefaultCellStyle = _
    columnHeaderStyle

' Set the column header names.
dataGridView.Columns(0).Name = "Recipe"
dataGridView.Columns(1).Name = "Category"
dataGridView.Columns(2).Name = thirdColumnHeader
dataGridView.Columns(3).Name = "Rating"

' Populate the rows.
Dim row1 As String() = New String() _
    {"Meatloaf", "Main Dish", boringMeatloaf, _
    boringMeatloafRanking}
Dim row2 As String() = New String() _
    {"Key Lime Pie", "Dessert", _
    "lime juice, evaporated milk", _
    "****"}
Dim row3 As String() = New String() _
    {"Orange-Salsa Pork Chops", "Main Dish", _
    "pork chops, salsa, orange juice", "****"}
Dim row4 As String() = New String() _
    {"Black Bean and Rice Salad", "Salad", _
    "black beans, brown rice", _
    "****"}
Dim row5 As String() = New String() _
    {"Chocolate Cheesecake", "Dessert", "cream cheese", _
    "***"}
Dim row6 As String() = New String() _
    {"Black Bean Dip", "Appetizer", _
    "black beans, sour cream", _
    "****"}

Dim rows As Object() = New Object() {row1, row2, _
    row3, row4, row5, row6}

Dim rowArray As String()
For Each rowArray In rows
    dataGridView.Rows.Add(rowArray)
Next

PostRowCreation()

shortMode = False
boringRecipe = True
End Sub

Protected Sub AddButton(ByVal button As Button, _
    ByVal buttonLabel As String)

    FlowLayoutPanel1.Controls.Add(button)
    button.TabIndex = FlowLayoutPanel1.Controls.Count
    button.Text = buttonLabel
    button.AutoSize = True
End Sub

' Reset columns to initial disorderly arrangement.
Private Sub ResetToDisorder(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles Button1.Click
    Controls.Remove(dataGridView)
    dataGridView.Dispose()
    InitializeDataGridView(Nothing, Nothing)
End Sub

```

```

' Change the header in column three.
Private Sub Button2_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
Handles Button2.Click

    Toggle(shortMode)
    If shortMode Then dataGridView.Columns(2).HeaderText = _
        "S" _
    Else dataGridView.Columns(2).HeaderText = _
        thirdColumnHeader
End Sub

Private Shared Sub Toggle(ByRef toggleThis As Boolean)
    toggleThis = Not toggleThis
End Sub

' Change the meatloaf recipe.
Private Sub Button3_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
Handles Button3.Click

    Toggle(boringRecipe)
    If boringRecipe Then
        SetMeatloaf(boringMeatloaf, boringMeatloafRanking)
    Else
        Dim greatMeatloafRecipe As String = "1 lb. lean ground beef, " _
            & "1/2 cup bread crumbs, 1/4 cup ketchup," _
            & "1/3 tsp onion powder, " _
            & "1 clove of garlic, 1/2 pack onion soup mix " _
            & "dash of your favorite BBQ Sauce"
        SetMeatloaf(greatMeatloafRecipe, "***")
    End If
End Sub

Private Sub SetMeatloaf(ByVal recipe As String, _
    ByVal rating As String)

    dataGridView.Rows(0).Cells(2).Value = recipe
    dataGridView.Rows(0).Cells(3).Value = rating
End Sub
#End Region

#Region "demonstration code"
    Private Sub AddAdditionalButtons()
        AddButton(Button4, "Freeze First Row")
        AddButton(Button5, "Freeze Second Column")
        AddButton(Button6, "Hide Salad Row")
        AddButton(Button7, "Disable First Column Resizing")
        AddButton(Button8, "Make ReadOnly")
        AddButton(Button9, "Style Using Tag")
    End Sub

    Private Sub AdjustDataGridViewSizing()
        dataGridView.AutoSizeRowsMode = _
            DataGridViewAutoSizeRowsMode.AllCells
        dataGridView.ColumnHeadersHeightSizeMode = _
            DataGridViewColumnHeadersHeightSizeMode.AutoSize
    End Sub

    ' Freeze the first row.
    Private Sub Button4_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Button4.Click

        FreezeBand(dataGridView.Rows(0))
    End Sub

    Private Sub FreezeColumn(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Button5.Click

```

```

    FreezeBand(dataGridView.Columns(1))
End Sub

Private Shared Sub FreezeBand(ByVal band As DataGridViewBand)

    band.Frozen = True
    Dim style As DataGridViewCellStyle = New DataGridViewCellStyle()
    style.BackColor = Color.WhiteSmoke
    band.DefaultCellStyle = style

End Sub

' Hide a band of cells.
Private Sub Button6_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button6.Click

    Dim band As DataGridViewBand = dataGridView.Rows(3)
    band.Visible = False
End Sub

' Turn off user's ability to resize a column.
Private Sub Button7_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button7.Click

    Dim band As DataGridViewBand = dataGridView.Columns(0)
    band.Resizable = DataGridViewTriState.False
End Sub

' Make the entire DataGridView read only.
Private Sub Button8_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button8.Click

    For Each band As DataGridViewBand In dataGridView.Columns
        band.ReadOnly = True
    Next
End Sub

Private Sub PostRowCreation()
    SetBandColor(dataGridView.Columns(0), Color.CadetBlue)
    SetBandColor(dataGridView.Rows(1), Color.Coral)
    SetBandColor(dataGridView.Columns(2), Color.DodgerBlue)
End Sub

Private Shared Sub SetBandColor(ByVal band As DataGridViewBand, _
    ByVal color As Color)
    band.Tag = color
End Sub

' Color the bands by the value stored in their tag.
Private Sub Button9_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button9.Click

    For Each band As DataGridViewBand In dataGridView.Columns
        If band.Tag IsNot Nothing Then
            band.DefaultCellStyle.BackColor = _
                CType(band.Tag, Color)
        End If
    Next

    For Each band As DataGridViewBand In dataGridView.Rows
        If band.Tag IsNot Nothing Then
            band.DefaultCellStyle.BackColor = _
                CType(band.Tag, Color)
        End If
    Next
End Sub

#End Region

```

```
<STAThreadAttribute()> _
Public Shared Sub Main()
    Application.Run(New DataGridViewBandDemo())
End Sub
End Class
```

Compiling the Code

This example requires:

- References to the System, System.Drawing, and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[DataGridView](#)

[DataGridViewBand](#)

[DataGridViewRow](#)

[DataGridViewColumn](#)

[Programming with Cells, Rows, and Columns in the Windows Forms DataGridView Control](#)

How to: Manipulate Rows in the Windows Forms DataGridView Control

5/4/2018 • 12 min to read • [Edit Online](#)

The following code example shows the various ways to manipulate `DataGridView` rows using properties of the `DataGridViewRow` class.

Example

```
#using <System.Drawing.dll>
#using <System.dll>
#using <system.windows.forms.dll>

using namespace System;
using namespace System::Windows::Forms;
using namespace System::Drawing;
public ref class DataGridViewRowDemo: public Form
{
private:

#pragma region S " form setup "

public:
    DataGridViewRowDemo()
    {
        Button1 = gcnew Button;
        Button2 = gcnew Button;
        Button3 = gcnew Button;
        Button4 = gcnew Button;
        Button5 = gcnew Button;
        Button6 = gcnew Button;
        Button7 = gcnew Button;
        Button8 = gcnew Button;
        Button9 = gcnew Button;
        Button10 = gcnew Button;
        FlowLayoutPanel1 = gcnew FlowLayoutPanel;
        thirdColumnHeader = L"Main Ingredients";
        boringMeatloaf = L"ground beef";
        boringMeatloafRanking = L"*";
        ratingColumn = 3;
        AddButton( Button1, L"Reset", gcnew EventHandler( this, &DataGridViewRowDemo::Button1_Click ) );
        AddButton( Button2, L"Change Column 3 Header", gcnew EventHandler( this,
&DataGridViewRowDemo::Button2_Click ) );
        AddButton( Button3, L"Change Meatloaf Recipe", gcnew EventHandler( this,
&DataGridViewRowDemo::Button3_Click ) );
        InitializeComponent();
        InitializeDataGridView();
        AddAdditionalButtons();
    }

private:
    DataGridView^ dataGridView;
    Button^ Button1;
    Button^ Button2;
    Button^ Button3;
    Button^ Button4;
    Button^ Button5;
    Button^ Button6;
    Button^ Button7;
```

```

button^ Button7;
Button^ Button8;
Button^ Button9;
Button^ Button10;
FlowLayoutPanel^ FlowLayoutPanel1;
void InitializeComponent()
{
    FlowLayoutPanel1->Location = Point(454,0);
    FlowLayoutPanel1->AutoSize = true;
    FlowLayoutPanel1->FlowDirection = FlowDirection::TopDown;
    AutoSize = true;
    ClientSize = System::Drawing::Size( 614, 360 );
    FlowLayoutPanel1->Name = L"flowlayoutpanel";
    Controls->Add( this->FlowLayoutPanel1 );
    Text = this->GetType()->Name;
}

#pragma endregion
#pragma region S " setup DataGridView "
String^ thirdColumnHeader;
String^ boringMeatloaf;
String^ boringMeatloafRanking;
bool boringRecipe;
bool shortMode;
void InitializeDataGridView()
{
    dataGridView = gcnew System::Windows::Forms::DataGridView;
    Controls->Add( dataGridView );
    dataGridView->Size = System::Drawing::Size( 300, 200 );

    // Create an unbound DataGridView by declaring a
    // column count.
    dataGridView->ColumnCount = 4;
    dataGridView->ColumnHeadersVisible = true;
    AdjustDataGridViewSizing();

    // Set the column header style.
    DataGridViewCellStyle^ columnHeaderStyle = gcnew DataGridViewCellStyle;
    columnHeaderStyle->BackColor = Color::Aqua;
    columnHeaderStyle->Font = gcnew System::Drawing::Font( L"Verdana",10,FontStyle::Bold );
    dataGridView->ColumnHeadersDefaultCellStyle = columnHeaderStyle;

    // Set the column header names.
    dataGridView->Columns[ 0 ]->Name = L"Recipe";
    dataGridView->Columns[ 1 ]->Name = L"Category";
    dataGridView->Columns[ 2 ]->Name = thirdColumnHeader;
    dataGridView->Columns[ 3 ]->Name = L"Rating";

    // Populate the rows.
    array<String^>^row1 = gcnew array<String^>{
        L"Meatloaf",L"Main Dish",boringMeatloaf,boringMeatloafRanking
    };
    array<String^>^row2 = gcnew array<String^>{
        L"Key Lime Pie",L"Dessert",L"lime juice, evaporated milk",L"****"
    };
    array<String^>^row3 = gcnew array<String^>{
        L"Orange-Salsa Pork Chops",L"Main Dish",L"pork chops, salsa, orange juice",L"****"
    };
    array<String^>^row4 = gcnew array<String^>{
        L"Black Bean and Rice Salad",L"Salad",L"black beans, brown rice",L"****"
    };
    array<String^>^row5 = gcnew array<String^>{
        L"Chocolate Cheesecake",L"Dessert",L"cream cheese",L"***"
    };
    array<String^>^row6 = gcnew array<String^>{
        L"Black Bean Dip",L"Appetizer",L"black beans, sour cream",L"***"
    };
    array<Object^>^rows = gcnew array<Object^>{

```

```

    row1, row2, row3, row4, row5, row6
};

System::Collections::IEnumerator^ myEnum = rows->GetEnumerator();
while ( myEnum->MoveNext() )
{
    array<String^>^rowArray = safe_cast<array<String^>^>(myEnum->Current);
    dataGridView->Rows->Add( rowArray );
}

shortMode = false;
boringRecipe = true;
}

void AddButton( Button^ button, String^ buttonLabel, EventHandler^ handler )
{
    FlowLayoutPanel1->Controls->Add( button );
    button->TabIndex = FlowLayoutPanel1->Controls->Count;
    button->Text = buttonLabel;
    button->AutoSize = true;
    button->Click += handler;
}

// Reset columns to initial disorderly arrangement.
void Button1_Click( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    Controls->Remove( dataGridView );
    dataGridView->~DataGridView();
    InitializeDataGridView();
}

// Change column 3 header.
void Button2_Click( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    Toggle( &shortMode );
    if ( shortMode )
    {
        dataGridView->Columns[ 2 ]->HeaderText = L"S";
    }
    else
    {
        dataGridView->Columns[ 2 ]->HeaderText = thirdColumnHeader;
    }
}

void Toggle( interior_ptr<Boolean> toggleThis )
{
    *toggleThis = ! *toggleThis;
}

// Change meatloaf recipe.
void Button3_Click( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    Toggle( &boringRecipe );
    if ( boringRecipe )
    {
        SetMeatloaf( boringMeatloaf, boringMeatloafRanking );
    }
    else
    {
        String^ greatMeatloafRecipe = L"1 lb. lean ground beef, "
        L"1/2 cup bread crumbs, 1/4 cup ketchup,"
        L"1/3 tsp onion powder,"
        L"1 clove of garlic, 1/2 pack onion soup mix "
        L" dash of your favorite BBQ Sauce";
        SetMeatloaf( greatMeatloafRecipe, L"***" );
    }
}

```

```

}

void SetMeatloaf( String^ recipe, String^ rating )
{
    dataGridView->Rows[ 0 ]->Cells[ 2 ]->Value = recipe;
    dataGridView->Rows[ 0 ]->Cells[ 3 ]->Value = rating;
}

#pragma endregion
#pragma region S " demonstration code "
void AddAdditionalButtons()
{
    AddButton( Button4, L"Set Row Two Minimum Height", gcnew EventHandler( this,
&DataGridViewRowDemo::Button4_Click ) );
    AddButton( Button5, L"Set Row One Height", gcnew EventHandler( this, &DataGridViewRowDemo::Button5_Click
) );
    AddButton( Button6, L"Label Rows", gcnew EventHandler( this, &DataGridViewRowDemo::Button6_Click ) );
    AddButton( Button7, L"Turn on Extra Edge", gcnew EventHandler( this, &DataGridViewRowDemo::Button7_Click
) );
    AddButton( Button8, L"Give Cheesecake an Excellent Rating", gcnew EventHandler( this,
&DataGridViewRowDemo::Button8_Click ) );
}

void AdjustDataGridViewSizing()
{
    dataGridView->ColumnHeadersHeightSizeMode = DataGridViewColumnHeadersHeightSizeMode::AutoSize;
    dataGridView->Columns[ ratingColumn ]->Width = 50;
}

// Set minimum height.
void Button4_Click( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    int secondRow = 1;
    DataGridViewRow^ row = dataGridView->Rows[ secondRow ];
    row->MinimumHeight = 40;
}

// Set height.
void Button5_Click( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    DataGridViewRow^ row = dataGridView->Rows[ 0 ];
    row->Height = 15;
}

// Set row labels.
void Button6_Click( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    int lineNumber = 1;
    System::Collections::IEnumerator^ myEnum = safe_cast<System::Collections::IEnumerable^>(dataGridView-
>Rows)->GetEnumerator();
    while ( myEnum->MoveNext() )
    {
        DataGridViewRow^ row = safe_cast<DataGridViewRow^>(myEnum->Current);
        if ( row->IsNewRow )
            continue;
        row->HeaderCell->Value = String::Format( L"Row {0}", lineNumber );

        lineNumber = lineNumber + 1;
    }
}

dataGridView->AutoResizeRowHeadersWidth( DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders );
}

```

```

// Set a thick horizontal edge.
void Button7_Click( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    int secondRow = 1;
    int edgeThickness = 3;
    DataGridViewRow^ row = dataGridView->Rows[ secondRow ];
    row->DividerHeight = 10;
}

// Give cheesecake excellent rating.
void Button8_Click( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    UpdateStars( dataGridView->Rows[ 4 ], L"*****" );
}

int ratingColumn;
void UpdateStars( DataGridViewRow^ row, String^ stars )
{
    row->Cells[ ratingColumn ]->Value = stars;

    // Resize the column width to account for the new value.
    row->DataGridView->AutoResizeColumn( ratingColumn, DataGridViewAutoSizeColumnsMode::DisplayedCells );
}

#pragma endregion

public:
    static void Main()
    {
        Application::Run( gcnew DataGridViewRowDemo );
    }

};

int main()
{
    DataGridViewRowDemo::Main();
}

```

```

using System.Windows.Forms;
using System;
using System.Drawing;

public class DataGridViewRowDemo : Form
{
    #region "form setup"
    public DataGridViewRowDemo()
    {
        InitializeComponent();

        AddButton(Button1, "Reset",
            new EventHandler(Button1_Click));
        AddButton(Button2, "Change Column 3 Header",
            new EventHandler(Button2_Click));
        AddButton(Button3, "Change Meatloaf Recipe",
            new EventHandler(Button3_Click));
        AddAdditionalButtons();

        InitializeDataGridView();
    }

    private DataGridView dataGridView;
    private Button Button1 = new Button();
    private Button Button2 = new Button();

```

```

private Button Button3 = new Button();
private Button Button4 = new Button();
private Button Button5 = new Button();
private Button Button6 = new Button();
private Button Button7 = new Button();
private Button Button8 = new Button();
private Button Button9 = new Button();
private Button Button10 = new Button();
private FlowLayoutPanel FlowLayoutPanel1 = new FlowLayoutPanel();

private void InitializeComponent()
{
    FlowLayoutPanel1.Location = new Point(454, 0);
    FlowLayoutPanel1.AutoSize = true;
    FlowLayoutPanel1.FlowDirection = FlowDirection.TopDown;
    AutoSize = true;
    ClientSize = new System.Drawing.Size(614, 360);
    FlowLayoutPanel1.Name = "flowlayoutpanel";
    Controls.Add(this.flowLayoutPanel1);
    Text = this.GetType().Name;
}
#endregion

#region "setup DataGridView"

private string thirdColumnHeader = "Main Ingredients";
private string boringMeatloaf = "ground beef";
private string boringMeatloafRanking = "*";
private bool boringRecipe;
private bool shortMode;

private void InitializeDataGridView()
{
    dataGridView = new System.Windows.Forms.DataGridView();
    Controls.Add(dataGridView);
    dataGridView.Size = new Size(300, 200);

    // Create an unbound DataGridView by declaring a
    // column count.
    dataGridView.ColumnCount = 4;
    dataGridView.ColumnHeadersVisible = true;
    AdjustDataGridViewSizing();

    // Set the column header style.
    DataGridViewCellStyle columnHeaderStyle =
        new DataGridViewCellStyle();
    columnHeaderStyle.BackColor = Color.Aqua;
    columnHeaderStyle.Font =
        new Font("Verdana", 10, FontStyle.Bold);
    dataGridView.ColumnHeadersDefaultCellStyle =
        columnHeaderStyle;

    // Set the column header names.
    dataGridView.Columns[0].Name = "Recipe";
    dataGridView.Columns[1].Name = "Category";
    dataGridView.Columns[2].Name = thirdColumnHeader;
    dataGridView.Columns[3].Name = "Rating";

    // Populate the rows.
    string[] row1 = new string[]{"Meatloaf",
        "Main Dish", boringMeatloaf, boringMeatloafRanking};
    string[] row2 = new string[]{"Key Lime Pie",
        "Dessert", "lime juice, evaporated milk", "****"};
    string[] row3 = new string[]{"Orange-Salsa Pork Chops",
        "Main Dish", "pork chops, salsa, orange juice", "****"};
    string[] row4 = new string[]{"Black Bean and Rice Salad",
        "Salad", "black beans, brown rice", "****"};
    string[] row5 = new string[]{"Chocolate Cheesecake",
        "Dessert", "cream cheese", "***"};
}

```

```

        string[] row6 = new string[]{"Black Bean Dip", "Appetizer",
                                      "black beans, sour cream", "***"};
        object[] rows = new object[] { row1, row2, row3, row4, row5, row6 };

        foreach (string[] rowArray in rows)
        {
            dataGridView.Rows.Add(rowArray);
        }

        shortMode = false;
        boringRecipe = true;
    }

    private void AddButton(Button button, string buttonLabel,
                          EventHandler handler)
    {
        FlowLayoutPanel1.Controls.Add(button);
        button.TabIndex = FlowLayoutPanel1.Controls.Count;
        button.Text = buttonLabel;
        button.AutoSize = true;
        button.Click += handler;
    }

    // Reset columns to initial disorderly arrangement.
    private void Button1_Click(object sender, System.EventArgs e)
    {
        Controls.Remove(dataGridView);
        dataGridView.Dispose();
        InitializeDataGridView();
    }

    // Change column 3 header.
    private void Button2_Click(object sender,
                               System.EventArgs e)
    {
        Toggle(ref shortMode);
        if (shortMode)
        { dataGridView.Columns[2].HeaderText = "S"; }
        else
        { dataGridView.Columns[2].HeaderText = thirdColumnHeader; }
    }

    private static void Toggle(ref bool toggleThis)
    {
        toggleThis = !toggleThis;
    }

    // Change meatloaf recipe.
    private void Button3_Click(object sender,
                               System.EventArgs e)
    {
        Toggle(ref boringRecipe);
        if (boringRecipe)
        {
            SetMeatloaf(boringMeatloaf, boringMeatloafRanking);
        }
        else
        {
            string greatMeatloafRecipe =
                "1 lb. lean ground beef, " +
                "1/2 cup bread crumbs, 1/4 cup ketchup," +
                "1/3 tsp onion powder, " +
                "1 clove of garlic, 1/2 pack onion soup mix " +
                "dash of your favorite BBQ Sauce";
            SetMeatloaf(greatMeatloafRecipe, "***");
        }
    }

    private void SetMeatloaf(string recipe, string rating)

```

```

{
    dataGridView.Rows[0].Cells[2].Value = recipe;
    dataGridView.Rows[0].Cells[3].Value = rating;
}
#endregion

#region "demonstration code"
private void AddAdditionalButtons()
{
    AddButton(Button4, "Set Row Two Minimum Height",
        new EventHandler(Button4_Click));
    AddButton(Button5, "Set Row One Height",
        new EventHandler(Button5_Click));
    AddButton(Button6, "Label Rows",
        new EventHandler(Button6_Click));
    AddButton(Button7, "Turn on Extra Edge",
        new EventHandler(Button7_Click));
    AddButton(Button8, "Give Cheesecake an Excellent Rating",
        new EventHandler(Button8_Click));
}

private void AdjustDataGridViewSizing()
{
    dataGridView.ColumnHeadersHeightSizeMode =
        DataGridViewColumnHeadersHeightSizeMode.AutoSize;
    dataGridView.Columns[ratingColumn].Width = 50;
}

// Set minimum height.
private void Button4_Click(object sender, System.EventArgs e)
{
    int secondRow = 1;
    DataGridViewRow row = dataGridView.Rows[secondRow];
    row.MinimumHeight = 40;
}

// Set height.
private void Button5_Click(object sender, System.EventArgs e)
{
    DataGridViewRow row = dataGridView.Rows[0];
    row.Height = 15;
}

// Set row labels.
private void Button6_Click(object sender, System.EventArgs e)
{
    int rowCount = 1;
    foreach (DataGridViewRow row in dataGridView.Rows)
    {
        if (row.IsNewRow) continue;
        row.HeaderCell.Value = "Row " + rowCount;
        rowCount = rowCount + 1;
    }
    dataGridView.AutoResizeRowHeadersWidth(
        DataGridViewRowHeadersWidthSizeMode.AutoSizeToAllHeaders);
}

// Set a thick horizontal edge.
private void Button7_Click(object sender,
    System.EventArgs e)
{
    int secondRow = 1;
    DataGridViewRow row = dataGridView.Rows[secondRow];
    row.DividerHeight = 10;
}

```

```

// Give cheesecake excellent rating.
private void Button8_Click(object sender,
    System.EventArgs e)
{
    UpdateStars(dataGridView.Rows[4], "*****");
}

int ratingColumn = 3;

private void UpdateStars(DataGridViewRow row, string stars)
{
    row.Cells[ratingColumn].Value = stars;

    // Resize the column width to account for the new value.
    row.DataGridView.AutoResizeColumn(ratingColumn,
        DataGridViewAutoSizeColumnMode.DisplayedCells);
}
#endregion

[STAThreadAttribute()]
public static void Main()
{
    Application.Run(new DataGridViewRowDemo());
}
}

```

```

Imports System.Windows.Forms
Imports System.Drawing

Public Class DataGridViewRowDemo
    Inherits Form

#Region "Form setup"
    Public Sub New()
        MyBase.New()
        InitializeComponent()

        AddButton(Button1, "Reset")
        AddButton(Button2, "Change Column 3 Header")
        AddButton(Button3, "Change Meatloaf Recipe")
        AddAdditionalButtons()
    End Sub

    Friend WithEvents dataGridView As DataGridView
    Friend WithEvents Button1 As Button = New Button()
    Friend WithEvents Button2 As Button = New Button()
    Friend WithEvents Button3 As Button = New Button()
    Friend WithEvents Button4 As Button = New Button()
    Friend WithEvents Button5 As Button = New Button()
    Friend WithEvents Button6 As Button = New Button()
    Friend WithEvents Button7 As Button = New Button()
    Friend WithEvents Button8 As Button = New Button()
    Friend WithEvents Button9 As Button = New Button()
    Friend WithEvents Button10 As Button = New Button()
    Friend WithEvents FlowLayoutPanel1 As FlowLayoutPanel _
        = New FlowLayoutPanel()

    Private Sub InitializeComponent()
        FlowLayoutPanel1.Location = New Point(454, 0)
        FlowLayoutPanel1.AutoSize = True
        FlowLayoutPanel1.FlowDirection = FlowDirection.TopDown
        FlowLayoutPanel1.Name = "flowlayoutpanel"
        ClientSize = New System.Drawing.Size(614, 360)
        Controls.Add(FlowLayoutPanel1)
        Text = Me.GetType().Name
        AutoSize = True
    End Sub

```

```

End Sub
#End Region

#Region "setup DataGridView"
Private thirdColumnHeader As String = "Main Ingredients"
Private boringMeatloaf As String = "ground beef"
Private boringMeatloafRanking As String = "*"
Private boringRecipe As Boolean
Private shortMode As Boolean

Private Sub InitializeDataGridView(ByVal ignored As Object, _
ByVal ignoredToo As EventArgs) Handles Me.Load

    dataGridView = New System.Windows.Forms.DataGridView
    Controls.Add(dataGridView)
    dataGridView.Size = New Size(300, 200)

    ' Create an unbound DataGridView by declaring a
    ' column count.
    dataGridView.ColumnCount = 4
    dataGridView.ColumnHeadersVisible = True
    AdjustDataGridViewSizing()

    ' Set the column header style.
    Dim columnHeaderStyle As New DataGridViewCellStyle
    columnHeaderStyle.BackColor = Color.Aqua
    columnHeaderStyle.Font = _
        New Font("Verdana", 10, FontStyle.Bold)
    dataGridView.ColumnHeadersDefaultCellStyle = _
        columnHeaderStyle

    ' Set the column header names.
    dataGridView.Columns(0).Name = "Recipe"
    dataGridView.Columns(1).Name = "Category"
    dataGridView.Columns(2).Name = thirdColumnHeader
    dataGridView.Columns(3).Name = "Rating"

    ' Populate the rows.
    Dim row1 As String() = New String() _
        {"Meatloaf", "Main Dish", boringMeatloaf, _
        boringMeatloafRanking}
    Dim row2 As String() = New String() _
        {"Key Lime Pie", "Dessert", _
        "lime juice, evaporated milk", _
        "*****"}
    Dim row3 As String() = New String() _
        {"Orange-Salsa Pork Chops", "Main Dish", _
        "pork chops, salsa, orange juice", "*****"}
    Dim row4 As String() = New String() _
        {"Black Bean and Rice Salad", "Salad", _
        "black beans, brown rice", _
        "*****"}
    Dim row5 As String() = New String() _
        {"Chocolate Cheesecake", "Dessert", "cream cheese", _
        "****"}
    Dim row6 As String() = New String() _
        {"Black Bean Dip", "Appetizer", _
        "black beans, sour cream", _
        "****"}

    Dim rows As Object() = New Object() {row1, row2, _
        row3, row4, row5, row6}

    Dim rowArray As String()
    For Each rowArray In rows
        dataGridView.Rows.Add(rowArray)
    Next

    shortMode = False
    boringRecipe = True

```

```

    End Sub

Private Sub AddButton(ByVal button As Button, _
    ByVal buttonLabel As String)

    FlowLayoutPanel1.Controls.Add(button)
    button.TabIndex = FlowLayoutPanel1.Controls.Count
    button.Text = buttonLabel
    button.AutoSize = True
End Sub

' Reset columns to initial disorderly arrangement.
Private Sub Button1_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles Button1.Click
    Controls.Remove(dataGridView)
    dataGridView.Dispose()
    InitializeDataGridView(Nothing, Nothing)
End Sub

' Change column 3 header.
Private Sub Button2_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles Button2.Click

    Toggle(shortMode)
    If shortMode Then dataGridView.Columns(2).HeaderText = _
        "S" _
    Else dataGridView.Columns(2).HeaderText = _
        thirdColumnHeader
End Sub

Private Shared Sub Toggle(ByRef toggleThis As Boolean)
    toggleThis = Not toggleThis
End Sub

' Change meatloaf recipe.
Private Sub Button3_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles Button3.Click

    Toggle(boringRecipe)
    If boringRecipe Then
        SetMeatloaf(boringMeatloaf, boringMeatloafRanking)
    Else
        Dim greatMeatloafRecipe As String = "1 lb. lean ground beef, " _
            & "1/2 cup bread crumbs, 1/4 cup ketchup," _
            & "1/3 tsp onion powder, " _
            & "1 clove of garlic, 1/2 pack onion soup mix " _
            & "dash of your favorite BBQ Sauce"
        SetMeatloaf(greatMeatloafRecipe, "***")
    End If
End Sub

Private Sub SetMeatloaf(ByVal recipe As String, _
    ByVal rating As String)

    dataGridView.Rows(0).Cells(2).Value = recipe
    dataGridView.Rows(0).Cells(3).Value = rating
End Sub

#End Region

#Region "demonstration code"
Private Sub AddAdditionalButtons()
    AddButton(Button4, "Set Row Two Minimum Height")
    AddButton(Button5, "Set Row One Height")
    AddButton(Button6, "Label Rows")
    AddButton(Button7, "Turn on Extra Edge")
    AddButton(Button8, "Give chocolate an Excellent Rating")
End Sub

```

```

        ' Add button 4 buttons, give cheesecake an excellent rating.
End Sub

Private Sub AdjustDataGridViewSizing()
    dataGridView.ColumnHeadersHeightSizeMode = _
        DataGridViewColumnHeadersHeightSizeMode.AutoSize
    dataGridView.Columns(ratingColumn).Width = 50
End Sub

' Set minimum height.
Private Sub Button4_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button4.Click

    Dim secondRow As Integer = 1
    Dim row As DataGridViewRow = dataGridView.Rows(secondRow)
    row.MinimumHeight = 40
End Sub

' Set height.
Private Sub Button5_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button5.Click

    Dim row As DataGridViewRow = dataGridView.Rows(0)
    row.Height = 15
End Sub

' Set row labels.
Private Sub Button6_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button6.Click

    Dim rowNumber As Integer = 1
    For Each row As DataGridViewRow In dataGridView.Rows
        If row.IsNewRow Then Continue For
        row.HeaderCell.Value = "Row " & rowNumber
        rowNumber = rowNumber + 1
    Next
    dataGridView.AutoResizeRowHeadersWidth( _
        DataGridViewRowHeadersWidthSizeMode.AutoSizeToAllHeaders)
End Sub

' Set a thick horizontal edge.
Private Sub Button7_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button7.Click

    Dim secondRow As Integer = 1
    Dim row As DataGridViewRow = dataGridView.Rows(secondRow)
    row.DividerHeight = 10
End Sub

' Give cheesecake excellent rating.
Private Sub Button8_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button8.Click

    UpdateStars(dataGridView.Rows(4), "*****")
End Sub

Private ratingColumn As Integer = 3

Private Sub UpdateStars(ByVal row As DataGridViewRow, _
    ByVal stars As String)

    row.Cells(ratingColumn).Value = stars

    ' Resize the column width to account for the new value.
    row.DataGridView.AutoResizeColumn(ratingColumn, _
        DataGridViewAutoSizeColumnMode.DisplayedCells)
End Sub

```

```
#End Region

<STAThreadAttribute()> _
Public Shared Sub Main()
    Application.Run(New DataGridViewRowDemo())
End Sub

End Class
```

Compiling the Code

This example requires:

- References to the System, System.Drawing, and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[DataGridView](#)

[DataGridViewBand](#)

[DataGridViewRow](#)

[DataGridViewColumn](#)

[Programming with Cells, Rows, and Columns in the Windows Forms DataGridView Control](#)

How to: Manipulate Columns in the Windows Forms DataGridView Control

5/4/2018 • 17 min to read • [Edit Online](#)

The following code example shows the various ways to manipulate `DataGridView` columns using properties of the `DataGridViewColumn` class.

Example

```
#using <System.Drawing.dll>
#using <System.dll>
#using <system.windows.forms.dll>

using namespace System;
using namespace System::Windows::Forms;
using namespace System::Drawing;
using namespace System::Collections;
public ref class DataGridViewColumnDemo: public Form
{
private:

#pragma region S "set up form"

public:
    DataGridViewColumnDemo()
    {
        Button1 = gcnew Button;
        Button2 = gcnew Button;
        Button3 = gcnew Button;
        Button4 = gcnew Button;
        Button5 = gcnew Button;
        Button6 = gcnew Button;
        Button7 = gcnew Button;
        Button8 = gcnew Button;
        Button9 = gcnew Button;
        Button10 = gcnew Button;
        FlowLayoutPanel1 = gcnew FlowLayoutPanel;
        thirdColumnHeader = L"Main Ingredients";
        boringMeatloaf = L"ground beef";
        boringMeatloafRanking = L"*";
        toolStripItem1 = gcnew ToolStripMenuItem;
        InitializeComponent();
        AddButton( Button1, L"Reset", gcnew EventHandler( this, &DataGridViewColumnDemo::ResetToDisorder ) );
        AddButton( Button2, L"Change Column 3 Header", gcnew EventHandler( this,
&DataGridViewColumnDemo::ChangeColumn3Header ) );
        AddButton( Button3, L"Change Meatloaf Recipe", gcnew EventHandler( this,
&DataGridViewColumnDemo::ChangeMeatloafRecipe ) );
        AddAdditionalButtons();
        InitializeDataGridView();
    }

    DataGridView^ dataGridView;
    Button^ Button1;
    Button^ Button2;
    Button^ Button3;
    Button^ Button4;
    Button^ Button5;
    Button^ Button6;
    Button^ Button7;
    Button^ Button8;
}
```

```

        Button^ Button8,
        Button^ Button9;
        Button^ Button10;
        FlowLayoutPanel^ FlowLayoutPanel1;

private:
    void InitializeComponent()
    {
        FlowLayoutPanel1->Location = Point(454,0);
        FlowLayoutPanel1->AutoSize = true;
        FlowLayoutPanel1->FlowDirection = FlowDirection::TopDown;
        AutoSize = true;
        ClientSize = System::Drawing::Size( 614, 360 );
        FlowLayoutPanel1->Name = L"flowlayoutpanel";
        Controls->Add( this->FlowLayoutPanel1 );
        Text = this->GetType()->Name;
    }

#pragma endregion
#pragma region S " set up DataGridView "
String^ thirdColumnHeader;
String^ boringMeatloaf;
String^ boringMeatloafRanking;
bool boringRecipe;
bool shortMode;
void InitializeDataGridView()
{
    dataGridView = gcnew System::Windows::Forms::DataGridView;
    Controls->Add( dataGridView );
    dataGridView->Size = System::Drawing::Size( 300, 200 );

    // Create an unbound DataGridView by declaring a
    // column count.
    dataGridView->ColumnCount = 4;
    AdjustDataGridViewSizing();

    // Set the column header style.
    DataGridViewCellStyle^ columnHeaderStyle = gcnew DataGridViewCellStyle;
    columnHeaderStyle->BackColor = Color::Aqua;
    columnHeaderStyle->Font = gcnew System::Drawing::Font( L"Verdana",10,FontStyle::Bold );
    dataGridView->ColumnHeadersDefaultCellStyle = columnHeaderStyle;

    // Set the column header names.
    dataGridView->Columns[ 0 ]->Name = L"Recipe";
    dataGridView->Columns[ 1 ]->Name = L"Category";
    dataGridView->Columns[ 2 ]->Name = thirdColumnHeader;
    dataGridView->Columns[ 3 ]->Name = L"Rating";
    criteriaLabel = L"Column 3 sizing criteria: ";
    PostColumnCreation();

    // Populate the rows.
    array<String^>^row1 = gcnew array<String^>{
        L"Meatloaf",L"Main Dish",boringMeatloaf,boringMeatloafRanking
    };
    array<String^>^row2 = gcnew array<String^>{
        L"Key Lime Pie",L"Dessert",L"lime juice, evaporated milk",L"****"
    };
    array<String^>^row3 = gcnew array<String^>{
        L"Orange-Salsa Pork Chops",L"Main Dish",L"pork chops, salsa, orange juice",L"****"
    };
    array<String^>^row4 = gcnew array<String^>{
        L"Black Bean and Rice Salad",L"Salad",L"black beans, brown rice",L"****"
    };
    array<String^>^row5 = gcnew array<String^>{
        L"Chocolate Cheesecake",L"Dessert",L"cream cheese",L"***"
    };
    array<String^>^row6 = gcnew array<String^>{
        L"Black Bean Dip",L"Appetizer",L"black beans, sour cream",L"***"
    };
}

```

```

};

array<Object^>^rows = gcnew array<Object^>{
    row1, row2, row3, row4, row5, row6
};
System::Collections::IEnumerator^ myEnum = rows->GetEnumerator();
while ( myEnum->MoveNext() )
{
    array<String^>^rowArray = safe_cast<array<String^>^>(myEnum->Current);
    dataGridView->Rows->Add( rowArray );
}

shortMode = false;
boringRecipe = true;
}

void AddButton( Button^ button, String^ buttonLabel, EventHandler^ handler )
{
    FlowLayoutPanel1->Controls->Add( button );
    button->TabIndex = FlowLayoutPanel1->Controls->Count;
    button->Text = buttonLabel;
    button->AutoSize = true;
    button->Click += handler;
}

void ResetToDisorder( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    Controls->Remove( dataGridView );
    dataGridView->~DataGridView();
    InitializeDataGridView();
}

void ChangeColumn3Header( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    Toggle( &shortMode );
    if ( shortMode )
    {
        dataGridView->Columns[ 2 ]->HeaderText = L"S";
    }
    else
    {
        dataGridView->Columns[ 2 ]->HeaderText = thirdColumnHeader;
    }
}

void Toggle( interior_ptr<Boolean> toggleThis )
{
    *toggleThis = ! *toggleThis;
}

void ChangeMeatloafRecipe( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    Toggle( &boringRecipe );
    if ( boringRecipe )
    {
        SetMeatloaf( boringMeatloaf, boringMeatloafRanking );
    }
    else
    {
        String^ greatMeatloafRecipe = L"1 lb. lean ground beef, "
        L"1/2 cup bread crumbs, 1/4 cup ketchup,"
        L"1/3 tsp onion powder, "
        L"1 clove of garlic, 1/2 pack onion soup mix "
        L" dash of your favorite BBQ Sauce";
        SetMeatloaf( greatMeatloafRecipe, L"***" );
    }
}

void SetMeatloaf( String^ recipe, String^ rating )
{
}

```

```

        dataGridView->Rows[ 0 ]->Cells[ 2 ]->Value = recipe;
        dataGridView->Rows[ 0 ]->Cells[ 3 ]->Value = rating;
    }

#pragma endregion

public:
    static void Main()
{
    Application::Run( gcnew DataGridViewColumnDemo );
}

#pragma region S " demonstration code "

private:
    void PostColumnCreation()
{
    AddContextLabel();
    AddCriteriaLabel();
    CustomizeCellsInThirdColumn();
    AddContextMenu();
    SetDefaultCellInFirstColumn();
    ToolTips();
    dataGridView->CellMouseEnter += gcnew DataGridViewCellEventHandler( this,
&DataGridViewColumnDemo::dataGridView_CellMouseEnter );
    dataGridView->AutoSizeColumnsMode += gcnew DataGridViewAutoSizeColumnsModeEventHandler( this,
&DataGridViewColumnDemo::dataGridView_AutoSizeModeChanged );
}

String^ criteriaLabel;
void AddCriteriaLabel()
{
    AddLabelToPanelIfNotAlreadyThere( criteriaLabel, String::Concat( criteriaLabel, dataGridView->Columns[ 2
]->AutoSizeMode, L"." ) );
}

void AddContextLabel()
{
    String^ labelName = L"label";
    AddLabelToPanelIfNotAlreadyThere( labelName, L"Use shortcut menu to change cell color." );
}

void AddLabelToPanelIfNotAlreadyThere( String^ labelName, String^ labelText )
{
    Label^ label;
    if ( FlowLayoutPanel1->Controls[ labelName ] == nullptr )
    {
        label = gcnew Label;
        label->AutoSize = true;
        label->Name = labelName;
        label->BackColor = Color::Bisque;
        FlowLayoutPanel1->Controls->Add( label );
    }
    else
    {
        label = dynamic_cast<Label^>(FlowLayoutPanel1->Controls[ labelName ]);
    }

    label->Text = labelText;
}

void CustomizeCellsInThirdColumn()
{
    int thirdColumn = 2;
    DataGridViewColumn^ column = dataGridView->Columns[ thirdColumn ];
    DataGridViewCell^ cell = gcnew DataGridViewTextBoxCell;
}

```

```

        cell->Style->BackColor = Color::Wheat;
        column->CellTemplate = cell;
    }

    ToolStripMenuItem^ toolStripItem1;
    void AddContextMenu()
    {
        toolStripItem1->Text = L"Redden";
        toolStripItem1->Click += gcnew EventHandler( this, &DataGridViewColumnDemo::toolStripItem1_Click );
        System::Windows::Forms::ContextMenuStrip^ strip = gcnew System::Windows::Forms::ContextMenuStrip;
        IEnumrator^ myEnum = dataGridView->Columns->GetEnumerator();
        while ( myEnum->MoveNext() )
        {
            DataGridViewColumn^ column = safe_cast<DataGridViewColumn^>(myEnum->Current);
            column->ContextMenuStrip = strip;
            column->ContextMenuStrip->Items->Add( toolStripItem1 );
        }
    }

    DataGridViewCellEventArgs^ mouseLocation;

    // Change the cell's color.
    void toolStripItem1_Click( Object^ /*sender*/, EventArgs^ /*args*/ )
    {
        dataGridView->Rows[ mouseLocation->RowIndex ]->Cells[ mouseLocation->ColumnIndex ]->Style->BackColor =
Color::Red;
    }

    // Deal with hovering over a cell.
    void dataGridView_CellMouseEnter( Object^ /*sender*/, DataGridViewCellEventArgs^ location )
    {
        mouseLocation = location;
    }

    void SetDefaultCellInFirstColumn()
    {
        DataGridViewColumn^ firstColumn = dataGridView->Columns[ 0 ];
        DataGridViewCellStyle^ cellStyle = gcnew DataGridViewCellStyle;
        cellStyle->BackColor = Color::Thistle;
        firstColumn->DefaultCellStyle = cellStyle;
    }

    void ToolTips()
    {
        DataGridViewColumn^ firstColumn = dataGridView->Columns[ 0 ];
        DataGridViewColumn^ thirdColumn = dataGridView->Columns[ 2 ];
        firstColumn->ToolTipText = L"This column uses a default cell.";
        thirdColumn->ToolTipText = L"This column uses a template cell."
L" Style changes to one cell apply to all cells.";
    }

    void AddAdditionalButtons()
    {
        AddButton( Button4, L"Set Minimum Width of Column Two", gcnew EventHandler( this,
&DataGridViewColumnDemo::Button4_Click ) );
        AddButton( Button5, L"Set Width of Column One", gcnew EventHandler( this,
&DataGridViewColumnDemo::Button5_Click ) );
        AddButton( Button6, L"Autosize Third Column", gcnew EventHandler( this,
&DataGridViewColumnDemo::Button6_Click ) );
        AddButton( Button7, L"Add Thick Vertical Edge", gcnew EventHandler( this,
&DataGridViewColumnDemo::Button7_Click ) );
        AddButton( Button8, L"Style and Number Columns", gcnew EventHandler( this,
&DataGridViewColumnDemo::Button8_Click ) );
        AddButton( Button9, L"Change Column Header Text", gcnew EventHandler( this,

```

```

&DataGridViewColumnDemo::Button9_Click ) );
    AddButton( Button10, L"Swap First and Last Columns", gcnew EventHandler( this,
&DataGridViewColumnDemo::Button10_Click ) );
}

void AdjustDataGridViewSizing()
{
    dataGridView->ColumnHeadersHeightSizeMode = DataGridViewColumnHeadersHeightSizeMode::AutoSize;
}

//Set the minimum width.
void Button4_Click( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    DataGridViewColumn^ column = dataGridView->Columns[ 1 ];
    column->MinimumWidth = 40;
}

// Set the width.
void Button5_Click( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    DataGridViewColumn^ column = dataGridView->Columns[ 0 ];
    column->Width = 60;
}

// AutoSize the third column.
void Button6_Click( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    DataGridViewColumn^ column = dataGridView->Columns[ 2 ];
    column->AutoSizeMode = DataGridViewAutoSizeColumnsMode::DisplayedCells;
}

// Set the vertical edge.
void Button7_Click( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    int thirdColumn = 2;

    //      int edgeThickness = 5;
    DataGridViewColumn^ column = dataGridView->Columns[ thirdColumn ];
    column->DividerWidth = 10;
}

// Style and number columns.
void Button8_Click( Object^ /*sender*/, EventArgs^ /*args*/ )
{
    DataGridViewCellStyle^ style = gcnew DataGridViewCellStyle;
    style->Alignment = DataGridViewContentAlignment::MiddleCenter;
    style->ForeColor = Color::IndianRed;
    style->BackColor = Color::Ivory;
    IEnum^ myEnum1 = dataGridView->Columns->GetEnumerator();
    while ( myEnum1->MoveNext() )
    {
        DataGridViewColumn^ column = safe_cast<DataGridViewColumn^>(myEnum1->Current);
        column->HeaderCell->Value = column->Index.ToString();
        column->HeaderCell->Style = style;
    }
}

// Change the text in the column header.
void Button9_Click( Object^ /*sender*/, EventArgs^ /*args*/ )
{
    IEnum^ myEnum2 = dataGridView->Columns->GetEnumerator();
    while ( myEnum2->MoveNext() )
    {
}

```

```

        DataGridViewColumn^ column = safe_cast<DataGridViewColumn^>(myEnum2->Current);
        column->HeaderText = String::Concat( L"Column ", column->Index.ToString() );
    }

}

// Swap the last column with the first.
void Button10_Click( Object^ /*sender*/, EventArgs^ /*args*/ )
{
    DataGridViewColumnCollection^ columnCollection = dataGridView->Columns;
    DataGridViewColumn^ firstDisplayedColumn = columnCollection->GetFirstColumn(
DataGridViewElementStates::Visible );
    DataGridViewColumn^ lastDisplayedColumn = columnCollection->GetLastColumn(
DataGridViewElementStates::Visible, DataGridViewElementStates::None );
    int firstColumn_sIndex = firstDisplayedColumn->DisplayIndex;
    firstDisplayedColumn->DisplayIndex = lastDisplayedColumn->DisplayIndex;
    lastDisplayedColumn->DisplayIndex = firstColumn_sIndex;
}

// Updated the criteria label.
void dataGridView_AutoSizeColumnsModeChanged( Object^ /*sender*/, DataGridViewAutoSizeColumnsModeEventArgs^
args )
{
    args->Column->DataGridView->Parent->Controls[ L"flowLayoutPanel" ]->Controls[ criteriaLabel ]->Text =
String::Concat( criteriaLabel, args->Column->AutoSizeMode );
}
#pragma endregion

};

int main()
{
    DataGridViewColumnDemo::Main();
}

```

```

using System.Windows.Forms;
using System;
using System.Drawing;

public class DataGridViewColumnDemo : Form
{
    #region "set up form"
    public DataGridViewColumnDemo()
    {
        InitializeComponent();

        AddButton(Button1, "Reset",
            new EventHandler(ResetToDisorder));
        AddButton(Button2, "Change Column 3 Header",
            new EventHandler(ChangeColumn3Header));
        AddButton(Button3, "Change Meatloaf Recipe",
            new EventHandler(ChangeMeatloafRecipe));
        AddAdditionalButtons();

        InitializeDataGridView();
    }

    DataGridView dataGridView;
    Button Button1 = new Button();
    Button Button2 = new Button();
    Button Button3 = new Button();
    Button Button4 = new Button();
    Button Button5 = new Button();
    Button Button6 = new Button();
    Button Button7 = new Button();

```

```

        Button Button8 = new Button();
        Button Button9 = new Button();
        Button Button10 = new Button();
        FlowLayoutPanel FlowLayoutPanel1 = new FlowLayoutPanel();

        private void InitializeComponent()
        {
            FlowLayoutPanel1.Location = new Point(454, 0);
            FlowLayoutPanel1.AutoSize = true;
            FlowLayoutPanel1.FlowDirection = FlowDirection.TopDown;
            FlowLayoutPanel1.Name = "flowlayoutpanel";
            ClientSize = new System.Drawing.Size(614, 360);
            Controls.Add(this.flowLayoutPanel1);
            Text = this.GetType().Name;
            AutoSize = true;
        }
        #endregion

        #region "set up DataGridView"

        private string thirdColumnHeader = "Main Ingredients";
        private string boringMeatloaf = "ground beef";
        private string boringMeatloafRanking = "*";
        private bool boringRecipe;
        private bool shortMode;

        private void InitializeDataGridView()
        {
            dataGridView = new System.Windows.Forms.DataGridView();
            Controls.Add(dataGridView);
            dataGridView.Size = new Size(300, 200);

            // Create an unbound DataGridView by declaring a
            // column count.
            dataGridView.ColumnCount = 4;
            AdjustDataGridViewSizing();

            // Set the column header style.
            DataGridViewCellStyle columnHeaderStyle =
                new DataGridViewCellStyle();
            columnHeaderStyle.BackColor = Color.Aqua;
            columnHeaderStyle.Font =
                new Font("Verdana", 10, FontStyle.Bold);
            dataGridView.ColumnHeadersDefaultCellStyle =
                columnHeaderStyle;

            // Set the column header names.
            dataGridView.Columns[0].Name = "Recipe";
            dataGridView.Columns[1].Name = "Category";
            dataGridView.Columns[2].Name = thirdColumnHeader;
            dataGridView.Columns[3].Name = "Rating";

            PostColumnCreation();

            // Populate the rows.
            string[] row1 = new string[]{"Meatloaf",
                                         "Main Dish", boringMeatloaf, boringMeatloafRanking};
            string[] row2 = new string[]{"Key Lime Pie",
                                         "Dessert", "lime juice, evaporated milk", "****"};
            string[] row3 = new string[]{"Orange-Salsa Pork Chops",
                                         "Main Dish", "pork chops, salsa, orange juice", "****"};
            string[] row4 = new string[]{"Black Bean and Rice Salad",
                                         "Salad", "black beans, brown rice", "****"};
            string[] row5 = new string[]{"Chocolate Cheesecake",
                                         "Dessert", "cream cheese", "***"};
            string[] row6 = new string[]{"Black Bean Dip", "Appetizer",
                                         "black beans, sour cream", "***"};
            object[] rows = new object[] { row1, row2, row3, row4, row5, row6 };
        }
    
```

```

        foreach (string[] rowArray in rows)
        {
            dataGridView.Rows.Add(rowArray);
        }

        shortMode = false;
        boringRecipe = true;
    }

    private void AddButton(Button button, string buttonLabel,
        EventHandler handler)
    {
        FlowLayoutPanel1.Controls.Add(button);
        button.TabIndex = FlowLayoutPanel1.Controls.Count;
        button.Text = buttonLabel;
        button.AutoSize = true;
        button.Click += handler;
    }

    private void ResetToDisorder(object sender, System.EventArgs e)
    {
        Controls.Remove(dataGridView);
        dataGridView.Dispose();
        InitializeDataGridView();
    }

    private void ChangeColumn3Header(object sender,
        System.EventArgs e)
    {
        Toggle(ref shortMode);
        if (shortMode)
        { dataGridView.Columns[2].HeaderText = "S"; }
        else
        { dataGridView.Columns[2].HeaderText = thirdColumnHeader; }
    }

    private static void Toggle(ref bool toggleThis)
    {
        toggleThis = !toggleThis;
    }

    private void ChangeMeatloafRecipe(object sender,
        System.EventArgs e)
    {
        Toggle(ref boringRecipe);
        if (boringRecipe)
        {
            SetMeatloaf(boringMeatloaf, boringMeatloafRanking);
        }
        else
        {
            string greatMeatloafRecipe =
                "1 lb. lean ground beef, " +
                "1/2 cup bread crumbs, 1/4 cup ketchup," +
                "1/3 tsp onion powder, " +
                "1 clove of garlic, 1/2 pack onion soup mix " +
                " dash of your favorite BBQ Sauce";
            SetMeatloaf(greatMeatloafRecipe, "***");
        }
    }

    private void SetMeatloaf(string recipe, string rating)
    {
        dataGridView.Rows[0].Cells[2].Value = recipe;
        dataGridView.Rows[0].Cells[3].Value = rating;
    }
#endregion

#region "demonstration code"

```

```
#region DEMONSTRATION CODE
private void PostColumnCreation()
{
    AddContextLabel();
    AddCriteriaLabel();
    CustomizeCellsInThirdColumn();
    AddContextMenu();
    SetDefaultCellInFirstColumn();
    ToolTips();

    dataGridView.CellMouseEnter += 
        dataGridView_CellMouseEnter;
    dataGridView.AutoSizeColumnsMode += 
        dataGridView_AutoSizeColumnsMode;
}

private string criteriaLabel = "Column 3 sizing criteria: ";
private void AddCriteriaLabel()
{
    AddLabelToPanelIfNotAlreadyThere(criteriaLabel,
        criteriaLabel +
        dataGridView.Columns[2].AutoSizeMode.ToString() +
        ".");
}

private void AddContextLabel()
{
    string labelName = "label";
    AddLabelToPanelIfNotAlreadyThere(labelName,
        "Use shortcut menu to change cell color.");
}

private void AddLabelToPanelIfNotAlreadyThere(
    string labelName, string labelText)
{
    Label label;
    if (FlowLayoutPanel1.Controls[labelName] == null)
    {
        label = new Label();
        label.AutoSize = true;
        label.Name = labelName;
        label.BackColor = Color.Bisque;
        FlowLayoutPanel1.Controls.Add(label);
    }
    else
    {
        label = (Label)FlowLayoutPanel1.Controls[labelName];
    }
    label.Text = labelText;
}

private void CustomizeCellsInThirdColumn()
{
    int thirdColumn = 2;
    DataGridViewColumn column =
        dataGridView.Columns[thirdColumn];
    DataGridViewCell cell = new DataGridViewTextBoxCell();

    cell.Style.BackColor = Color.Wheat;
    column.CellTemplate = cell;
}

ToolStripMenuItem toolStripItem1 = new ToolStripMenuItem();

private void AddContextMenu()
{
    toolStripItem1.Text = "Redden";
    toolStripItem1.Click += new EventHandler(toolStripItem1_Click);
    ContextMenuStrip strip = new ContextMenuStrip();
    strip.Items.Add(toolStripItem1);
}
```

```

foreach (DataGridViewColumn column in dataGridView.Columns)
{
    column.ContextMenuStrip = strip;
    column.ContextMenuStrip.Items.Add(toolStripItem1);
}
}

private DataGridViewCellEventArgs mouseLocation;

// Change the cell's color.
private void toolStripItem1_Click(object sender, EventArgs args)
{
    dataGridView.Rows[mouseLocation.RowIndex]
        .Cells[mouseLocation.ColumnIndex].Style.BackColor
        = Color.Red;
}

// Deal with hovering over a cell.
private void dataGridView_CellMouseEnter(object sender,
    DataGridViewCellEventArgs location)
{
    mouseLocation = location;
}

private void SetDefaultCellInFirstColumn()
{
    DataGridViewColumn firstColumn = dataGridView.Columns[0];
    DataGridCellStyle cellStyle =
        new DataGridCellStyle();
    cellStyle.BackColor = Color.Thistle;

    firstColumn.DefaultCellStyle = cellStyle;
}

private void ToolTips()
{
    DataGridViewColumn firstColumn = dataGridView.Columns[0];
    DataGridViewColumn thirdColumn = dataGridView.Columns[2];
    firstColumn.ToolTipText =
        "This column uses a default cell.";
    thirdColumn.ToolTipText =
        "This column uses a template cell." +
        " Style changes to one cell apply to all cells.";
}

private void AddAdditionalButtons()
{
    AddButton(Button4, "Set Minimum Width of Column Two",
        new EventHandler(Button4_Click));
    AddButton(Button5, "Set Width of Column One",
        new EventHandler(Button5_Click));
    AddButton(Button6, "Autosize Third Column",
        new EventHandler(Button6_Click));
    AddButton(Button7, "Add Thick Vertical Edge",
        new EventHandler(Button7_Click));
    AddButton(Button8, "Style and Number Columns",
        new EventHandler(Button8_Click));
    AddButton(Button9, "Change Column Header Text",
        new EventHandler(Button9_Click));
    AddButton(Button10, "Swap First and Last Columns",
        new EventHandler(Button10_Click));
}

private void AdjustDataGridViewSizing()
{
    dataGridView.ColumnHeadersHeightSizeMode =
        DataGridViewColumnHeadersHeightSizeMode.AutoSize;
}

```

```

}

//Set the minimum width.
private void Button4_Click(object sender,
    System.EventArgs e)
{
    DataGridViewColumn column = dataGridView.Columns[1];
    column.MinimumWidth = 40;
}

// Set the width.
private void Button5_Click(object sender, System.EventArgs e)
{
    DataGridViewColumn column = dataGridView.Columns[0];
    column.Width = 60;
}

// AutoSize the third column.
private void Button6_Click(object sender,
    System.EventArgs e)
{
    DataGridViewColumn column = dataGridView.Columns[2];
    column.AutoSizeMode = DataGridViewAutoSizeColumnMode.DisplayedCells;
}

// Set the vertical edge.
private void Button7_Click(object sender,
    System.EventArgs e)
{
    int thirdColumn = 2;
    DataGridViewColumn column =
        dataGridView.Columns[thirdColumn];
    column.DividerWidth = 10;
}

// Style and number columns.
private void Button8_Click(object sender,
    EventArgs args)
{
    DataGridViewCellStyle style = new DataGridViewCellStyle();
    style.Alignment =
        DataGridViewContentAlignment.MiddleCenter;
    style.ForeColor = Color.IndianRed;
    style.BackColor = Color.Ivory;

    foreach (DataGridViewColumn column in dataGridView.Columns)
    {
        column.HeaderCell.Value = column.Index.ToString();
        column.HeaderCell.Style = style;
    }
}

// Change the text in the column header.
private void Button9_Click(object sender,
    EventArgs args)
{
    foreach (DataGridViewColumn column in dataGridView.Columns)
    {

        column.HeaderText = String.Concat("Column ",
            column.Index.ToString());
    }
}

// Swap the last column with the first.
private void Button10_Click(object sender, EventArgs args)
{
    DataGridViewColumnCollection columnCollection = dataGridView.Columns;
}

```

```

        DataGridViewColumn firstVisibleColumn =
            columnCollection.GetFirstColumn(DataGridViewElementStates.Visible);
        DataGridViewColumn lastVisibleColumn =
            columnCollection.GetLastColumn(
                DataGridViewElementStates.Visible, DataGridViewElementStates.None);

        int firstColumn_sIndex = firstVisibleColumn.DisplayIndex;
        firstVisibleColumn.DisplayIndex = lastVisibleColumn.DisplayIndex;
        lastVisibleColumn.DisplayIndex = firstColumn_sIndex;
    }

    // Updated the criteria label.
    private void dataGridView_AutoSizeColumnModeChanged(object sender,
        DataGridViewAutoSizeColumnModeEventArgs args)
    {
        args.Column.DataGridView.Parent.
            Controls["flowLayoutPanel"].Controls[criteriaLabel].
            Text = criteriaLabel
            + args.Column.AutoSizeMode.ToString();
    }
}

#endregion

[STAThreadAttribute()]
public static void Main()
{
    Application.Run(new DataGridViewColumnDemo());
}
}

```

```

Imports System.Windows.Forms
Imports System.Drawing

Public Class DataGridViewColumnDemo
    Inherits Form

#Region "set up form"
    Public Sub New()
        InitializeComponent()

        AddButton(Button1, "Reset")
        AddButton(Button2, "Change Column 3 Header")
        AddButton(Button3, "Change Meatloaf Recipe")
        AddAdditionalButtons()
    End Sub

    Friend WithEvents dataGridView As DataGridView
    Friend WithEvents Button1 As Button = New Button()
    Friend WithEvents Button2 As Button = New Button()
    Friend WithEvents Button3 As Button = New Button()
    Friend WithEvents Button4 As Button = New Button()
    Friend WithEvents Button5 As Button = New Button()
    Friend WithEvents Button6 As Button = New Button()
    Friend WithEvents Button7 As Button = New Button()
    Friend WithEvents Button8 As Button = New Button()
    Friend WithEvents Button9 As Button = New Button()
    Friend WithEvents Button10 As Button = New Button()
    Friend WithEvents FlowLayoutPanel1 As FlowLayoutPanel _
        = New FlowLayoutPanel()

    Private Sub InitializeComponent()
        FlowLayoutPanel1.Location = New Point(454, 0)
        FlowLayoutPanel1.AutoSize = True
        FlowLayoutPanel1.FlowDirection = FlowDirection.TopDown
        FlowLayoutPanel1.Name = "flowLayoutPanel"
        ClientSize = New System.Drawing.Size(614, 360)
    End Sub

```

```

        Controls.Add(FlowLayoutPanel1)
        Text = Me.GetType.Name
        AutoSize = True
    End Sub
#End Region

#Region "set up DataGridView"
    Private thirdColumnHeader As String = "Main Ingredients"
    Private boringMeatloaf As String = "ground beef"
    Private boringMeatloafRanking As String = "*"
    Private boringRecipe As Boolean
    Private shortMode As Boolean

    Private Sub InitializeDataGridView(ByVal ignored As Object, _
    ByVal ignoredToo As EventArgs) Handles Me.Load

        dataGridView = New System.Windows.Forms.DataGridView
        Controls.Add(dataGridView)
        dataGridView.Size = New Size(300, 200)

        ' Create an unbound DataGridView by declaring a
        ' column count.
        dataGridView.ColumnCount = 4
        AdjustDataGridViewSizing()

        ' Set the column header style.
        Dim columnHeaderStyle As New DataGridViewCellStyle()
        columnHeaderStyle.BackColor = Color.Aqua
        columnHeaderStyle.Font = _
            New Font("Verdana", 10, FontStyle.Bold)
        dataGridView.ColumnHeadersDefaultCellStyle = _
            columnHeaderStyle

        ' Set the column header names.
        dataGridView.Columns(0).Name = "Recipe"
        dataGridView.Columns(1).Name = "Category"
        dataGridView.Columns(2).Name = thirdColumnHeader
        dataGridView.Columns(3).Name = "Rating"

        PostColumnCreation()

        ' Populate the rows.
        Dim row1 As String() = New String() _
            {"Meatloaf", "Main Dish", boringMeatloaf, _
            boringMeatloafRanking}
        Dim row2 As String() = New String() _
            {"Key Lime Pie", "Dessert", _
            "lime juice, evaporated milk", _
            "****"}
        Dim row3 As String() = New String() _
            {"Orange-Salsa Pork Chops", "Main Dish", _
            "pork chops, salsa, orange juice", "****"}
        Dim row4 As String() = New String() _
            {"Black Bean and Rice Salad", "Salad", _
            "black beans, brown rice", _
            "****"}
        Dim row5 As String() = New String() _
            {"Chocolate Cheesecake", "Dessert", "cream cheese", _
            "****"}
        Dim row6 As String() = New String() _
            {"Black Bean Dip", "Appetizer", _
            "black beans, sour cream", _
            "****"}

        Dim rows As Object() = New Object() {row1, row2, _
            row3, row4, row5, row6}

        Dim rowArray As String()
        For Each rowArray In rows
            dataGridView.Rows.Add(rowArray)
        Next
    End Sub

```

Next

```
shortMode = False
boringRecipe = True
End Sub

Private Sub AddButton(ByVal button As Button, _
ByVal buttonLabel As String)

    FlowLayoutPanel1.Controls.Add(button)
    button.TabIndex = FlowLayoutPanel1.Controls.Count
    button.Text = buttonLabel
    button.AutoSize = True
End Sub

Private Sub ResetToDisorder(ByVal sender As Object, _
ByVal e As System.EventArgs) _
Handles Button1.Click
    Controls.Remove(dataGridView)
    dataGridView.Dispose()
    InitializeDataGridView(Nothing, Nothing)
End Sub

Private Sub ChangeColumn3Header(ByVal sender As Object, _
ByVal e As System.EventArgs) _
Handles Button2.Click

    Toggle(shortMode)
    If shortMode Then dataGridView.Columns(2).HeaderText = _
        "S" _
    Else dataGridView.Columns(2).HeaderText = _
        thirdColumnHeader
End Sub

Private Shared Sub Toggle(ByRef toggleThis As Boolean)
    toggleThis = Not toggleThis
End Sub

Private Sub ChangeMeatloafRecipe(ByVal sender As Object, _
ByVal e As System.EventArgs) _
Handles Button3.Click

    Toggle(boringRecipe)
    If boringRecipe Then
        SetMeatloaf(boringMeatloaf, boringMeatloafRanking)
    Else
        Dim greatMeatloafRecipe As String = "1 lb. lean ground beef, " _
            & "1/2 cup bread crumbs, 1/4 cup ketchup," _
            & "1/3 tsp onion powder, " _
            & "1 clove of garlic, 1/2 pack onion soup mix " _
            & "dash of your favorite BBQ Sauce"
        SetMeatloaf(greatMeatloafRecipe, "***")
    End If
End Sub

Private Sub SetMeatloaf(ByVal recipe As String, _
 ByVal rating As String)

    dataGridView.Rows(0).Cells(2).Value = recipe
    dataGridView.Rows(0).Cells(3).Value = rating
End Sub
#End Region

#Region "demonstration code"
Private Sub PostColumnCreation()
    AddContextLabel()
    AddCriteriaLabel()
    CustomizeCellsInThirdColumn()
    AddContextMenu()
End Sub
```

```

        SetDefaultCellInFirstColumn()
        ToolTips()
    End Sub

    Private criteriaLabel As String = "Column 3 sizing criteria: "
    Private Sub AddCriteriaLabel()
        AddLabelToPanelIfNotAlreadyThere(criteriaLabel, _
            criteriaLabel & _
            dataGridView.Columns(2).AutoSizeMode.ToString() _
            & ".")
    End Sub

    Private Sub AddContextLabel()
        Dim labelName As String = "label"
        AddLabelToPanelIfNotAlreadyThere(labelName, _
            "Use shortcut menu to change cell color.")
    End Sub

    Private Sub AddLabelToPanelIfNotAlreadyThere( _
        ByVal labelName As String, _
        ByVal labelText As String)

        Dim label As Label
        If FlowLayoutPanel1.Controls(labelName) Is Nothing Then
            label = New Label()
            label.AutoSize = True
            label.Name = labelName
            label.BackColor = Color.Bisque
            FlowLayoutPanel1.Controls.Add(label)
        Else
            label = CType(FlowLayoutPanel1.Controls(labelName), Label)
        End If
        label.Text = labelText
    End Sub

    Private Sub CustomizeCellsInThirdColumn()

        Dim thirdColumn As Integer = 2
        Dim column As DataGridViewColumn = _
            dataGridView.Columns(thirdColumn)
        Dim cell As DataGridViewCell = _
            New DataGridViewTextBoxCell()

        cell.Style.BackColor = Color.Wheat
        column.CellTemplate = cell
    End Sub

    WithEvents toolStripItem1 As New ToolStripMenuItem()

    Private Sub AddContextMenu()
        toolStripItem1.Text = "Redden"
        Dim strip As New ContextMenuStrip()
        For Each column As DataGridViewColumn _
            In dataGridView.Columns()

            column.ContextMenuStrip = strip
            column.ContextMenuStrip.Items.Add(toolStripItem1)
        Next
    End Sub
    ' Change the cell's color.
    Private Sub toolStripItem1_Click(ByVal sender As Object, _
        ByVal args As EventArgs) _
        Handles toolStripItem1.Click

        dataGridView.Rows(mouseLocation.RowIndex) _
            .Cells(mouseLocation.ColumnIndex) _
            .Style.BackColor = Color.Red
    End Sub

```

```

Private mouseLocation As DataGridViewCellEventArgs

' Deal with hovering over a cell.
Private Sub dataGridView_CellMouseEnter(ByVal sender As Object, _
    ByVal location As DataGridViewCellEventArgs) _
Handles DataGridView.CellMouseEnter

    mouseLocation = location
End Sub

Private Sub SetDefaultCellInFirstColumn()
    Dim firstColumn As DataGridViewColumn = _
        dataGridView.Columns(0)
    Dim cellStyle As DataGridViewCellStyle = _
        New DataGridViewCellStyle()
    cellStyle.BackColor = Color.Thistle

    firstColumn.DefaultCellStyle = cellStyle
End Sub

Private Sub ToolTips()
    Dim firstColumn As DataGridViewColumn = _
        dataGridView.Columns(0)
    Dim thirdColumn As DataGridViewColumn = _
        dataGridView.Columns(2)
    firstColumn.ToolTipText = _
        "This is column uses a default cell."
    thirdColumn.ToolTipText = _
        "This is column uses a template cell." _
        & " Changes to one cell's style changes them all."
End Sub

Private Sub AddAdditionalButtons()
    AddButton(Button4, "Set Minimum Width of Column Two")
    AddButton(Button5, "Set Width of Column One")
    AddButton(Button6, "Autosize Third Column")
    AddButton(Button7, "Add Thick Vertical Edge")
    AddButton(Button8, "Style and Number Columns")
    AddButton(Button9, "Change Column Header Text")
    AddButton(Button10, "Swap First and Last Columns")
End Sub

Private Sub AdjustDataGridViewSizing()
    dataGridView.ColumnHeadersHeightSizeMode = _
        DataGridViewColumnHeadersHeightSizeMode.AutoSize
End Sub

' Set the minimum width.
Private Sub Button4_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button4.Click

    Dim column As DataGridViewColumn = dataGridView.Columns(1)
    column.MinimumWidth = 40
End Sub

' Set the width.
Private Sub Button5_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button5.Click

    Dim column As DataGridViewColumn = dataGridView.Columns(0)
    column.Width = 60
End Sub

' AutoSize the third column.
Private Sub Button6_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button6.Click

    Dim column As DataGridViewColumn = dataGridView.Columns(2)
    column.AutoSizeMode = DataGridViewAutoSizeColumnMode.DisplayedCells

```

```

    COLUMN1.AUTOSIZemode = DataGridViewAutoSizeColumnsMode.DisplayedCells
End Sub

' Set the vertical edge.
Private Sub Button7_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button7.Click

    Dim thirdColumn As Integer = 2
    Dim column As DataGridViewColumn = _
        dataGridView.Columns(thirdColumn)
    column.DividerWidth = 10

End Sub

' Style and number columns.
Private Sub Button8_Click(ByVal sender As Object, _
    ByVal args As EventArgs) Handles Button8.Click

    Dim style As DataGridViewCellStyle = _
        New DataGridViewCellStyle()
    style.Alignment = _
        DataGridViewContentAlignment.MiddleCenter
    style.ForeColor = Color.IndianRed
    style.BackColor = Color.Ivory

    For Each column As DataGridViewColumn _
        In dataGridView.Columns

        column.HeaderCell.Value = _
            column.Index.ToString
        column.HeaderCell.Style = style
    Next
End Sub

' Change the text in the column header.
Private Sub Button9_Click(ByVal sender As Object, _
    ByVal args As EventArgs) Handles Button9.Click

    For Each column As DataGridViewColumn _
        In dataGridView.Columns

        column.HeaderText = String.Concat("Column ", _
            column.Index.ToString)
    Next
End Sub

' Swap the last column with the first.
Private Sub Button10_Click(ByVal sender As Object, _
    ByVal args As EventArgs) Handles Button10.Click

    Dim columnCollection As DataGridViewColumnCollection = _
        dataGridView.Columns

    Dim firstVisibleColumn As DataGridViewColumn = _
        columnCollection.GetFirstColumn(DataGridViewElementStates.Visible)
    Dim lastVisibleColumn As DataGridViewColumn = _
        columnCollection.GetLastColumn(DataGridViewElementStates.Visible, _
        Nothing)

    Dim firstColumn_sIndex As Integer = firstVisibleColumn.DisplayIndex
    firstVisibleColumn.DisplayIndex = _
        lastVisibleColumn.DisplayIndex
    lastVisibleColumn.DisplayIndex = firstColumn_sIndex
End Sub

' Updated the criteria label.
Private Sub dataGridView_AutoSizeColumnCriteriaChanged( _
    ByVal sender As Object, _
    ByVal args As DataGridViewAutoSizeColumnModeEventArgs) _
    Handles dataGridView.AutoSizeColumnCriteriaChanged

```

```

Handles DataGridView.AutoSizeColumnsModeChanged

    args.Column.DataGridView.Parent. _
    Controls("flowLayoutPanel"). _
    Controls(criteriaLabel).Text = _
        criteriaLabel & args.Column.AutoSizeMode.ToString
End Sub
#End Region

<STAThreadAttribute()> _
Public Shared Sub Main()
    Application.Run(New DataGridViewColumnDemo())
End Sub

End Class

```

Compiling the Code

This example requires:

- References to the System, System.Drawing, and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[DataGridView](#)

[DataGridViewBand](#)

[DataGridViewRow](#)

[DataGridViewColumn](#)

[Programming with Cells, Rows, and Columns in the Windows Forms DataGridView Control](#)

How to: Work with Image Columns in the Windows Forms DataGridView Control

5/4/2018 • 23 min to read • [Edit Online](#)

The following code example shows how to use the [DataGridView](#) image columns in an interactive user interface (UI). The example also demonstrates image sizing and layout possibilities with the [DataGridViewImageColumn](#).

Example

```

        0x0F, 0x0F, 0x0FF, 0x0D, 0x0D, 0x0FF, 0x0D, 0x0FF, 0x0D,
        0xFF, 0xF0, 0x0, 0x0, 0xFF, 0xF, 0xFF, 0xF, 0xF0, 0x0,
        0x0, 0xF0, 0xFF, 0xFF, 0xF0, 0xF0, 0x0, 0x0, 0xFF,
        0xFF, 0xFF, 0xFF, 0xF0, 0x0, 0x0};

blankBytes = gcnew array<Byte>{
    0x42, 0x4D, 0xC6, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x76,
    0x0, 0x0, 0x0, 0x28, 0x0, 0x0, 0xB, 0x0, 0x0, 0x0, 0xA,
    0x0, 0x0, 0x0, 0x1, 0x0, 0x4, 0x0, 0x0, 0x0, 0x0, 0x0, 0x50,
    0x0, 0x10,
    0x0, 0x0, 0x0, 0x10, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
    0x0, 0x80, 0x0, 0x0, 0x80, 0x0, 0x0, 0x0, 0x80, 0x80, 0x0,
    0x80, 0x0, 0x0, 0x0, 0x80, 0x0, 0x80, 0x0, 0x80, 0x80, 0x0,
    0x0, 0xC0, 0xC0, 0xC0, 0x0, 0x80, 0x80, 0x0, 0x0, 0x0, 0x0,
    0xFF, 0x0, 0x0, 0xFF, 0x0, 0x0, 0xFF, 0xFF, 0x0, 0xFF,
    0x0, 0x0, 0x0, 0xFF, 0x0, 0xFF, 0x0, 0xFF, 0x0, 0x0,
    0xFF, 0xFF, 0xFF, 0xFF, 0xF0, 0x0, 0x0, 0xFF, 0xFF,
    0xFF, 0xFF, 0xF0, 0x0, 0x0, 0xFF, 0xFF, 0x0, 0xFF,
    0x0, 0x0, 0xFF, 0xFF, 0xFF, 0xF0, 0x0, 0x0, 0x0,
    0xFF, 0xFF, 0xFF, 0xF0, 0x0, 0x0, 0x0, 0x0, 0x0,
    0xFF, 0xFF, 0xF0, 0x0, 0x0, 0xFF, 0xFF, 0x0, 0xFF,
    0xFF, 0xFF, 0xF0, 0x0, 0x0, 0xFF, 0xFF, 0x0, 0xFF,
    0xFF, 0xF0, 0x0, 0x0, 0xFF, 0xFF, 0x0, 0xFF, 0x0,
    0x0, 0x0, 0xFF, 0xFF, 0xFF, 0xF0, 0x0, 0x0, 0x0,
    0xFF, 0xFF, 0xFF, 0xF0, 0x0, 0x0, 0x0, 0x0, 0x0,
    0xFF, 0xFF, 0xF0, 0x0, 0x0, 0xFF, 0xFF, 0x0, 0xFF,
    0xFF, 0xFF, 0xF0, 0x0, 0x0, 0xFF, 0xFF, 0x0, 0xFF,
    0xFF, 0xF0, 0x0, 0x0, 0xFF, 0xFF, 0x0, 0xFF, 0x0,
    0x0, 0x0, 0xFF, 0xFF, 0xFF, 0xF0, 0x0, 0x0, 0x0,
    0xFF, 0xFF, 0xFF, 0xF0, 0x0, 0x0, 0x0, 0x0, 0x0};

blank = gcnew Bitmap( gcnew MemoryStream( blankBytes ) );
x = gcnew Bitmap( gcnew MemoryStream( xBytes ) );
o = gcnew Bitmap( gcnew MemoryStream( oBytes ) );
this->AutoSize = true;
turn = gcnew Label;
Button1 = gcnew Button;
Button2 = gcnew Button;
Button3 = gcnew Button;
Button4 = gcnew Button;
Button5 = gcnew Button;
Panel1 = gcnew FlowLayoutPanel;
SetupButtons();
InitializeDataGridView( nullptr, nullptr );

xString = L"X's turn";
oString = L"O's turn";
gameOverString = L"Game Over";
bitmapPadding = 6;
}

private:
    DataGridView^ dataGridView1;
    Button^ Button1;
    Label^ turn;
    Button^ Button2;
    Button^ Button3;
    Button^ Button4;
    Button^ Button5;
    FlowLayoutPanel^ Panel1;

#pragma region S " bitmaps "
array<Byte>^oBytes;
array<Byte>^xBytes;
array<Byte>^blankBytes;

#pragma endregion
    Bitmap^ blank;
    Bitmap^ x;
    Bitmap^ o;
    String^ xString;
    String^ oString;
    String^ gameOverString;
    int bitmapPadding;
    void InitializeDataGridView( Object^ ignored, EventArgs^ e )
    {

```

```

{
    this->Panel1->SuspendLayout();
    this->SuspendLayout();
    ConfigureForm();
    SizeGrid();
    CreateColumns();
    CreateRows();
    this->Panel1->ResumeLayout( false );
    this->ResumeLayout( false );
}

void ConfigureForm()
{
    AutoSize = true;
    turn->Size = System::Drawing::Size( 75, 34 );
    turn->TextAlign = ContentAlignment::MiddleLeft;
    Panel1->Location = System::Drawing::Point( 0, 8 );
    Panel1->Size = System::Drawing::Size( 120, 196 );
    Panel1->FlowDirection = FlowDirection::TopDown;
    ClientSize = System::Drawing::Size( 355, 200 );
    Controls->Add( this->Panel1 );
    Text = L"TicTacToe";
    dataGridView1 = gcnew System::Windows::Forms::DataGridView;
    dataGridView1->Location = Point(120,0);
    dataGridView1->AllowUserToAddRows = false;
    dataGridView1->CellClick += gcnew DataGridViewCellEventHandler( this,
&TicTacToe::dataGridView1_CellClick );
    dataGridView1->CellMouseEnter += gcnew DataGridViewCellEventHandler( this,
&TicTacToe::dataGridView1_CellMouseEnter );
    dataGridView1->CellMouseLeave += gcnew DataGridViewCellEventHandler( this,
&TicTacToe::dataGridView1_CellMouseLeave );
    Controls->Add( dataGridView1 );
    turn->Text = xString;
    turn->AutoSize = true;
}

void SetupButtons()
{
    Button1->AutoSize = true;
    SetupButton( Button1, L"Restart", gcnew EventHandler( this, &TicTacToe::Reset ) );
    Panel1->Controls->Add( turn );
    SetupButton( Button2, L"Increase Cell Size", gcnew EventHandler( this, &TicTacToe::MakeCellsLarger ) );
    SetupButton( Button3, L"Stretch Images", gcnew EventHandler( this, &TicTacToe::Stretch ) );
    SetupButton( Button4, L"Zoom Images", gcnew EventHandler( this, &TicTacToe::ZoomToImage ) );
    SetupButton( Button5, L"Normal Images", gcnew EventHandler( this, &TicTacToe::NormalImage ) );
}

void SetupButton( Button^ button, String^ buttonLabel, EventHandler^ handler )
{
    Panel1->Controls->Add( button );
    button->Text = buttonLabel;
    button->AutoSize = true;
    button->Click += handler;
}

void CreateColumns()
{
    DataGridViewImageColumn^ imageColumn;
    int columnCount = 0;
    do
    {
        Bitmap^ unMarked = blank;
        imageColumn = gcnew DataGridViewImageColumn;

        //Add twice the padding for the left and
        //right sides of the cell.
        imageColumn->Width = x->Width + 2 * bitmapPadding + 1;
        imageColumn->Image = unMarked;
}

```

```

        dataGridView1->Columns->Add( imageColumn );
        columnCount = columnCount + 1;
    }
    while ( columnCount < 3 );
}

void CreateRows()
{
    dataGridView1->Rows->Add();
    dataGridView1->Rows->Add();
    dataGridView1->Rows->Add();
}

void SizeGrid()
{
    dataGridView1->ColumnHeadersVisible = false;
    dataGridView1->RowHeadersVisible = false;
    dataGridView1->AllowUserToResizeColumns = false;
    dataGridView1->AllowUserToResizeRows = false;
    dataGridView1->BorderStyle = BorderStyle::None;

    //Add twice the padding for the top of the cell
    //and the bottom.
    dataGridView1->RowTemplate->Height = x->Height + 2 * bitmapPadding + 1;
    dataGridView1->AutoSize = true;
}

void Reset( Object^ sender, System::EventArgs^ e )
{
    dataGridView1->~DataGridView();
    InitializeDataGridView( nullptr, nullptr );
}

void dataGridView1_CellClick( Object^ sender, DataGridViewCellEventArgs^ e )
{
    if ( turn->Equals( gameOverString ) )
    {
        return;
    }

    DataGridViewImageCell^ cell = dynamic_cast<DataGridViewImageCell^>(dataGridView1->Rows[ e->RowIndex ]->Cells[ e->ColumnIndex ]);
    if ( cell->Value == blank )
    {
        if ( IsOsTurn() )
        {
            cell->Value = o;
        }
        else
        {
            cell->Value = x;
        }

        ToggleTurn();
    }

    if ( IsAWin( cell ) )
    {
        turn->Text = gameOverString;
    }
}

void dataGridView1_CellMouseEnter( Object^ sender, DataGridViewCellEventArgs^ e )
{
}

```

```

        Bitmap^ markingUnderMouse = dynamic_cast<Bitmap^>(dataGridView1->Rows[ e->RowIndex ]->Cells[ e->ColumnIndex ]->Value);
        if ( markingUnderMouse == blank )
        {
            dataGridView1->Cursor = Cursors::Default;
        }
        else
        if ( markingUnderMouse == o || markingUnderMouse == x )
        {
            dataGridView1->Cursor = Cursors::No;
            ToolTip(e,true);
        }
    }

void ToolTip( DataGridViewCellEventArgs^ e, bool showTip )
{
    DataGridViewImageCell^ cell = dynamic_cast<DataGridViewImageCell^>(dataGridView1->Rows[ e->RowIndex ]->Cells[ e->ColumnIndex ]);
    DataGridViewImageColumn^ imageColumn = dynamic_cast<DataGridViewImageColumn^>(dataGridView1->Columns[ cell->ColumnIndex ]);
    if ( showTip )
        cell->ToolTipText = imageColumn->Description;
    else
    {
        cell->ToolTipText = String::Empty;
    }
}

void dataGridView1_CellMouseLeave( Object^ sender, DataGridViewCellEventArgs^ e )
{
    ToolTip( e, false );
    dataGridView1->Cursor = Cursors::Default;
}

void Stretch( Object^ sender, EventArgs^ e )
{
    System::Collections::IEnumerator^ myEnum = dataGridView1->Columns->GetEnumerator();
    while ( myEnum->MoveNext() )
    {
        DataGridViewImageColumn^ column = safe_cast<DataGridViewImageColumn^>(myEnum->Current);
        column->ImageLayout = DataGridViewCellLayout::Stretch;
        column->Description = L"Stretched";
    }
}

void ZoomToImage( Object^ sender, EventArgs^ e )
{
    System::Collections::IEnumerator^ myEnum1 = dataGridView1->Columns->GetEnumerator();
    while ( myEnum1->MoveNext() )
    {
        DataGridViewImageColumn^ column = safe_cast<DataGridViewImageColumn^>(myEnum1->Current);
        column->ImageLayout = DataGridViewCellLayout::Zoom;
        column->Description = L"Zoomed";
    }
}

void NormalImage( Object^ sender, EventArgs^ e )
{
    System::Collections::IEnumerator^ myEnum2 = dataGridView1->Columns->GetEnumerator();
    while ( myEnum2->MoveNext() )
    {
        DataGridViewImageColumn^ column = safe_cast<DataGridViewImageColumn^>(myEnum2->Current);
        column->ImageLayout = DataGridViewCellLayout::Normal;
        column->Description = L"Normal";
    }
}

```

```

void MakeCellsLarger( Object^ sender, EventArgs^ e )
{
    System::Collections::IEnumerator^ myEnum3 = dataGridView1->Columns->GetEnumerator();
    while ( myEnum3->MoveNext() )
    {
        DataGridViewImageColumn^ column = safe_cast<DataGridViewImageColumn^>(myEnum3->Current);
        column->Width = column->Width * 2;
    }

    System::Collections::IEnumerable^ temp = safe_cast<System::Collections::IEnumerable^>(dataGridView1-
>Rows);
    System::Collections::IEnumerator^ myEnum4 = temp->GetEnumerator();

    //System::Collections::IEnumerator^ myEnum4 = dataGridView1->Rows->GetEnumerator();
    while ( myEnum4->MoveNext() )
    {
        DataGridViewRow^ row = safe_cast<DataGridViewRow^>(myEnum4->Current);
        if ( row->IsNewRow )
            break;
        row->Height = (int)(row->Height * 1.5);
    }
}

bool IsAWin( DataGridViewCell^ cell )
{
    if ( ARowIsSame() || AColumnIsSame() || ADiagonalIsSame() )
        return true;
    else
        return false;
}

bool ARowIsSame()
{
    Bitmap^ marking = nullptr;
    System::Collections::IEnumerable^ temp = safe_cast<System::Collections::IEnumerable^>(dataGridView1-
>Rows);
    System::Collections::IEnumerator^ myEnum5 = temp->GetEnumerator();
    //System::Collections::IEnumerator^ myEnum5 = dataGridView1->Rows->GetEnumerator();
    while ( myEnum5->MoveNext() )
    {
        DataGridViewRow^ row = safe_cast<DataGridViewRow^>(myEnum5->Current);
        if ( row->IsNewRow )
            break;

        marking = dynamic_cast<Bitmap^>(row->Cells[ 0 ]->Value);
        if ( marking != blank )
        {
            if ( marking == row->Cells[ 1 ]->Value && marking == row->Cells[ 2 ]->Value )
                return true;
        }
    }

    return false;
}

bool AColumnIsSame()
{
    int columnIndex = 0;
    Bitmap^ marking;
    do
    {
        marking = dynamic_cast<Bitmap^>(dataGridView1->Rows[ 0 ]->Cells[ columnIndex ]->Value);
        if ( marking != blank )
        {
            if ( marking == dynamic_cast<Bitmap^>(dataGridView1->Rows[ 1 ]->Cells[ columnIndex ]->Value) &&
marking == dynamic_cast<Bitmap^>(dataGridView1->Rows[ 2 ]->Cells[ columnIndex ]->Value) )
                return true;
        }
    }
}

```

```

        columnIndex = columnIndex + 1;
    }
    while ( columnIndex < dataGridView1->Columns->GetColumnCount( DataGridViewElementStates::Visible ) );

    return false;
}

bool ADiagonalIsSame()
{
    if ( LeftToRightDiagonalIsSame() )
    {
        return true;
    }

    if ( RightToLeftDiagonalIsSame() )
    {
        return true;
    }

    return false;
}

bool LeftToRightDiagonalIsSame()
{
    return IsDiagonalSame( 0, 2 );
}

bool RightToLeftDiagonalIsSame()
{
    return IsDiagonalSame( 2, 0 );
}

bool IsDiagonalSame( int startingColumn, int lastColumn )
{
    Bitmap^ marking = dynamic_cast<Bitmap^>(dataGridView1->Rows[ 0 ]->Cells[ startingColumn ]->Value);
    if ( marking == blank )
        return false;

    if ( marking == dataGridView1->Rows[ 1 ]->Cells[ 1 ]->Value && marking == dataGridView1->Rows[ 2 ]-
>Cells[ lastColumn ]->Value )
        return true;

    return false;
}

void ToggleTurn()
{
    if ( turn->Text->Equals( xString ) )
    {
        turn->Text = oString;
    }
    else
    {
        turn->Text = xString;
    }
}

bool IsOsTurn()
{
    if ( turn->Text->Equals( oString ) )
        return true;

    return false;
}

public:
    [STAThread]

```

```

L:\TicTacToe\

static void Main()
{
    Application::Run( gcnew TicTacToe );
}

};

int main()
{
    TicTacToe::Main();
}

```

```

using System.IO;
using System.Windows.Forms;
using System.Drawing;
using System;

public class TicTacToe : System.Windows.Forms.Form
{
    public TicTacToe()
    {
        blank = new Bitmap(new MemoryStream(blankImage));
        x = new Bitmap(new MemoryStream(xImage));
        o = new Bitmap(new MemoryStream(oImage));

        this.AutoSize = true;
        SetupButtons();
        InitializeDataGridView(null, null);
    }

    private DataGridView dataGridView1;
    private Button Button1 = new Button();
    private Label turn = new Label();
    private Button Button2 = new Button();
    private Button Button3 = new Button();
    private Button Button4 = new Button();
    private Button Button5 = new Button();
    private FlowLayoutPanel Panel1 = new FlowLayoutPanel();

    #region "bitmaps"
    private byte[] oImage = new byte[] {
        0x42, 0x4D, 0xC6, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x76,
        0x0, 0x0, 0x0, 0x28, 0x0, 0x0, 0xB, 0x0, 0x0, 0x0, 0xA,
        0x0, 0x0, 0x0, 0x1, 0x0, 0x4, 0x0, 0x0, 0x0, 0x0, 0x0, 0x50,
        0x0, 0x10,
        0x0, 0x0, 0x0, 0x10, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
        0x0, 0x80, 0x0, 0x80, 0x0, 0x0, 0x0, 0x80, 0x0, 0x80, 0x0,
        0x80, 0x0, 0x0, 0x80, 0x0, 0x0, 0x80, 0x0, 0x80, 0x0, 0x0,
        0x0, 0xC0, 0xC0, 0xC0, 0x0, 0x80, 0x80, 0x80, 0x0, 0x0, 0x0,
        0xFF, 0x0, 0x0, 0xFF, 0x0, 0x0, 0xFF, 0x0, 0xFF, 0x0, 0xFF,
        0x0, 0x0, 0x0, 0xFF, 0x0, 0x0, 0xFF, 0x0, 0xFF, 0x0, 0x0,
        0xFF, 0xFF, 0x0, 0xFF, 0x0, 0x0, 0xF, 0x0, 0xFF, 0x0, 0xF0,
        0x0, 0x0, 0xFF, 0x0, 0xF0, 0x0, 0xF, 0x0, 0xF0, 0x0, 0x0,
        0xFF, 0xFF, 0xF0, 0x0, 0x0, 0x0, 0xF0, 0x0, 0x0, 0x0, 0x0,
        0x0, 0x0, 0x0, 0xF, 0xFF, 0x0, 0xFF, 0x0, 0x0, 0x0,
        0xF0, 0xFF, 0x0, 0xF0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
        0xFF, 0xFF, 0xF0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
        0xF, 0xF0, 0x0, 0x0, 0xFF, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
        0x0
    };

    private byte[] xImage = new byte[]{
        0x42, 0x4D, 0xC6, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
        0x0, 0x0, 0x0, 0x76, 0x0, 0x0, 0x0, 0x28, 0x0, 0x0, 0x0,
        0xB, 0x0, 0x0, 0x0, 0xA, 0x0, 0x0, 0x0, 0x1, 0x0, 0x4, 0x0,
        0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
    };

```

```
    0x0, 0x0, 0x0, 0x0, 0x10, 0x0, 0x0, 0x10, 0x0, 0x0, 0x0,
    0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x80, 0x0, 0x0, 0x80,
    0x0, 0x0, 0x0, 0x80, 0x80, 0x0, 0x80, 0x0, 0x0, 0x0, 0x80,
    0x0, 0x80, 0x0, 0x80, 0x80, 0x0, 0x0, 0xC0, 0x0C0, 0x0C0, 0x0,
    0x80, 0x80, 0x80, 0x0, 0x0, 0x0, 0xFF, 0x0, 0x0, 0xFF, 0x0,
    0x0, 0x0, 0xFF, 0xFF, 0x0, 0xFF, 0x0, 0x0, 0x0, 0xFF, 0x0,
    0xFF, 0x0, 0xFF, 0xFF, 0x0, 0xF0, 0x0F0, 0x0, 0x0, 0xFF, 0xF,
    0xFF, 0xFF, 0xF0, 0x0F0, 0x0, 0x0, 0xFF, 0xF0, 0xFF, 0xF0,
    0xFF, 0xF0, 0x0, 0x0, 0xFF, 0xFF, 0xF, 0xF, 0xFF, 0xF0, 0x0,
    0x0, 0xFF, 0xFF, 0xF, 0xF, 0xFF, 0xF0, 0x0, 0x0, 0xFF, 0xFF,
    0xF, 0xF, 0xFF, 0xF0, 0x0, 0x0, 0xFF, 0xF0, 0xFF, 0xF0,
    0xFF, 0xF0, 0x0, 0x0, 0xFF, 0xF, 0xFF, 0xF0, 0xFF, 0xF, 0xF0,
    0x0, 0xF0, 0xFF, 0xFF, 0xF0, 0x0, 0x0, 0x0, 0xFF, 0x0};
```

```

private byte[] blankImage = new byte[] {
    0x42, 0x4D, 0xC6, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x76,
    0x0, 0x0, 0x0, 0x28, 0x0, 0x0, 0xB, 0x0, 0x0, 0x0, 0xA,
    0x0, 0x0, 0x0, 0x1, 0x0, 0x4, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x50,
    0x0, 0x10,
    0x0, 0x0, 0x0, 0x10, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
    0x0, 0x80, 0x0, 0x0, 0x80, 0x0, 0x0, 0x0, 0x80, 0x80, 0x0,
    0x80, 0x0, 0x0, 0x80, 0x0, 0x80, 0x0, 0x80, 0x80, 0x0,
    0x0, 0xC0, 0xC0, 0xC0, 0x0, 0x80, 0x80, 0x80, 0x0, 0x0, 0x0,
    0xFF, 0x0, 0x0, 0xFF, 0x0, 0x0, 0x0, 0xFF, 0xFF, 0x0,
    0x0, 0x0, 0x0, 0xFF, 0x0, 0xFF, 0x0, 0xFF, 0xFF, 0x0,
    0x0, 0x0, 0xFF, 0xFF, 0x0, 0xFF, 0x0, 0xF0, 0x0, 0x0,
    0xFF, 0xFF, 0xFF, 0xF0, 0x0, 0x0, 0xFF, 0xFF, 0xFF,
    0xFF, 0xF0, 0x0, 0x0, 0xFF, 0xFF, 0xFF, 0x0, 0xF0, 0x0,
    0x0, 0x0, 0xFF, 0xFF, 0xFF, 0xFF, 0x0, 0xF0, 0x0, 0x0,
    0xFF, 0xFF, 0xFF, 0xF0, 0x0, 0x0, 0x0, 0xFF, 0xFF, 0xFF,
    0xFF, 0xF0, 0x0, 0x0, 0x0};
}

#endifregion

```

```
private Bitmap blank;
private Bitmap x;
private Bitmap o;
private string xString = "X's turn";
private string oString = "O's turn";
private string gameOverString = "Game Over";
private int bitmapPadding = 6;
```

```
private void InitializeDataGridView(object sender,
    EventArgs e)
{
    this.Panel1.SuspendLayout();
    this.SuspendLayout();

    ConfigureForm();
    SizeGrid();
    CreateColumns();
    CreateRows();

    this.Panel1.ResumeLayout(false);
    this.ResumeLayout(false);
}
```

```
private void ConfigureForm()
{
    AutoSize = true;
    turn.Size = new System.Drawing.Size(75, 34);
    turn.TextAlign = ContentAlignment.MiddleLeft;
    turn.Text = xString;
```

```

        Panel1.Location = new System.Drawing.Point(0, 0);
        Panel1.Size = new System.Drawing.Size(120, 196);
        Panel1.FlowDirection = FlowDirection.TopDown;

        ClientSize = new System.Drawing.Size(355, 200);
        Controls.Add(this.Panel1);
        Text = "TicTacToe";

        dataGridView1 = new System.Windows.Forms.DataGridView();
        dataGridView1.Location = new Point(120, 0);
        dataGridView1.AllowUserToAddRows = false;
        dataGridView1.CellClick += new
            DataGridViewCellEventHandler(dataGridView1_CellClick);
        dataGridView1.CellMouseEnter += new
            DataGridViewCellEventHandler(dataGridView1_CellMouseEnter);
        dataGridView1.CellMouseLeave += new
            DataGridViewCellEventHandler(dataGridView1_CellMouseLeave);

        Controls.Add(dataGridView1);
    }

    private void SetupButtons()
    {
        Button1.AutoSize = true;
        SetupButton(Button1, "Restart", new EventHandler(Reset));
        Panel1.Controls.Add(turn);
        SetupButton(Button2, "Increase Cell Size", new EventHandler(MakeCellsLarger));
        SetupButton(Button3, "Stretch Images", new EventHandler(Stretch));
        SetupButton(Button4, "Zoom Images", new EventHandler(ZoomToImage));
        SetupButton(Button5, "Normal Images", new EventHandler(NormalImage));
    }

    private void SetupButton(Button button, string buttonLabel, EventHandler handler)
    {
        Panel1.Controls.Add(button);
        button.Text = buttonLabel;
        button.AutoSize = true;
        button.Click += handler;
    }

    private void CreateColumns()
    {
        DataGridViewImageColumn imageColumn;
        int columnCount = 0;
        do
        {
            Bitmap unMarked = blank;
            imageColumn = new DataGridViewImageColumn();

            //Add twice the padding for the left and
            //right sides of the cell.
            imageColumn.Width = x.Width + 2 * bitmapPadding + 1;

            imageColumn.Image = unMarked;
            dataGridView1.Columns.Add(imageColumn);
            columnCount = columnCount + 1;
        }
        while (columnCount < 3);
    }

    private void CreateRows()
    {
        dataGridView1.Rows.Add();
        dataGridView1.Rows.Add();
        dataGridView1.Rows.Add();
    }

    private void SizeGrid()
    {

```

```

dataGridView1.ColumnHeadersVisible = false;
dataGridView1.RowHeadersVisible = false;
dataGridView1.AllowUserToResizeColumns = false;;
dataGridView1.AllowUserToResizeRows = false;
dataGridView1.BorderStyle = BorderStyle.None;

//Add twice the padding for the top of the cell
//and the bottom.
dataGridView1.RowTemplate.Height = x.Height +
    2 * bitmapPadding + 1;

dataGridView1.AutoSize = true;
}

private void Reset(object sender, System.EventArgs e)
{
    dataGridView1.Dispose();
    InitializeDataGridView(null, null);
}

private void dataGridView1_CellClick(object sender,
    DataGridViewCellEventArgs e)
{
    if (turn.Text.Equals(gameOverString)) { return; }

    DataGridViewImageCell cell = (DataGridViewImageCell)
        dataGridView1.Rows[e.RowIndex].Cells[e.ColumnIndex];

    if (cell.Value == blank)
    {
        if (IsOsTurn())
        {
            cell.Value = o;
        }
        else
        {
            cell.Value = x;
        }
        ToggleTurn();
    }
    if (IsAWin())
    {
        turn.Text = gameOverString;
    }
}

private void dataGridView1_CellMouseEnter(object sender,
    DataGridViewCellEventArgs e)
{
    Bitmap markingUnderMouse = (Bitmap)dataGridView1.
        Rows[e.RowIndex].
        Cells[e.ColumnIndex].Value;

    if (markingUnderMouse == blank)
    {
        dataGridView1.Cursor = Cursors.Default;
    }
    else if (markingUnderMouse == o || markingUnderMouse == x)
    {
        dataGridView1.Cursor = Cursors.No;
        ToolTip(e, true);
    }
}

private void ToolTip(DataGridViewCellEventArgs e, bool showTip)
{
    DataGridViewImageCell cell = (DataGridViewImageCell)
        dataGridView1

```

```

    .Rows[e.RowIndex].Cells[e.ColumnIndex];
    DataGridViewImageColumn imageColumn =
        (DataGridViewImageColumn)
        dataGridView1.Columns[cell.ColumnIndex];

    if (showTip) cell.ToolTipText = imageColumn.Description;
    else { cell.ToolTipText = String.Empty; }
}

private void dataGridView1_CellMouseLeave(object sender,
    DataGridViewCellEventArgs e)
{
    ToolTip(e, false);
    dataGridView1.Cursor = Cursors.Default;
}

private void Stretch(object sender, EventArgs e)
{
    foreach (DataGridViewImageColumn column in
        dataGridView1.Columns)
    {
        column.ImageLayout = DataGridViewImageCellLayout.Stretch;
        column.Description = "Stretched";
    }
}

private void ZoomToImage(object sender, EventArgs e)
{
    foreach (DataGridViewImageColumn column in
        dataGridView1.Columns)
    {
        column.ImageLayout = DataGridViewImageCellLayout.Zoom;
        column.Description = "Zoomed";
    }
}

private void NormalImage(object sender, EventArgs e)
{
    foreach (DataGridViewImageColumn column in
        dataGridView1.Columns)
    {
        column.ImageLayout = DataGridViewImageCellLayout.Normal;
        column.Description = "Normal";
    }
}

private void MakeCellsLarger(object sender, EventArgs e)
{
    foreach (DataGridViewImageColumn column in
        dataGridView1.Columns)
    {
        column.Width = column.Width * 2;
    }
    foreach (DataGridViewRow row in dataGridView1.Rows)
    {
        if (row.IsNewRow) break;
        row.Height = (int)(row.Height * 1.5);
    }
}

private bool IsAWin()
{
    if (ARowIsSame() || AColumnIsSame() || ADiagonalIsSame())
        return true;
    else return false;
}

```

```

private bool ARowIsSame()
{
    Bitmap marking = null;

    foreach (DataGridViewRow row in dataGridView1.Rows)
    {
        if (row.IsNewRow) break;
        marking = (Bitmap)row.Cells[0].Value;
        if (marking != blank)
        {
            if (marking == row.Cells[1].Value &&
                marking == row.Cells[2].Value) return true;
        }
    }
    return false;
}

private bool AColumnIsSame()
{
    int columnIndex = 0;
    Bitmap marking;
    do
    {
        marking = (Bitmap)
            dataGridView1.Rows[0].Cells[columnIndex].Value;
        if (marking != blank)
        {
            if (marking == (Bitmap)dataGridView1.Rows[1]
                .Cells[columnIndex].Value
                && marking == (Bitmap)dataGridView1.Rows[2].
                Cells[columnIndex].Value) return true;
        }
        columnIndex = columnIndex + 1;
    }
    while (columnIndex < dataGridView1.Columns.GetColumnCount(
        DataGridViewElementStates.Visible));
    return false;
}

private bool ADiagonalIsSame()
{
    if (LeftToRightDiagonalIsSame()) { return true; }
    if (RightToLeftDiagonalIsSame()) { return true; }
    return false;
}

private bool LeftToRightDiagonalIsSame()
{
    return IsDiagonalSame(0, 2);
}

private bool RightToLeftDiagonalIsSame()
{
    return IsDiagonalSame(2, 0);
}

private bool IsDiagonalSame(int startingColumn, int lastColumn)
{
    Bitmap marking = (Bitmap)dataGridView1.Rows[0]
        .Cells[startingColumn].Value;

    if (marking == blank) return false;
    if (marking == dataGridView1.Rows[1].Cells[1]
        .Value && marking == dataGridView1.Rows[2]
        .Cells[lastColumn].Value) return true;

    return false;
}

```

```

private void ToggleTurn()
{
    if (turn.Text.Equals(xString)) { turn.Text = oString; }
    else { turn.Text = xString; }
}

private bool IsOsTurn()
{
    if (turn.Text.Equals(oString)) return true;
    return false;
}

[STAThread]
public static void Main()
{
    Application.Run(new TicTacToe());
}

```

```

Imports System.IO
Imports System.Windows.Forms
Imports System.Drawing
Imports System

Public Class TicTacToe
    Inherits System.Windows.Forms.Form

    Friend WithEvents dataGridView1 As DataGridView
    Friend WithEvents Button1 As Button = New Button()
    Friend WithEvents turn As Label = New Label()
    Friend WithEvents Button2 As Button = New Button()
    Friend WithEvents Button3 As Button = New Button()
    Friend WithEvents Button4 As Button = New Button()
    Friend WithEvents Button5 As Button = New Button()
    Friend WithEvents Panel1 As FlowLayoutPanel()

#Region "bitmaps"
Private oImage As Byte() = {
    &H42, &H4D, &HC6, &H0, &H0, &H0, &H0, &H0, &H0, &H0,
    &H0, &H0, &H76, &H0, &H0, &H28, &H0, &H0,
    &H0, &H0, &HB, &H0, &H0, &HA, &H0, &H0,
    &H0, &H0, &H1, &H0, &H4, &H0, &H0, &H0, &H0,
    &H0, &H0, &H50, &H0, &H0, &H0, &H0, &H0, &H0,
    &H0, &H0, &H0, &H0, &H0, &H10, &H0, &H0,
    &H0, &H0, &H10, &H0, &H0, &H0, &H0, &H0, &H0,
    &H0, &H0, &H0, &H80, &H0, &H0, &H80, &H0,
    &H0, &H0, &H80, &H80, &H0, &H80, &H0, &H0,
    &H0, &H0, &HC0, &HC0, &HC0, &H0, &H80, &H80,
    &H80, &H0, &H0, &HFF, &H0, &H0, &HFF, &H0,
    &H0, &H0, &HFF, &HFF, &H0, &HFF, &H0, &HFF,
    &H0, &H0, &HFF, &HFF, &H0, &HFF, &HFF, &H0,
    &H0, &HF, &HFF, &HF0, &H0, &H0, &HFF, &H0,
    &HFF, &HF0, &HF0, &H0, &H0, &HF0, &HFF, &HFF,
    &H0, &HF, &HFF, &HF0, &H0, &H0}
}

Private xImage As Byte() = {
    &H42, &H4D, &HC6, &H0, &H0, &H0, &H0, &H0, &H0,
```



```

CreateColumns()
CreateRows()

ResumeLayout(False)
Panel1.ResumeLayout(False)
End Sub

Private Sub ConfigureForm()
    AutoSize = True
    turn.Size = New System.Drawing.Size(75, 34)
    turn.TextAlign = ContentAlignment.MiddleLeft
    turn.Text = xString

    Panel1.FlowDirection = FlowDirection.TopDown
    Panel1.Location = New System.Drawing.Point(0, 8)
    Panel1.Size = New System.Drawing.Size(120, 196)

    ClientSize = New System.Drawing.Size(355, 200)
    Controls.Add(Me.Panel1)
    Text = "TicTacToe"

    dataGridView1 = New System.Windows.Forms.DataGridView
    dataGridView1.Location = New Point(120, 0)
    dataGridView1.AllowUserToAddRows = False
    Controls.Add(dataGridView1)
    SetupButtons()
End Sub

Private Sub SetupButtons()
    SetupButton(Button1, "Restart")
    Panel1.Controls.Add(Me.turn)
    SetupButton(Button2, "Increase Cell Size")
    SetupButton(Button3, "Stretch Images")
    SetupButton(Button4, "Zoom Images")
    SetupButton(Button5, "Normal Images")
End Sub

Private Sub SetupButton(ByVal button As Button, _
ByVal buttonLabel As String)

    Panel1.Controls.Add(button)
    button.Text = buttonLabel
    button.AutoSize = True
End Sub

End Sub

Private Sub CreateColumns()

    Dim imageColumn As DataGridViewImageColumn
    Dim columnCount As Integer = 0
    Do
        Dim unMarked As Bitmap = blank
        imageColumn = New DataGridViewImageColumn()

        ' Add twice the padding for the left and
        ' right sides of the cell.
        imageColumn.Width = x.Width + 2 * bitmapPadding + 1

        imageColumn.Image = unMarked
        imageColumn.ImageLayout = DataGridViewImageCellLayout.NotSet
        imageColumn.Description = "default image layout"
        dataGridView1.Columns.Add(imageColumn)
        columnCount = columnCount + 1
    Loop While columnCount < 3
End Sub

Private Sub CreateRows()
    dataGridView1.Rows.Add()
    dataGridView1.Rows.Add()

```

```

    dataGridView1.Rows.Add()
End Sub

Private Sub SizeGrid()
    dataGridView1.ColumnHeadersVisible = False
    dataGridView1.RowHeadersVisible = False
    dataGridView1.AllowUserToResizeColumns = False
    dataGridView1.AllowUserToResizeRows = False
    dataGridView1.BorderStyle = BorderStyle.None

    ' Add twice the padding for the top and bottom of the cell.
    dataGridView1.RowTemplate.Height = x.Height + _
        2 * bitmapPadding + 1

    dataGridView1.AutoSize = True
End Sub

Private Sub reset(ByVal sender As Object, _
    ByVal e As EventArgs) _
    Handles Button1.Click

    dataGridView1.Dispose()
    InitializeDataGridView(Nothing, Nothing)
End Sub

Private Sub dataGridView1_CellClick(ByVal sender As Object, _
    ByVal e As DataGridViewCellEventArgs) _
    Handles dataGridView1.CellClick

    If turn.Text.Equals(gameOverString) Then Return

    Dim cell As DataGridViewImageCell = _
        CType(dataGridView1.Rows(e.RowIndex). _
            Cells(e.ColumnIndex), DataGridViewImageCell)
    If (cell.Value Is blank) Then
        If IsOsTurn() Then
            cell.Value = o
        Else
            cell.Value = x
        End If
        ToggleTurn()
        ToolTip(e)
    End If
    If IsAWin() Then
        turn.Text = gameOverString
    End If
End Sub

Private Sub dataGridView1_CellMouseEnter(ByVal sender As Object, _
    ByVal e As DataGridViewCellEventArgs) _
    Handles dataGridView1.CellMouseEnter

    Dim markingUnderMouse As Bitmap = _
        CType(dataGridView1.Rows(e.RowIndex). _
            Cells(e.ColumnIndex).Value, Bitmap)

    If markingUnderMouse Is blank Then
        dataGridView1.Cursor = Cursors.Default
    ElseIf markingUnderMouse Is o OrElse markingUnderMouse Is x Then
        dataGridView1.Cursor = Cursors.No
        ToolTip(e)
    End If
End Sub

Private Sub ToolTip( _
    ByVal e As DataGridViewCellEventArgs)

    Dim cell As DataGridViewImageCell = _
        CType(dataGridView1.Rows(e.RowIndex).Cells(e.ColumnIndex), DataGridViewImageCell)

```

```

        CType(dataGridView1.Rows(e.RowIndex), _
        Cells(e.ColumnIndex), DataGridViewImageCell)
    Dim imageColumn As DataGridViewImageColumn = _
        CType(dataGridView1.Columns(cell.ColumnIndex), _
        DataGridViewImageColumn)

    cell.ToolTipText = imageColumn.Description
End Sub

Private Sub dataGridView1_CellMouseLeave(ByVal sender As Object, _
    ByVal e As DataGridViewCellEventArgs) _
Handles dataGridView1.CellMouseLeave

    dataGridView1.Cursor = Cursors.Default
End Sub

Private Sub Stretch(ByVal sender As Object, _
    ByVal e As EventArgs) Handles Button3.Click

    For Each column As DataGridViewImageColumn _
        In dataGridView1.Columns
        column.ImageLayout = DataGridViewImageCellLayout.Stretch
        column.Description = "Stretched image layout"
    Next
End Sub

Private Sub ZoomToImage(ByVal sender As Object, _
    ByVal e As EventArgs) Handles Button4.Click

    For Each column As DataGridViewImageColumn _
        In dataGridView1.Columns
        column.ImageLayout = DataGridViewImageCellLayout.Zoom
        column.Description = "Zoomed image layout"
    Next
End Sub

Private Sub NormalImage(ByVal sender As Object, _
    ByVal e As EventArgs) Handles Button5.Click

    For Each column As DataGridViewImageColumn _
        In dataGridView1.Columns
        column.ImageLayout = DataGridViewImageCellLayout.Normal
        column.Description = "Normal image layout"
    Next
End Sub

Private Sub MakeCellsLarger(ByVal sender As Object, _
    ByVal e As EventArgs) Handles Button2.Click

    For Each column As DataGridViewImageColumn _
        In dataGridView1.Columns
        column.Width = column.Width * 2
    Next
    For Each row As DataGridViewRow In dataGridView1.Rows
        If row.IsNewRow Then Continue For
        row.Height = CInt(row.Height * 1.5)
    Next
End Sub

Private Function IsAWin() As Boolean
    If ARowIsSame() OrElse AColumnIsSame() OrElse ADiagonalIsSame() Then
        Return True
    Else
        Return False
    End If
End Function

Private Function ARowIsSame() As Boolean
    Dim marking As Bitmap = Nothing
    ToMarking(marking, dataGridView1)
    For i = 0 To dataGridView1.Rows.Count - 1
        If marking(i, 0).AValue <> marking(i, 1).AValue Then
            Return False
        End If
    Next
    Return True
End Function

Private Function AColumnIsSame() As Boolean
    Dim marking As Bitmap = Nothing
    ToMarking(marking, dataGridView1)
    For i = 0 To dataGridView1.Rows.Count - 1
        If marking(0, i).AValue <> marking(1, i).AValue Then
            Return False
        End If
    Next
    Return True
End Function

Private Function ADiagonalIsSame() As Boolean
    Dim marking As Bitmap = Nothing
    ToMarking(marking, dataGridView1)
    For i = 0 To dataGridView1.Rows.Count - 1
        If marking(i, 0).AValue <> marking(i, 1).AValue Then
            Return False
        End If
    Next
    Return True
End Function

```

```

    If marking Is blank Then Return False
    For Each row As DataGridViewRow In dataGridView1.Rows
        If row.IsNewRow Then Continue For
        marking = CType(row.Cells(0).Value, Bitmap)
        If marking IsNot blank Then
            If marking Is row.Cells(1).Value AndAlso _
                marking Is row.Cells(2).Value Then Return True
        End If
    Next
    Return False
End Function

Private Function AColumnIsSame() As Boolean

    Dim columnIndex As Integer = 0
    Dim marking As Bitmap
    Do
        marking = CType(dataGridView1.Rows(0).Cells(columnIndex).Value, _
            Bitmap)
        If marking IsNot blank Then
            If marking Is _
                dataGridView1.Rows(1).Cells(columnIndex).Value _
                AndAlso marking Is _
                dataGridView1.Rows(2).Cells(columnIndex).Value _
                Then Return True
        End If
        columnIndex = columnIndex + 1
    Loop While columnIndex < _
        dataGridView1.Columns.GetColumnCount( _
        DataGridViewElementStates.Visible)
    Return False
End Function

Private Function ADiagonalIsSame() As Boolean

    If LeftToRightDiagonalIsSame() Then Return True
    If RightToLeftDiagonalIsSame() Then Return True
    Return False
End Function

Private Function LeftToRightDiagonalIsSame() As Boolean
    Return IsDiagonalSame(0, 2)
End Function

Private Function RightToLeftDiagonalIsSame() As Boolean
    Return IsDiagonalSame(2, 0)
End Function

Private Function IsDiagonalSame(ByVal startingColumn As Integer, _
    ByVal lastColumn As Integer) As Boolean

    Dim marking As Bitmap = CType( _
        dataGridView1.Rows(0).Cells(startingColumn).Value, Bitmap)
    If marking Is blank Then Return False
    If marking Is dataGridView1.Rows(1).Cells(1).Value AndAlso _
        marking Is dataGridView1.Rows(2).Cells(lastColumn).Value _
        Then Return True
End Function

Private Sub ToggleTurn()
    If turn.Text.Equals(xString) Then turn.Text = oString _
    Else turn.Text = xString
End Sub

Private Function IsOsTurn() As Boolean
    If turn.Text.Equals(oString) Then Return True _
    Else : Return False
End Function

```

```
<STAThread> Public Shared Sub Main()
    Application.Run(New TicTacToe())
End Sub

End Class
```

Compiling the Code

This example requires:

- References to the System and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[DataGridView](#)

[DataGridViewImageColumn](#)

[Programming with Cells, Rows, and Columns in the Windows Forms DataGridView Control](#)

[How to: Display Images in Cells of the Windows Forms DataGridView Control](#)

Customizing the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

The `DataGridView` control provides several properties that you can use to adjust the appearance and basic behavior (look and feel) of its cells, rows, and columns. If you have special needs that go beyond the capabilities of the `DataGridViewCellStyle` class, however, you can also implement owner drawing for the control or extend its capabilities by creating custom cells, columns, and rows.

To paint cells and rows yourself, you can handle various `DataGridView` painting events. To modify existing functionality or provide new functionality, you can create your own types derived from the existing `DataGridViewCell`, `DataGridViewColumn`, and `DataGridViewRow` types. You can also provide new editing capabilities by creating derived types that display a control of your choosing when a cell is in edit mode.

In This Section

[How to: Customize the Appearance of Cells in the Windows Forms DataGridView Control](#)

Describes how to handle the `CellPainting` event in order to paint cells manually.

[How to: Customize the Appearance of Rows in the Windows Forms DataGridView Control](#)

Describes how to handle the `RowPrePaint` and `RowPostPaint` events in order to paint rows with a custom, gradient background and content that spans multiple columns.

[How to: Customize Cells and Columns in the Windows Forms DataGridView Control by Extending Their Behavior and Appearance](#)

Describes how to create custom types derived from `DataGridViewCell` and `DataGridViewColumn` in order to highlight cells when the mouse pointer rests on them.

[How to: Disable Buttons in a Button Column in the Windows Forms DataGridView Control](#)

Describes how to create custom types derived from `DataGridViewButtonCell` and `DataGridViewButtonColumn` in order to display disabled buttons in a button column.

[How to: Host Controls in Windows Forms DataGridView Cells](#)

Describes how to implement the `IDataGridViewEditingControl` interface and create custom types derived from `DataGridViewCell` and `DataGridViewColumn` in order to display a `DateTimePicker` control when a cell is in edit mode.

Reference

[DataGridView](#)

Provides reference documentation for the `DataGridView` control.

[DataGridViewCell](#)

Provides reference documentation for the `DataGridViewCell` class.

[DataGridViewRow](#)

Provides reference documentation for the `DataGridViewRow` class.

[DataGridViewColumn](#)

Provides reference documentation for the `DataGridViewColumn` class.

[IDataGridViewEditingControl](#)

Provides reference documentation for the [IDataGridViewEditingControl](#) interface.

Related Sections

[Basic Formatting and Styling in the Windows Forms DataGridView Control](#)

Provides topics that describe how to modify the basic appearance of the control and the display formatting of cell data.

See Also

[DataGridView Control](#)

[Column Types in the Windows Forms DataGridView Control](#)

How to: Customize the Appearance of Cells in the Windows Forms DataGridView Control

5/4/2018 • 2 min to read • [Edit Online](#)

You can customize the appearance of any cell by handling the [DataGridView](#) control's [CellPainting](#) event. You can extract the [DataGridView](#) control's [Graphics](#) from the [Graphics](#) property of the [DataGridViewCellPaintingEventArgs](#). With this [Graphics](#), you can affect the appearance of the entire [DataGridView](#) control, but you will usually want to affect only the appearance of the cell that is currently being painted. The [ClipBounds](#) property of the [DataGridViewCellPaintingEventArgs](#) enables you to restrict your painting operations to the cell that is currently being painted.

In the following code example, you will paint all the cells in a `ContactName` column using the [DataGridView](#) control's color scheme. Each cell's text content is painted in [Crimson](#), and an inset rectangle is drawn in the same color as the [DataGridView](#) control's [GridColor](#) property.

Example

```
private void dataGridView1_CellPainting(object sender,
System.Windows.Forms.DataGridViewCellPaintingEventArgs e)
{
    if (this.dataGridView1.Columns["ContactName"].Index ==
        e.ColumnIndex && e.RowIndex >= 0)
    {
        Rectangle newRect = new Rectangle(e.CellBounds.X + 1,
            e.CellBounds.Y + 1, e.CellBounds.Width - 4,
            e.CellBounds.Height - 4);

        using (
            Brush gridBrush = new SolidBrush(this.dataGridView1.GridColor),
            backColorBrush = new SolidBrush(eCellStyle.BackColor))
        {
            using (Pen gridLinePen = new Pen(gridBrush))
            {
                // Erase the cell.
                e.Graphics.FillRectangle(backColorBrush, e.CellBounds);

                // Draw the grid lines (only the right and bottom lines;
                // DataGridView takes care of the others).
                e.Graphics.DrawLine(gridLinePen, e.CellBounds.Left,
                    e.CellBounds.Bottom - 1, e.CellBounds.Right - 1,
                    e.CellBounds.Bottom - 1);
                e.Graphics.DrawLine(gridLinePen, e.CellBounds.Right - 1,
                    e.CellBounds.Top, e.CellBounds.Right - 1,
                    e.CellBounds.Bottom);

                // Draw the inset highlight box.
                e.Graphics.DrawRectangle(Pens.Blue, newRect);

                // Draw the text content of the cell, ignoring alignment.
                if (e.Value != null)
                {
                    e.Graphics.DrawString((String)e.Value, eCellStyle.Font,
                        Brushes.Crimson, e.CellBounds.X + 2,
                        e.CellBounds.Y + 2, StringFormat.GenericDefault);
                }
                e.Handled = true;
            }
        }
    }
}
```

```

Private Sub dataGridView1_CellPainting(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DataGridViewCellPaintingEventArgs) _
    Handles dataGridView1.CellPainting

    If Me.dataGridView1.Columns("ContactName").Index = _
        e.ColumnIndex AndAlso e.RowIndex >= 0 Then

        Dim newRect As New Rectangle(e.CellBounds.X + 1, e.CellBounds.Y + 1, _
            e.CellBounds.Width - 4, e.CellBounds.Height - 4)
        Dim backColorBrush As New SolidBrush(eCellStyle.BackColor)
        Dim gridBrush As New SolidBrush(Me.dataGridView1.GridColor)
        Dim gridLinePen As New Pen(gridBrush)

        Try

            ' Erase the cell.
            e.Graphics.FillRectangle(backColorBrush, e.CellBounds)

            ' Draw the grid lines (only the right and bottom lines;
            ' DataGridView takes care of the others).
            e.Graphics.DrawLine(gridLinePen, e.CellBounds.Left, _
                e.CellBounds.Bottom - 1, e.CellBounds.Right - 1, _
                e.CellBounds.Bottom - 1)
            e.Graphics.DrawLine(gridLinePen, e.CellBounds.Right - 1, _
                e.CellBounds.Top, e.CellBounds.Right - 1, _
                e.CellBounds.Bottom)

            ' Draw the inset highlight box.
            e.Graphics.DrawRectangle(Pens.Blue, newRect)

            ' Draw the text content of the cell, ignoring alignment.
            If (e.Value IsNot Nothing) Then
                e.Graphics.DrawString(CStr(e.Value), eCellStyle.Font, _
                    Brushes.Crimson, e.CellBounds.X + 2, e.CellBounds.Y + 2, _
                    StringFormat.GenericDefault)
            End If
            e.Handled = True

            Finally
                gridLinePen.Dispose()
                gridBrush.Dispose()
                backColorBrush.Dispose()
            End Try

        End If

    End Sub

```

Compiling the Code

This example requires:

- A [DataGridView](#) control named `dataGridView1` with a `ContactName` column such as the one in the Customers table in the Northwind sample database.
- References to the System, System.Windows.Forms, and System.Drawing assemblies.

See Also

[DataGridView](#)

[CellPainting](#)

[Customizing the Windows Forms DataGridView Control](#)

How to: Customize the Appearance of Rows in the Windows Forms DataGridView Control

5/4/2018 • 9 min to read • [Edit Online](#)

You can control the appearance of [DataGridView](#) rows by handling one or both of the [DataGridView.RowPrePaint](#) and [DataGridView.RowPostPaint](#) events. These events are designed so that you can paint only what you want to while letting the [DataGridView](#) control paint the rest. For example, if you want to paint a custom background, you can handle the [DataGridView.RowPrePaint](#) event and let the individual cells paint their own foreground content. Alternately, you can let the cells paint themselves and add custom foreground content in a handler for the [DataGridView.RowPostPaint](#) event. You can also disable cell painting and paint everything yourself in a [DataGridView.RowPrePaint](#) event handler.

The following code example implements handlers for both events in order to provide a gradient selection background and some custom foreground content that spans multiple columns.

Example

```
using System;
using System.Drawing;
using System.Windows.Forms;

class DataGridViewRowPainting : Form
{
    private DataGridView dataGridView1 = new DataGridView();
    private Int32 oldRowIndex = 0;
    private const Int32 CUSTOM_CONTENT_HEIGHT = 30;

    [STAThreadAttribute()]
    public static void Main()
    {
        Application.Run(new DataGridViewRowPainting());
    }

    public DataGridViewRowPainting()
    {
        this.dataGridView1.Dock = DockStyle.Fill;
        this.Controls.Add(this.dataGridView1);
        this.Load += new EventHandler(DataGridViewRowPainting_Load);
        this.Text = "DataGridView row painting demo";
    }

    void DataGridViewRowPainting_Load(object sender, EventArgs e)
    {
        // Set a cell padding to provide space for the top of the focus
        // rectangle and for the content that spans multiple columns.
        Padding newPadding = new Padding(0, 1, 0, CUSTOM_CONTENT_HEIGHT);
        this.dataGridView1.RowTemplate.DefaultCellStyle.Padding = newPadding;

        // Set the selection background color to transparent so
        // the cell won't paint over the custom selection background.
        this.dataGridView1.RowTemplate.DefaultCellStyle.SelectionBackColor =
            Color.Transparent;

        // Set the row height to accommodate the content that
        // spans multiple columns.
        this.dataGridView1.RowTemplate.Height += CUSTOM_CONTENT_HEIGHT;
    }
}
```

```

// Initialize other DataGridView properties.
this.dataGridView1.AllowUserToAddRows = false;
this.dataGridView1.EditMode = DataGridViewEditMode.EditOnKeystrokeOrF2;
this.dataGridView1.CellBorderStyle = DataGridViewCellBorderStyle.None;
this.dataGridView1.SelectionMode =
    DataGridViewSelectionMode.FullRowSelect;

// Set the column header names.
this.dataGridView1.ColumnCount = 4;
this.dataGridView1.Columns[0].Name = "Recipe";
this.dataGridView1.Columns[0].SortMode =
    DataGridViewColumnSortMode.NotSortable;
this.dataGridView1.Columns[1].Name = "Category";
this.dataGridView1.Columns[2].Name = "Main Ingredients";
this.dataGridView1.Columns[3].Name = "Rating";

// Hide the column that contains the content that spans
// multiple columns.
this.dataGridView1.Columns[2].Visible = false;

// Populate the rows of the DataGridView.
string[] row1 = new string[]{"Meatloaf", "Main Dish",
    "1 lb. lean ground beef, 1/2 cup bread crumbs, " +
    "1/4 cup ketchup, 1/3 tsp onion powder, 1 clove of garlic, " +
    "1/2 pack onion soup mix, dash of your favorite BBQ Sauce",
    "*****"};
string[] row2 = new string[]{"Key Lime Pie", "Dessert",
    "lime juice, whipped cream, eggs, evaporated milk", "*****"};
string[] row3 = new string[]{"Orange-Salsa Pork Chops",
    "Main Dish", "pork chops, salsa, orange juice, pineapple", "*****"};
string[] row4 = new string[]{"Black Bean and Rice Salad",
    "Salad", "black beans, brown rice", "*****"};
string[] row5 = new string[]{"Chocolate Cheesecake",
    "Dessert", "cream cheese, unsweetened chocolate", "***"};
string[] row6 = new string[]{"Black Bean Dip", "Appetizer",
    "black beans, sour cream, salsa, chips", "***"};
object[] rows = new object[] { row1, row2, row3, row4, row5, row6 };
foreach (string[] rowArray in rows)
{
    this.dataGridView1.Rows.Add(rowArray);
}

// Adjust the row heights to accommodate the normal cell content.
this.dataGridView1.AutoResizeRows(
    DataGridViewAutoSizeRowsMode.AllCellsExceptHeaders);

// Attach handlers to DataGridView events.
this.dataGridView1.ColumnWidthChanged += new
    DataGridViewColumnEventHandler(dataGridView1_ColumnWidthChanged);
this.dataGridView1.RowPrePaint += new
    DataGridViewRowPrePaintEventHandler(dataGridView1_RowPrePaint);
this.dataGridView1.RowPostPaint += new
    DataGridViewRowPostPaintEventHandler(dataGridView1_RowPostPaint);
this.dataGridView1.CurrentCellChanged += new
    EventHandler(dataGridView1_CurrentCellChanged);
this.dataGridView1.RowHeightChanged += new
    DataGridViewRowEventHandler(dataGridView1_RowHeightChanged);
}

// Forces the control to repaint itself when the user
// manually changes the width of a column.
void dataGridView1_ColumnWidthChanged(object sender,
    DataGridViewColumnEventArgs e)
{
    this.dataGridView1.Invalidate();
}

// Forces the row to repaint itself when the user changes the
// current cell. This is necessary to refresh the focus rectangle.

```

```

void dataGridview1_CurrentCellChanged(object sender, EventArgs e)
{
    if (oldRowIndex != -1)
    {
        this.dataGridView1.InvalidateRow(oldRowIndex);
    }
    oldRowIndex = this.dataGridView1.CurrentCellAddress.Y;
}

// Paints the custom selection background for selected rows.
void dataGridView1_RowPrePaint(object sender,
                               DataGridViewRowPrePaintEventArgs e)
{
    // Do not automatically paint the focus rectangle.
    e.PaintParts &= ~DataGridViewPaintParts.Focus;

    // Determine whether the cell should be painted
    // with the custom selection background.
    if ((e.State & DataGridViewElementStates.Selected) ==
        DataGridViewElementStates.Selected)
    {
        // Calculate the bounds of the row.
        Rectangle rowBounds = new Rectangle(
            this.dataGridView1.RowHeadersWidth, e.RowBounds.Top,
            this.dataGridView1.Columns.GetColumnsWidth(
                DataGridViewElementStates.Visible) -
            this.dataGridView1.HorizontalScrollingOffset + 1,
            e.RowBounds.Height);

        // Paint the custom selection background.
        using (Brush backbrush =
            new System.Drawing.Drawing2D.LinearGradientBrush(rowBounds,
                this.dataGridView1.DefaultCellStyle.SelectionBackColor,
                e.InheritedRowStyle.ForeColor,
                System.Drawing.Drawing2D.LinearGradientMode.Horizontal))
        {
            e.Graphics.FillRectangle(backbrush, rowBounds);
        }
    }
}

// Paints the content that spans multiple columns and the focus rectangle.
void dataGridView1_RowPostPaint(object sender,
                               DataGridViewRowPostPaintEventArgs e)
{
    // Calculate the bounds of the row.
    Rectangle rowBounds = new Rectangle(
        this.dataGridView1.RowHeadersWidth, e.RowBounds.Top,
        this.dataGridView1.Columns.GetColumnsWidth(
            DataGridViewElementStates.Visible) -
        this.dataGridView1.HorizontalScrollingOffset + 1,
        e.RowBounds.Height);

    SolidBrush forebrush = null;
    try
    {
        // Determine the foreground color.
        if ((e.State & DataGridViewElementStates.Selected) ==
            DataGridViewElementStates.Selected)
        {
            forebrush = new SolidBrush(e.InheritedRowStyle.SelectionForeColor);
        }
        else
        {
            forebrush = new SolidBrush(e.InheritedRowStyle.ForeColor);
        }

        // Get the content that spans multiple columns.
        object recipe =

```

```

        this.dataGridView1.Rows.SharedRow(e.RowIndex).Cells[2].Value;

        if (recipe != null)
        {
            String text = recipe.ToString();

            // Calculate the bounds for the content that spans multiple
            // columns, adjusting for the horizontal scrolling position
            // and the current row height, and displaying only whole
            // lines of text.
            Rectangle textArea = rowBounds;
            textArea.X -= this.dataGridView1.HorizontalScrollingOffset;
            textArea.Width += this.dataGridView1.HorizontalScrollingOffset;
            textArea.Y += rowBounds.Height - e.InheritedRowStyle.Padding.Bottom;
            textArea.Height -= rowBounds.Height -
                e.InheritedRowStyle.Padding.Bottom;
            textArea.Height = (textArea.Height / e.InheritedRowStyle.Font.Height) *
                e.InheritedRowStyle.Font.Height;

            // Calculate the portion of the text area that needs painting.
            RectangleF clip = textArea;
            clip.Width -= this.dataGridView1.RowHeadersWidth + 1 - clip.X;
            clip.X = this.dataGridView1.RowHeadersWidth + 1;
            RectangleF oldClip = e.Graphics.ClipBounds;
            e.Graphics.SetClip(clip);

            // Draw the content that spans multiple columns.
            e.Graphics.DrawString(
                text, e.InheritedRowStyle.Font, forebrush, textArea);

            e.Graphics.SetClip(oldClip);
        }
    }
    finally
    {
        forebrush.Dispose();
    }

    if (this.dataGridView1.CurrentCellAddress.Y == e.RowIndex)
    {
        // Paint the focus rectangle.
        e.DrawFocus(rowBounds, true);
    }
}

// Adjusts the padding when the user changes the row height so that
// the normal cell content is fully displayed and any extra
// height is used for the content that spans multiple columns.
void dataGridView1_RowHeightChanged(object sender,
    DataGridViewEventArgs e)
{
    // Calculate the new height of the normal cell content.
    Int32 preferredNormalContentHeight =
        e.Row.GetPreferredHeight(e.Row.Index,
        DataGridViewAutoSizeRowMode.AllCellsExceptHeader, true) -
        e.Row.DefaultCellStyle.Padding.Bottom;

    // Specify a new padding.
    Padding newPadding = e.Row.DefaultCellStyle.Padding;
    newPadding.Bottom = e.Row.Height - preferredNormalContentHeight;
    e.Row.DefaultCellStyle.Padding = newPadding;
}
}

```

```

Imports System
Imports System.Drawing
Imports System.Windows.Forms

```

```

Class DataGridViewRowPainting
    Inherits Form
    Private WithEvents dataGridView1 As New DataGridView()
    Private oldRowIndex As Int32 = 0
    Private Const CUSTOM_CONTENT_HEIGHT As Int32 = 30

    <STAThreadAttribute()> _
    Public Shared Sub Main()

        Application.Run(New DataGridViewRowPainting())

    End Sub 'Main

    Public Sub New()

        Me.dataGridView1.Dock = DockStyle.Fill
        Me.Controls.Add(Me.dataGridView1)
        Me.Text = "DataGridView row painting demo"

    End Sub 'New

    Sub DataGridViewRowPainting_Load(ByVal sender As Object, _
        ByVal e As EventArgs) Handles Me.Load

        ' Set a cell padding to provide space for the top of the focus
        ' rectangle and for the content that spans multiple columns.
        Dim newPadding As New Padding(0, 1, 0, CUSTOM_CONTENT_HEIGHT)
        Me.dataGridView1.RowTemplate.DefaultCellStyle.Padding = newPadding

        ' Set the selection background color to transparent so
        ' the cell won't paint over the custom selection background.
        Me.dataGridView1.RowTemplate.DefaultCellStyle.SelectionBackColor = _
            Color.Transparent

        ' Set the row height to accommodate the normal cell content and the
        ' content that spans multiple columns.
        Me.dataGridView1.RowTemplate.Height += CUSTOM_CONTENT_HEIGHT

        ' Initialize other DataGridView properties.
        Me.dataGridView1.AllowUserToAddRows = False
        Me.dataGridView1.EditMode = DataGridViewEditMode.EditOnKeystrokeOrF2
        Me.dataGridView1.CellBorderStyle = DataGridViewCellBorderStyle.None
        Me.dataGridView1.SelectionMode = DataGridViewSelectionMode.FullRowSelect

        ' Set the column header names.
        Me.dataGridView1.ColumnCount = 4
        Me.dataGridView1.Columns(0).Name = "Recipe"
        Me.dataGridView1.Columns(0).SortMode = _
            DataGridViewColumnSortMode.NotSortable
        Me.dataGridView1.Columns(1).Name = "Category"
        Me.dataGridView1.Columns(2).Name = "Main Ingredients"
        Me.dataGridView1.Columns(3).Name = "Rating"

        ' Hide the column that contains the content that spans
        ' multiple columns.
        Me.dataGridView1.Columns(2).Visible = False

        ' Populate the rows of the DataGridView.
        Dim row1() As String = {"Meatloaf", "Main Dish", _
            "1 lb. lean ground beef, 1/2 cup bread crumbs, " + _
            "1/4 cup ketchup, 1/3 tsp onion powder, 1 clove of garlic, " + _
            "1/2 pack onion soup mix, dash of your favorite BBQ Sauce", "****"}
        Dim row2() As String = {"Key Lime Pie", "Dessert", _
            "lime juice, whipped cream, eggs, evaporated milk", "****"}
        Dim row3() As String = {"Orange-Salsa Pork Chops", "Main Dish", _
            "pork chops, salsa, orange juice, pineapple", "****"}
        Dim row4() As String = {"Black Bean and Rice Salad", "Salad", _
            "black beans, brown rice", "****"}
    End Sub

```

```

Dim row5() As String = {"Chocolate Cheesecake", "Dessert", _
    "cream cheese, unsweetened chocolate", "***"}
Dim row6() As String = {"Black Bean Dip", "Appetizer", _
    "black beans, sour cream, salsa, chips", "***"}
Dim rows() As Object = {row1, row2, row3, row4, row5, row6}
Dim rowArray As String()
For Each rowArray In rows
    Me.dataGridView1.Rows.Add(rowArray)
Next rowArray

' Adjust the row heights to accommodate the normal cell content.
Me.dataGridView1.AutoResizeRows( _
    DataGridViewAutoSizeRowsMode.AllCellsExceptHeaders)
End Sub 'DataGridViewRowPainting_Load

' Forces the control to repaint itself when the user
' manually changes the width of a column.
Sub dataGridView1_ColumnWidthChanged(ByVal sender As Object, _
    ByVal e As DataGridViewColumnEventArgs) _
    Handles dataGridView1.ColumnWidthChanged

    Me.dataGridView1.Invalidate()

End Sub 'dataGridView1_ColumnWidthChanged

' Forces the row to repaint itself when the user changes the
' current cell. This is necessary to refresh the focus rectangle.
Sub dataGridView1_CurrentCellChanged(ByVal sender As Object, _
    ByVal e As EventArgs) Handles dataGridView1.CurrentCellChanged

    If oldRowIndex <> -1 Then
        Me.dataGridView1.InvalidateRow(oldRowIndex)
    End If
    oldRowIndex = Me.dataGridView1.CurrentCellAddress.Y

End Sub 'dataGridView1_CurrentCellChanged

' Paints the custom selection background for selected rows.
Sub dataGridView1_RowPrePaint(ByVal sender As Object, _
    ByVal e As DataGridViewRowPrePaintEventArgs) _
    Handles dataGridView1.RowPrePaint

    ' Do not automatically paint the focus rectangle.
    e.PaintParts = e.PaintParts And Not DataGridViewPaintParts.Focus

    ' Determine whether the cell should be painted with the
    ' custom selection background.
    If (e.State And DataGridViewElementStates.Selected) = _
        DataGridViewElementStates.Selected Then

        ' Calculate the bounds of the row.
        Dim rowBounds As New Rectangle( _
            Me.dataGridView1.RowHeadersWidth, e.RowBounds.Top, _
            Me.dataGridView1.Columns.GetColumnsWidth( _
                DataGridViewElementStates.Visible) - _
            Me.dataGridView1.HorizontalScrollingOffset + 1, _
            e.RowBounds.Height)

        ' Paint the custom selection background.
        Dim backbrush As New _
            System.Drawing.Drawing2D.LinearGradientBrush(rowBounds, _
            Me.dataGridView1.DefaultCellStyle.SelectionBackColor, _
            e.InheritedRowStyle.ForeColor, _
            System.Drawing.Drawing2D.LinearGradientMode.Horizontal)
        Try
            e.Graphics.FillRectangle(backbrush, rowBounds)
        Finally
            backbrush.Dispose()
        End Try
    End If
End Sub 'dataGridView1_RowPrePaint

```

```

    End If

End Sub 'dataGridView1_RowPrePaint

' Paints the content that spans multiple columns and the focus rectangle.
Sub dataGridView1_RowPostPaint(ByVal sender As Object, _
    ByVal e As DataGridViewRowPostPaintEventArgs) _
Handles dataGridView1.RowPostPaint

    ' Calculate the bounds of the row.
    Dim rowBounds As New Rectangle(Me.dataGridView1.RowHeadersWidth, _
        e.RowBounds.Top, Me.dataGridView1.Columns.GetColumnsWidth( _
        DataGridViewElementStates.Visible) - _
        Me.dataGridView1.HorizontalScrollingOffset + 1, e.RowBounds.Height)

    Dim forebrush As SolidBrush = Nothing
    Try
        ' Determine the foreground color.
        If (e.State And DataGridViewElementStates.Selected) = _
            DataGridViewElementStates.Selected Then

            forebrush = New SolidBrush(e.InheritedRowStyle.SelectionForeColor)
        Else
            forebrush = New SolidBrush(e.InheritedRowStyle.ForeColor)
        End If

        ' Get the content that spans multiple columns.
        Dim recipe As Object = _
            Me.dataGridView1.Rows.SharedRow(e.RowIndex).Cells(2).Value

        If (recipe IsNot Nothing) Then
            Dim text As String = recipe.ToString()

            ' Calculate the bounds for the content that spans multiple
            ' columns, adjusting for the horizontal scrolling position
            ' and the current row height, and displaying only whole
            ' lines of text.
            Dim textArea As Rectangle = rowBounds
            textArea.X -= Me.dataGridView1.HorizontalScrollingOffset
            textArea.Width += Me.dataGridView1.HorizontalScrollingOffset
            textArea.Y += rowBounds.Height - e.InheritedRowStyle.Padding.Bottom
            textArea.Height -= rowBounds.Height - e.InheritedRowStyle.Padding.Bottom
            textArea.Height = (textArea.Height \ e.InheritedRowStyle.Font.Height) * _
                e.InheritedRowStyle.Font.Height

            ' Calculate the portion of the text area that needs painting.
            Dim clip As RectangleF = textArea
            clip.Width -= Me.dataGridView1.RowHeadersWidth + 1 - clip.X
            clip.X = Me.dataGridView1.RowHeadersWidth + 1
            Dim oldClip As RectangleF = e.Graphics.ClipBounds
            e.Graphics.SetClip(clip)

            ' Draw the content that spans multiple columns.
            e.Graphics.DrawString(text, e.InheritedRowStyle.Font, forebrush, _
                textArea)

            e.Graphics.SetClip(oldClip)
        End If
    Finally
        forebrush.Dispose()
    End Try

    If Me.dataGridView1.CurrentCellAddress.Y = e.RowIndex Then
        ' Paint the focus rectangle.
        e.DrawFocus(rowBounds, True)
    End If

End Sub 'dataGridView1_RowPostPaint

```

```

' Adjusts the padding when the user changes the row height so that
' the normal cell content is fully displayed and any extra
' height is used for the content that spans multiple columns.
Sub dataGridView1_RowHeightChanged(ByVal sender As Object, _
    ByVal e As DataGridViewRowEventArgs) _
Handles dataGridView1.RowHeightChanged

    ' Calculate the new height of the normal cell content.
    Dim preferredNormalContentHeight As Int32 = _
        e.Row.GetPreferredHeight(e.Row.Index, _
        DataGridViewAutoSizeRowMode.AllCellsExceptHeader, True) - _
        e.Row.DefaultCellStyle.Padding.Bottom()

    ' Specify a new padding.
    Dim newPadding As Padding = e.Row.DefaultCellStyle.Padding
    newPadding.Bottom = e.Row.Height - preferredNormalContentHeight
    e.Row.DefaultCellStyle.Padding = newPadding

End Sub

End Class 'DataGridViewRowPainting

```

Compiling the Code

This example requires:

- References to the System, System.Drawing, and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[DataGridView](#)
[DataGridView.RowPrePaint](#)
[DataGridView.RowPostPaint](#)
[Customizing the Windows Forms DataGridView Control](#)
[DataGridView Control Architecture](#)

How to: Customize Cells and Columns in the Windows Forms DataGridView Control by Extending Their Behavior and Appearance

5/4/2018 • 7 min to read • [Edit Online](#)

The [DataGridView](#) control provides a number of ways to customize its appearance and behavior using properties, events, and companion classes. Occasionally, you may have requirements for your cells that go beyond what these features can provide. You can create your own custom [DataGridViewCell](#) class to provide extended functionality.

You create a custom [DataGridViewCell](#) class by deriving from the [DataGridViewCell](#) base class or one of its derived classes. Although you can display any type of cell in any type of column, you will typically also create a custom [DataGridViewColumn](#) class specialized for displaying your cell type. Column classes derive from [DataGridViewColumn](#) or one of its derived types.

In the following code example, you will create a custom cell class called `DataGridViewRolloverCell` that detects when the mouse enters and leaves the cell boundaries. While the mouse is within the cell's bounds, an inset rectangle is drawn. This new type derives from [DataGridViewTextBoxCell](#) and behaves in all other respects as its base class. The companion column class is called `DataGridViewRolloverColumn`.

To use these classes, create a form containing a [DataGridView](#) control, add one or more `DataGridViewRolloverColumn` objects to the [Columns](#) collection, and populate the control with rows containing values.

NOTE

This example will not work correctly if you add empty rows. Empty rows are created, for example, when you add rows to the control by setting the [RowCount](#) property. This is because the rows added in this case are automatically shared, which means that `DataGridViewRolloverCell` objects are not instantiated until you click on individual cells, thereby causing the associated rows to become unshared.

Because this type of cell customization requires unshared rows, it is not appropriate for use with large data sets. For more information about row sharing, see [Best Practices for Scaling the Windows Forms DataGridView Control](#).

NOTE

When you derive from [DataGridViewCell](#) or [DataGridViewColumn](#) and add new properties to the derived class, be sure to override the `Clone` method to copy the new properties during cloning operations. You should also call the base class's `Clone` method so that the properties of the base class are copied to the new cell or column.

To customize cells and columns in the DataGridView control

- Derive a new cell class, called `DataGridViewRolloverCell`, from the [DataGridViewTextBoxCell](#) type.

```
public class DataGridViewRolloverCell : DataGridViewTextBoxCell
{
```

```
Public Class DataGridViewRolloverCell  
    Inherits DataGridViewTextBoxCell
```

```
}
```

```
End Class
```

2. Override the [Paint](#) method in the `DataGridViewRolloverCell` class. In the override, first call the base class implementation, which handles the hosted text box functionality. Then use the control's [PointToClient](#) method to transform the cursor position (in screen coordinates) to the [DataGridView](#) client area's coordinates. If the mouse coordinates fall within the bounds of the cell, draw the inset rectangle.

```
protected override void Paint(  
    Graphics graphics,  
    Rectangle clipBounds,  
    Rectangle cellBounds,  
    int rowIndex,  
    DataGridViewElementStates cellState,  
    object value,  
    object formattedValue,  
    string errorText,  
    DataGridViewCellStyle cellStyle,  
    DataGridViewAdvancedBorderStyle advancedBorderStyle,  
    DataGridViewPaintParts paintParts)  
{  
    // Call the base class method to paint the default cell appearance.  
    base.Paint(graphics, clipBounds, cellBounds, rowIndex, cellState,  
        value, formattedValue, errorText, cellStyle,  
        advancedBorderStyle, paintParts);  
  
    // Retrieve the client location of the mouse pointer.  
    Point cursorPosition =  
        this.DataGridView.PointToClient(Cursor.Position);  
  
    // If the mouse pointer is over the current cell, draw a custom border.  
    if (cellBounds.Contains(cursorPosition))  
    {  
        Rectangle newRect = new Rectangle(cellBounds.X + 1,  
            cellBounds.Y + 1, cellBounds.Width - 4,  
            cellBounds.Height - 4);  
        graphics.DrawRectangle(Pens.Red, newRect);  
    }  
}
```

```

Protected Overrides Sub Paint( _
    ByVal graphics As Graphics, _
    ByVal clipBounds As Rectangle, _
    ByVal cellBounds As Rectangle, _
    ByVal rowIndex As Integer, _
    ByVal elementState As DataGridViewElementStates, _
    ByVal value As Object, _
    ByVal formattedValue As Object, _
    ByVal errorText As String, _
    ByVal cellStyle As DataGridViewCellStyle, _
    ByVal advancedBorderStyle As DataGridViewAdvancedBorderStyle, _
    ByVal paintParts As DataGridViewPaintParts)

    ' Call the base class method to paint the default cell appearance.
    MyBase.Paint(graphics, clipBounds, cellBounds, rowIndex, elementState, _
        value, formattedValue, errorText, cellStyle, _
        advancedBorderStyle, paintParts)

    ' Retrieve the client location of the mouse pointer.
    Dim cursorPosition As Point = _
        Me.DataGridView.PointToClient(Cursor.Position)

    ' If the mouse pointer is over the current cell, draw a custom border.
    If cellBounds.Contains(cursorPosition) Then
        Dim newRect As New Rectangle(cellBounds.X + 1, _
            cellBounds.Y + 1, cellBounds.Width - 4, _
            cellBounds.Height - 4)
        graphics.DrawRectangle(Pens.Red, newRect)
    End If

End Sub

```

3. Override the [OnMouseEnter](#) and [OnMouseLeave](#) methods in the [DataGridViewRolloverCell](#) class to force cells to repaint themselves when the mouse pointer enters or leaves them.

```

// Force the cell to repaint itself when the mouse pointer enters it.
protected override void OnMouseEnter(int rowIndex)
{
    this.DataGridView.InvalidateCell(this);
}

// Force the cell to repaint itself when the mouse pointer leaves it.
protected override void OnMouseLeave(int rowIndex)
{
    this.DataGridView.InvalidateCell(this);
}

```

```

' Force the cell to repaint itself when the mouse pointer enters it.
Protected Overrides Sub OnMouseEnter(ByVal rowIndex As Integer)
    Me.DataGridView.InvalidateCell(Me)
End Sub

' Force the cell to repaint itself when the mouse pointer leaves it.
Protected Overrides Sub OnMouseLeave(ByVal rowIndex As Integer)
    Me.DataGridView.InvalidateCell(Me)
End Sub

```

4. Derive a new class, called [DataGridViewRolloverCellColumn](#), from the [DataGridViewColumn](#) type. In the constructor, assign a new [DataGridViewRolloverCell](#) object to its [CellTemplate](#) property.

```

public class DataGridViewRolloverCellColumn : DataGridViewColumn
{
    public DataGridViewRolloverCellColumn()
    {
        this.CellTemplate = new DataGridViewRolloverCell();
    }
}

```

```

Public Class DataGridViewRolloverCellColumn
    Inherits DataGridViewColumn

    Public Sub New()
        Me.CellTemplate = New DataGridViewRolloverCell()
    End Sub

End Class

```

Example

The complete code example includes a small test form that demonstrates the behavior of the custom cell type.

```

using System;
using System.Drawing;
using System.Windows.Forms;

class Form1 : Form
{
    [STAThreadAttribute()]
    public static void Main()
    {
        Application.Run(new Form1());
    }

    public Form1()
    {
        DataGridView dataGridView1 = new DataGridView();
        DataGridViewRolloverCellColumn col =
            new DataGridViewRolloverCellColumn();
        dataGridView1.Columns.Add(col);
        dataGridView1.Rows.Add(new string[] { "" });
        this.Controls.Add(dataGridView1);
        this.Text = "DataGridView rollover-cell demo";
    }
}

public class DataGridViewRolloverCell : DataGridViewTextBoxCell
{
    protected override void Paint(
        Graphics graphics,
        Rectangle clipBounds,
        Rectangle cellBounds,
        int rowIndex,
        DataGridViewElementStates cellState,
        object value,
        object formattedValue,
        string errorText,
        DataGridViewCellStyle cellStyle,
        DataGridViewAdvancedBorderStyle advancedBorderStyle,
        DataGridViewPaintParts paintParts)
    {

```

```

// Call the base class method to paint the default cell appearance.
base.Paint(graphics, clipBounds, cellBounds, rowIndex, cellState,
    value, formattedValue, errorText, cellStyle,
    advancedBorderStyle, paintParts);

// Retrieve the client location of the mouse pointer.
Point cursorPosition =
    this.DataGridView.PointToClient(Cursor.Position);

// If the mouse pointer is over the current cell, draw a custom border.
if (cellBounds.Contains(cursorPosition))
{
    Rectangle newRect = new Rectangle(cellBounds.X + 1,
        cellBounds.Y + 1, cellBounds.Width - 4,
        cellBounds.Height - 4);
    graphics.DrawRectangle(Pens.Red, newRect);
}

// Force the cell to repaint itself when the mouse pointer enters it.
protected override void OnMouseEnter(int rowIndex)
{
    this.DataGridView.InvalidateCell(this);
}

// Force the cell to repaint itself when the mouse pointer leaves it.
protected override void OnMouseLeave(int rowIndex)
{
    this.DataGridView.InvalidateCell(this);
}

}

public class DataGridViewRolloverCellColumn : DataGridViewColumn
{
    public DataGridViewRolloverCellColumn()
    {
        this.CellTemplate = new DataGridViewRolloverCell();
    }
}

```

```

Imports System
Imports System.Drawing
Imports System.Windows.Forms

Class Form1
    Inherits Form

    <STAThreadAttribute()> _
    Public Shared Sub Main()
        Application.Run(New Form1())
    End Sub

    Public Sub New()
        Dim dataGridView1 As New DataGridView()
        Dim col As New DataGridViewRolloverCellColumn()
        dataGridView1.Columns.Add(col)
        dataGridView1.Rows.Add(New String() {""})
        dataGridView1.Rows.Add(New String() {""})
        dataGridView1.Rows.Add(New String() {""})
        dataGridView1.Rows.Add(New String() {""})
        Me.Controls.Add(dataGridView1)
        Me.Text = "DataGridView rollover-cell demo"
    End Sub

End Class

```

```

Public Class DataGridViewRolloverCell
    Inherits DataGridViewTextBoxCell

    Protected Overrides Sub Paint( _
        ByVal graphics As Graphics, _
        ByVal clipBounds As Rectangle, _
        ByVal cellBounds As Rectangle, _
        ByVal rowIndex As Integer, _
        ByVal elementState As DataGridViewElementStates, _
        ByVal value As Object, _
        ByVal formattedValue As Object, _
        ByVal errorText As String, _
        ByVal cellStyle As DataGridViewCellStyle, _
        ByVal advancedBorderStyle As DataGridViewAdvancedBorderStyle, _
        ByVal paintParts As DataGridViewPaintParts)

        ' Call the base class method to paint the default cell appearance.
        MyBase.Paint(graphics, clipBounds, cellBounds, rowIndex, elementState, _
            value, formattedValue, errorText, cellStyle, _
            advancedBorderStyle, paintParts)

        ' Retrieve the client location of the mouse pointer.
        Dim cursorPosition As Point = _
            Me.DataGridView.PointToClient(Cursor.Position)

        ' If the mouse pointer is over the current cell, draw a custom border.
        If cellBounds.Contains(cursorPosition) Then
            Dim newRect As New Rectangle(cellBounds.X + 1, _
                cellBounds.Y + 1, cellBounds.Width - 4, _
                cellBounds.Height - 4)
            graphics.DrawRectangle(Pens.Red, newRect)
        End If

    End Sub

    ' Force the cell to repaint itself when the mouse pointer enters it.
    Protected Overrides Sub OnMouseEnter(ByVal rowIndex As Integer)
        Me.DataGridView.InvalidateCell(Me)
    End Sub

    ' Force the cell to repaint itself when the mouse pointer leaves it.
    Protected Overrides Sub OnMouseLeave(ByVal rowIndex As Integer)
        Me.DataGridView.InvalidateCell(Me)
    End Sub

End Class

Public Class DataGridViewRolloverCellColumn
    Inherits DataGridViewColumn

    Public Sub New()
        Me.CellTemplate = New DataGridViewRolloverCell()
    End Sub

End Class

```

Compiling the Code

This example requires:

- References to the System, System.Windows.Forms, and System.Drawing assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[DataGridView](#)

[DataGridViewCell](#)

[DataGridViewColumn](#)

[Customizing the Windows Forms DataGridView Control](#)

[DataGridView Control Architecture](#)

[Column Types in the Windows Forms DataGridView Control](#)

[Best Practices for Scaling the Windows Forms DataGridView Control](#)

How to: Disable Buttons in a Button Column in the Windows Forms DataGridView Control

5/4/2018 • 6 min to read • [Edit Online](#)

The [DataGridView](#) control includes the [DataGridViewButtonCell](#) class for displaying cells with a user interface (UI) like a button. However, [DataGridViewButtonCell](#) does not provide a way to disable the appearance of the button displayed by the cell.

The following code example demonstrates how to customize the [DataGridViewButtonCell](#) class to display buttons that can appear disabled. The example defines a new cell type, [DataGridViewDisableButtonCell](#), that derives from [DataGridViewButtonCell](#). This cell type provides a new [Enabled](#) property that can be set to [false](#) to draw a disabled button in the cell. The example also defines a new column type, [DataGridViewDisableButtonColumn](#), that displays [DataGridViewDisableButtonCell](#) objects. To demonstrate this new cell and column type, the current value of each [DataGridViewCheckBoxCell](#) in the parent [DataGridView](#) determines whether the [Enabled](#) property of the [DataGridViewDisableButtonCell](#) in the same row is [true](#) or [false](#).

NOTE

When you derive from [DataGridViewCell](#) or [DataGridViewColumn](#) and add new properties to the derived class, be sure to override the [Clone](#) method to copy the new properties during cloning operations. You should also call the base class's [Clone](#) method so that the properties of the base class are copied to the new cell or column.

Example

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Windows.Forms.VisualStyles;

class Form1 : Form
{
    private DataGridView dataGridView1 = new DataGridView();

    [STAThread]
    public static void Main()
    {
        Application.EnableVisualStyles();
        Application.Run(new Form1());
    }

    public Form1()
    {
        this.AutoSize = true;
        this.Load += new EventHandler(Form1_Load);
    }

    public void Form1_Load(object sender, EventArgs e)
    {
        DataGridViewCheckBoxColumn column0 =
            new DataGridViewCheckBoxColumn();
        DataGridViewDisableButtonColumn column1 =
            new DataGridViewDisableButtonColumn();
        column0.Name = "CheckBoxes";
        column1.Name = "Buttons";
        dataGridView1.Columns.Add(column0);
    }
}
```

```

        dataGridView1.Columns.Add(column1);
        dataGridView1.Columns.Add(column1);
        dataGridView1.RowCount = 8;
        dataGridView1.AutoSize = true;
        dataGridView1.AllowUserToAddRows = false;
        dataGridView1.ColumnHeadersDefaultCellStyle.Alignment =
            DataGridViewContentAlignment.MiddleCenter;

        // Set the text for each button.
        for (int i = 0; i < dataGridView1.RowCount; i++)
        {
            dataGridView1.Rows[i].Cells["Buttons"].Value =
                "Button " + i.ToString();
        }

        dataGridView1.CellValueChanged +=
            new DataGridViewCellEventHandler(dataGridView1_CellValueChanged);
        dataGridView1.CurrentCellDirtyStateChanged +=
            new EventHandler(dataGridView1_CurrentCellDirtyStateChanged);
        dataGridView1.CellClick +=
            new DataGridViewCellEventHandler(dataGridView1_CellClick);

        this.Controls.Add(dataGridView1);
    }

    // This event handler manually raises the CellValueChanged event
    // by calling the CommitEdit method.
    void dataGridView1_CurrentCellDirtyStateChanged(object sender,
        EventArgs e)
    {
        if (dataGridView1.IsCurrentCellDirty)
        {
            dataGridView1.CommitEdit(DataGridViewDataErrorContexts.Commit);
        }
    }

    // If a check box cell is clicked, this event handler disables
    // or enables the button in the same row as the clicked cell.
    public void dataGridView1_CellValueChanged(object sender,
        DataGridViewCellEventArgs e)
    {
        if (dataGridView1.Columns[e.ColumnIndex].Name == "Checkboxes")
        {
            DataGridViewDisableButtonCell buttonCell =
                (DataGridViewDisableButtonCell)dataGridView1.
                    Rows[e.RowIndex].Cells["Buttons"];

            DataGridViewCheckBoxCell checkCell =
                (DataGridViewCheckBoxCell)dataGridView1.
                    Rows[e.RowIndex].Cells["Checkboxes"];
            buttonCell.Enabled = !(Boolean)checkCell.Value;

            dataGridView1.Invalidate();
        }
    }

    // If the user clicks on an enabled button cell, this event handler
    // reports that the button is enabled.
    void dataGridView1_CellClick(object sender,
        DataGridViewCellEventArgs e)
    {
        if (dataGridView1.Columns[e.ColumnIndex].Name == "Buttons")
        {
            DataGridViewDisableButtonCell buttonCell =
                (DataGridViewDisableButtonCell)dataGridView1.
                    Rows[e.RowIndex].Cells["Buttons"];

            if (buttonCell.Enabled)
            {

```

```

        MessageBox.Show(dataGridView1.Rows[e.RowIndex].
            Cells[e.ColumnIndex].Value.ToString() +
            " is enabled");
    }
}
}

public class DataGridViewDisableButtonColumn : DataGridViewButtonColumn
{
    public DataGridViewDisableButtonColumn()
    {
        this.CellTemplate = new DataGridViewDisableButtonCell();
    }
}

public class DataGridViewDisableButtonCell : DataGridViewButtonCell
{
    private bool enabledValue;
    public bool Enabled
    {
        get
        {
            return enabledValue;
        }
        set
        {
            enabledValue = value;
        }
    }

    // Override the Clone method so that the Enabled property is copied.
    public override object Clone()
    {
        DataGridViewDisableButtonCell cell =
            (DataGridViewDisableButtonCell)base.Clone();
        cell.Enabled = this.Enabled;
        return cell;
    }

    // By default, enable the button cell.
    public DataGridViewDisableButtonCell()
    {
        this.enabledValue = true;
    }

    protected override void Paint(Graphics graphics,
        Rectangle clipBounds, Rectangle cellBounds, int rowIndex,
        DataGridViewElementStates elementState, object value,
        object formattedValue, string errorText,
        DataGridViewCellStyle cellStyle,
        DataGridViewAdvancedBorderStyle advancedBorderStyle,
        DataGridViewPaintParts paintParts)
    {
        // The button cell is disabled, so paint the border,
        // background, and disabled button for the cell.
        if (!this.enabledValue)
        {
            // Draw the cell background, if specified.
            if ((paintParts & DataGridViewPaintParts.Background) ==
                DataGridViewPaintParts.Background)
            {
                SolidBrush cellBackground =
                    new SolidBrush(cellStyle.BackColor);
                graphics.FillRectangle(cellBackground, cellBounds);
                cellBackground.Dispose();
            }
        }

        // Draw the cell borders, if specified.
    }
}

```

```

        if ((paintParts & DataGridViewPaintParts.Border) ==
            DataGridViewPaintParts.Border)
    {
        PaintBorder(graphics, clipBounds, cellBounds, cellStyle,
            advancedBorderStyle);
    }

    // Calculate the area in which to draw the button.
    Rectangle buttonArea = cellBounds;
    Rectangle buttonAdjustment =
        this.BorderWidths(advancedBorderStyle);
    buttonArea.X += buttonAdjustment.X;
    buttonArea.Y += buttonAdjustment.Y;
    buttonArea.Height -= buttonAdjustment.Height;
    buttonArea.Width -= buttonAdjustment.Width;

    // Draw the disabled button.
    ButtonRenderer.DrawButton(graphics, buttonArea,
        PushButtonState.Disabled);

    // Draw the disabled button text.
    if (this.FormattedValue is String)
    {
        TextRenderer.DrawText(graphics,
            (string)this.FormattedValue,
            this.DataGridView.Font,
            buttonArea, SystemColors.GrayText);
    }
}
else
{
    // The button cell is enabled, so let the base class
    // handle the painting.
    base.Paint(graphics, clipBounds, cellBounds, rowIndex,
        elementState, value, formattedValue, errorText,
        cellStyle, advancedBorderStyle, paintParts);
}
}
}
}

```

```

Imports System
Imports System.Drawing
Imports System.Windows.Forms
Imports System.Windows.Forms.VisualStyles

Class Form1
    Inherits Form
    Private WithEvents dataGridView1 As New DataGridView()

    <STAThread()> _
    Public Shared Sub Main()
        Application.EnableVisualStyles()
        Application.Run(New Form1())
    End Sub

    Public Sub New()
        Me.AutoSize = True
    End Sub

    Public Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs) _
        Handles Me.Load

        Dim column0 As New DataGridViewCheckBoxColumn()
        Dim column1 As New DataGridViewDisableButtonColumn()
        column0.Name = "Checkboxes"
        column1.Name = "Buttons"
        dataGridView1.Columns.Add(column0)
    End Sub

```

```

    dataGridView1.Columns.Add(column1)

    dataGridView1.RowCount = 8
    dataGridView1.AutoSize = True
    dataGridView1.AllowUserToAddRows = False
    dataGridView1.ColumnHeadersDefaultCellStyle.Alignment = _
        DataGridViewContentAlignment.MiddleCenter

    ' Set the text for each button.
    Dim i As Integer
    For i = 0 To dataGridView1.RowCount - 1
        dataGridView1.Rows(i).Cells("Buttons").Value = _
            "Button " + i.ToString()
    Next i

    Me.Controls.Add(dataGridView1)

End Sub

' This event handler manually raises the CellValueChanged event
' by calling the CommitEdit method.
Sub dataGridView1_CurrentCellDirtyStateChanged( _
    ByVal sender As Object, ByVal e As EventArgs) _
Handles dataGridView1.CurrentCellDirtyStateChanged

    If dataGridView1.IsCurrentCellDirty Then
        dataGridView1.CommitEdit(DataGridViewDataErrorContexts.Commit)
    End If
End Sub

' If a check box cell is clicked, this event handler disables
' or enables the button in the same row as the clicked cell.
Public Sub dataGridView1_CellValueChanged(ByVal sender As Object, _
    ByVal e As DataGridViewCellEventArgs) _
Handles dataGridView1.CellValueChanged

    If dataGridView1.Columns(e.ColumnIndex).Name = "Checkboxes" Then
        Dim buttonCell As DataGridViewDisableButtonCell = _
            CType(dataGridView1.Rows(e.RowIndex).Cells("Buttons"), _
            DataGridViewDisableButtonCell)

        Dim checkCell As DataGridViewCheckBoxCell = _
            CType(dataGridView1.Rows(e.RowIndex).Cells("Checkboxes"), _
            DataGridViewCheckBoxCell)
        buttonCell.Enabled = Not CType(checkCell.Value, [Boolean])

        dataGridView1.Invalidate()
    End If
End Sub

' If the user clicks on an enabled button cell, this event handler
' reports that the button is enabled.
Sub dataGridView1_CellClick(ByVal sender As Object, _
    ByVal e As DataGridViewCellEventArgs) _
Handles dataGridView1.CellClick

    If dataGridView1.Columns(e.ColumnIndex).Name = "Buttons" Then
        Dim buttonCell As DataGridViewDisableButtonCell = _
            CType(dataGridView1.Rows(e.RowIndex).Cells("Buttons"), _
            DataGridViewDisableButtonCell)

        If buttonCell.Enabled Then
            MsgBox(dataGridView1.Rows(e.RowIndex). _
                Cells(e.ColumnIndex).Value.ToString() + _
                " is enabled")
        End If
    End If
End Sub

```

```

End Class

Public Class DataGridViewDisableButtonColumn
    Inherits DataGridViewButtonColumn

    Public Sub New()
        Me.CellTemplate = New DataGridViewDisableButtonCell()
    End Sub
End Class

Public Class DataGridViewDisableButtonCell
    Inherits DataGridViewButtonCell

    Private enabledValue As Boolean
    Public Property Enabled() As Boolean
        Get
            Return enabledValue
        End Get
        Set(ByVal value As Boolean)
            enabledValue = value
        End Set
    End Property

    ' Override the Clone method so that the Enabled property is copied.
    Public Overrides Function Clone() As Object
        Dim Cell As DataGridViewDisableButtonCell = _
            CType(MyBase.Clone(), DataGridViewDisableButtonCell)
        Cell.Enabled = Me.Enabled
        Return Cell
    End Function

    ' By default, enable the button cell.
    Public Sub New()
        Me.enabledValue = True
    End Sub

    Protected Overrides Sub Paint(ByVal graphics As Graphics, _
        ByVal clipBounds As Rectangle, ByVal cellBounds As Rectangle, _
        ByVal rowIndex As Integer, _
        ByVal elementState As DataGridViewElementStates, _
        ByVal value As Object, ByVal formattedValue As Object, _
        ByVal errorText As String, _
        ByVal cellStyle As DataGridViewCellStyle, _
        ByVal advancedBorderStyle As DataGridViewAdvancedBorderStyle, _
        ByVal paintParts As DataGridViewPaintParts)

        ' The button cell is disabled, so paint the border,
        ' background, and disabled button for the cell.
        If Not Me.enabledValue Then

            ' Draw the background of the cell, if specified.
            If (paintParts And DataGridViewPaintParts.Background) = _
                DataGridViewPaintParts.Background Then

                Dim cellBackground As New SolidBrush(cellStyle.BackColor)
                graphics.FillRectangle(cellBackground, cellBounds)
                cellBackground.Dispose()
            End If

            ' Draw the cell borders, if specified.
            If (paintParts And DataGridViewPaintParts.Border) = _
                DataGridViewPaintParts.Border Then

                PaintBorder(graphics, clipBounds, cellBounds, cellStyle, _
                    advancedBorderStyle)
            End If

            ' Calculate the area in which to draw the button.
            Dim buttonArea As Rectangle = cellBounds

```

```

Dim buttonAdjustment As Rectangle = _
    Me.BorderWidths(advancedBorderStyle)
buttonArea.X += buttonAdjustment.X
buttonArea.Y += buttonAdjustment.Y
buttonArea.Height -= buttonAdjustment.Height
buttonArea.Width -= buttonAdjustment.Width

' Draw the disabled button.
ButtonRenderer.DrawButton(graphics, buttonArea, _
    PushButtonState.Disabled)

' Draw the disabled button text.
If TypeOf Me.FormattedValue Is String Then
    TextRenderer.DrawText(graphics, CStr(Me.FormattedValue), _
        Me.DataGridView.Font, buttonArea, SystemColors.GrayText)
End If

Else
    ' The button cell is enabled, so let the base class
    ' handle the painting.
    MyBase.Paint(graphics, clipBounds, cellBounds, rowIndex, _
        elementState, value, formattedValue, errorText, _
        cellStyle, advancedBorderStyle, paintParts)
End If
End Sub

End Class

```

Compiling the Code

This example requires:

- References to the System, System.Drawing, System.Windows.Forms and System.Windows.Forms.VisualStyles assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[Customizing the Windows Forms DataGridView Control](#)

[DataGridView Control Architecture](#)

[Column Types in the Windows Forms DataGridView Control](#)

How to: Host Controls in Windows Forms DataGridView Cells

5/4/2018 • 7 min to read • [Edit Online](#)

The [DataGridView](#) control provides several column types, enabling your users to enter and edit values in a variety of ways. If these column types do not meet your data-entry needs, however, you can create your own column types with cells that host controls of your choosing. To do this, you must define classes that derive from [DataGridViewColumn](#) and [DataGridViewCell](#). You must also define a class that derives from [Control](#) and implements the [IDataGridViewEditingControl](#) interface.

The following code example shows how to create a calendar column. The cells of this column display dates in ordinary text box cells, but when the user edits a cell, a [DateTimePicker](#) control appears. In order to avoid having to implement text box display functionality again, the `CalendarCell` class derives from the [DataGridViewTextBoxCell](#) class rather than inheriting the [DataGridViewCell](#) class directly.

NOTE

When you derive from [DataGridViewCell](#) or [DataGridViewColumn](#) and add new properties to the derived class, be sure to override the `Clone` method to copy the new properties during cloning operations. You should also call the base class's `Clone` method so that the properties of the base class are copied to the new cell or column.

Example

```
using System;
using System.Windows.Forms;

public class CalendarColumn : DataGridViewColumn
{
    public CalendarColumn() : base(new CalendarCell())
    {
    }

    public override DataGridViewCell CellTemplate
    {
        get
        {
            return base.CellTemplate;
        }
        set
        {
            // Ensure that the cell used for the template is a CalendarCell.
            if (value != null &&
                !value.GetType().IsAssignableFrom(typeof(CalendarCell)))
            {
                throw new InvalidCastException("Must be a CalendarCell");
            }
            base.CellTemplate = value;
        }
    }
}

public class CalendarCell : DataGridViewTextBoxCell
{
    public CalendarCell()
    {
    }
}
```

```

        : base()
    {
        // Use the short date format.
        this.Style.Format = "d";
    }

    public override void InitializeEditingControl(int rowIndex, object
        initialFormattedValue, DataGridViewCellStyle dataGridViewCellStyle)
    {
        // Set the value of the editing control to the current cell value.
        base.InitializeEditingControl(rowIndex, initialFormattedValue,
            dataGridViewCellStyle);
        CalendarEditingControl ctl =
            DataGridView.EditingControl as CalendarEditingControl;
        // Use the default row value when Value property is null.
        if (this.Value == null)
        {
            ctl.Value = (DateTime)this.DefaultNewRowValue;
        }
        else
        {
            ctl.Value = (DateTime)this.Value;
        }
    }

    public override Type EditType
    {
        get
        {
            // Return the type of the editing control that CalendarCell uses.
            return typeof(CalendarEditingControl);
        }
    }

    public override Type ValueType
    {
        get
        {
            // Return the type of the value that CalendarCell contains.

            return typeof(DateTime);
        }
    }

    public override object DefaultNewRowValue
    {
        get
        {
            // Use the current date and time as the default value.
            return DateTime.Now;
        }
    }
}

class CalendarEditingControl : DateTimePicker, IDataGridViewEditingControl
{
    DataGridView dataGridView;
    private bool valueChanged = false;
    int rowIndex;

    public CalendarEditingControl()
    {
        this.Format = DateTimePickerFormat.Short;
    }

    // Implements the IDataGridViewEditingControl.EditingControlFormattedValue
    // property.
    public object EditingControlFormattedValue
    {

```

```

        get
        {
            return this.Value.ToShortDateString();
        }
        set
        {
            if (value is String)
            {
                try
                {
                    // This will throw an exception if the string is
                    // null, empty, or not in the format of a date.
                    this.Value = DateTime.Parse((String)value);
                }
                catch
                {
                    // In the case of an exception, just use the
                    // default value so we're not left with a null
                    // value.
                    this.Value = DateTime.Now;
                }
            }
        }
    }

    // Implements the
    // IDataGridViewEditingControl.GetEditingControlFormattedValue method.
    public object GetEditingControlFormattedValue(
        DataGridViewDataErrorContexts context)
    {
        return EditingControlFormattedValue;
    }

    // Implements the
    // IDataGridViewEditingControl.ApplyCellStyleToEditingControl method.
    public void ApplyCellStyleToEditingControl(
        DataGridViewCellStyle dataGridViewCellStyle)
    {
        this.Font = dataGridViewCellStyle.Font;
        this.CalendarForeColor = dataGridViewCellStyle.ForeColor;
        this.CalendarMonthBackground = dataGridViewCellStyle.BackColor;
    }

    // Implements the IDataGridViewEditingControl.EditingControlRowIndex
    // property.
    public int EditingControlRowIndex
    {
        get
        {
            return rowIndex;
        }
        set
        {
            rowIndex = value;
        }
    }

    // Implements the IDataGridViewEditingControl.EditingControlWantsInputKey
    // method.
    public bool EditingControlWantsInputKey(
        Keys key, bool dataGridViewWantsInputKey)
    {
        // Let the DateTimePicker handle the keys listed.
        switch (key & Keys.KeyCode)
        {
            case Keys.Left:
            case Keys.Up:
            case Keys.Down:
            case Keys.Right:

```

```
        case Keys.Home:
        case Keys.End:
        case Keys.PageDown:
        case Keys.PageUp:
            return true;
        default:
            return !dataGridViewWantsInputKey;
    }
}

// Implements the IDataGridViewEditingControl.PrepareEditingControlForEdit
// method.
public void PrepareEditingControlForEdit(bool selectAll)
{
    // No preparation needs to be done.
}

// Implements the IDataGridViewEditingControl
// .RepositionEditingControlOnValueChange property.
public bool RepositionEditingControlOnValueChange
{
    get
    {
        return false;
    }
}

// Implements the IDataGridViewEditingControl
// .EditingControlDataGridView property.
public DataGridView EditingControlDataGridView
{
    get
    {
        return dataGridView;
    }
    set
    {
        dataGridView = value;
    }
}

// Implements the IDataGridViewEditingControl
// .EditingControlValueChanged property.
public bool EditingControlValueChanged
{
    get
    {
        return valueChanged;
    }
    set
    {
        valueChanged = value;
    }
}

// Implements the IDataGridViewEditingControl
// .EditingPanelCursor property.
public Cursor EditingPanelCursor
{
    get
    {
        return base.Cursor;
    }
}

protected override void OnValueChanged(EventArgs eventargs)
{
    // Notify the DataGridView that the contents of the cell
    // have changed.
}
```

```

        // ... more changes ...
        valueChanged = true;
        this.EditingControlDataGridView.NotifyCurrentCellDirty(true);
        base.OnValueChanged(eventargs);
    }
}

public class Form1 : Form
{
    private DataGridView dataGridView1 = new DataGridView();

    [STAThreadAttribute()]
    public static void Main()
    {
        Application.Run(new Form1());
    }

    public Form1()
    {
        this.dataGridView1.Dock = DockStyle.Fill;
        this.Controls.Add(this.dataGridView1);
        this.Load += new EventHandler(Form1_Load);
        this.Text = "DataGridView calendar column demo";
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        CalendarColumn col = new CalendarColumn();
        this.dataGridView1.Columns.Add(col);
        this.dataGridView1.RowCount = 5;
        foreach (DataGridViewRow row in this.dataGridView1.Rows)
        {
            row.Cells[0].Value = DateTime.Now;
        }
    }
}

```

```

Imports System
Imports System.Windows.Forms

Public Class CalendarColumn
    Inherits DataGridViewColumn

    Public Sub New()
        MyBase.New(New CalendarCell())
    End Sub

    Public Overrides Property CellTemplate() As DataGridViewCell
        Get
            Return MyBase.CellTemplate
        End Get
        Set(ByVal value As DataGridViewCell)

            ' Ensure that the cell used for the template is a CalendarCell.
            If (value IsNot Nothing) AndAlso _
                Not value.GetType().IsAssignableFrom(GetType(CalendarCell)) _
            Then
                Throw New InvalidCastException("Must be a CalendarCell")
            End If
            MyBase.CellTemplate = value

        End Set
    End Property

End Class

Public Class CalendarCell
    Inherits DataGridViewTextBoxCell

```

```

Public Sub New()
    ' Use the short date format.
    Me.Style.Format = "d"
End Sub

Public Overrides Sub InitializeEditingControl(ByVal rowIndex As Integer, _
    ByVal initialFormattedValue As Object, _
    ByVal dataGridViewCellStyle As DataGridViewCellStyle)

    ' Set the value of the editing control to the current cell value.
    MyBase.InitializeEditingControl(rowIndex, initialFormattedValue, _
        dataGridViewCellStyle)

    Dim ctl As CalendarEditingControl = _
        CType(DataGridView.EditingControl, CalendarEditingControl)

    ' Use the default row value when Value property is null.
    If (Me.Value Is Nothing) Then
        ctl.Value = CType(Me.DefaultNewRowValue, DateTime)
    Else
        ctl.Value = CType(Me.Value, DateTime)
    End If
End Sub

Public Overrides ReadOnly Property EditType() As Type
    Get
        ' Return the type of the editing control that CalendarCell uses.
        Return GetType(CalendarEditingControl)
    End Get
End Property

Public Overrides ReadOnly Property ValueType() As Type
    Get
        ' Return the type of the value that CalendarCell contains.
        Return GetType(DateTime)
    End Get
End Property

Public Overrides ReadOnly Property DefaultNewRowValue() As Object
    Get
        ' Use the current date and time as the default value.
        Return DateTime.Now
    End Get
End Property

End Class

Class CalendarEditingControl
    Inherits DateTimePicker
    Implements IDataGridViewEditingControl

    Private dataGridViewControl As DataGridView
    Private valueIsChanged As Boolean = False
    Private rowIndexNum As Integer

    Public Sub New()
        Me.Format = DateTimePickerFormat.Short
    End Sub

    Public Property EditingControlFormattedValue() As Object _
        Implements IDataGridViewEditingControl.EditingControlFormattedValue

        Get
            Return Me.Value.ToShortDateString()
        End Get

        Set(ByVal value As Object)
            Trv
        End Set
    End Property

```

```

    ...
        ' This will throw an exception of the string is
        ' null, empty, or not in the format of a date.
        Me.Value = DateTime.Parse(CStr(value))
    Catch
        ' In the case of an exception, just use the default
        ' value so we're not left with a null value.
        Me.Value = DateTime.Now
    End Try
End Set

End Property

Public Function GetEditingControlFormattedValue(ByVal context _
As DataGridViewDataErrorContexts) As Object _
Implements IDataGridViewEditingControl.GetEditingControlFormattedValue

    Return Me.Value.ToShortDateString()

End Function

Public Sub ApplyCellStyleToEditingControl(ByVal dataGridViewCellStyle As _
DataGridViewCellStyle) _
Implements IDataGridViewEditingControl.ApplyCellStyleToEditingControl

    Me.Font = dataGridViewCellStyle.Font
    Me.CalendarForeColor = dataGridViewCellStyle.ForeColor
    Me.CalendarMonthBackground = dataGridViewCellStyle.BackColor

End Sub

Public Property EditingControlRowIndex() As Integer _
Implements IDataGridViewEditingControl.EditingControlRowIndex

    Get
        Return rowIndexNum
    End Get
    Set(ByVal value As Integer)
        rowIndexNum = value
    End Set
End Property

Public Function EditingControlWantsInputKey(ByVal key As Keys, _
ByVal dataGridViewWantsInputKey As Boolean) As Boolean _
Implements IDataGridViewEditingControl.EditingControlWantsInputKey

    ' Let the DateTimePicker handle the keys listed.
    Select Case key And Keys.KeyCode
        Case Keys.Left, Keys.Up, Keys.Down, Keys.Right, _
Keys.Home, Keys.End, Keys.PageDown, Keys.PageUp

            Return True

        Case Else
            Return Not dataGridViewWantsInputKey
    End Select
End Function

Public Sub PrepareEditingControlForEdit(ByVal selectAll As Boolean) _
Implements IDataGridViewEditingControl.PrepareEditingControlForEdit

    ' No preparation needs to be done.

End Sub

Public ReadOnly Property RepositionEditingControlOnValueChange() _
As Boolean Implements _

```

```

    Implements IDataGridViewEditingControl.RepositionEditingControlOnValueChange

    Get
        Return False
    End Get

End Property

Public Property EditingControlDataGridView() As DataGridView _
    Implements IDataGridViewEditingControl.EditingControlDataGridView

    Get
        Return dataGridViewControl
    End Get
    Set(ByVal value As DataGridView)
        dataGridViewControl = value
    End Set

End Property

Public Property EditingControlValueChanged() As Boolean _
    Implements IDataGridViewEditingControl.EditingControlValueChanged

    Get
        Return valueIsChanged
    End Get
    Set(ByVal value As Boolean)
        valueIsChanged = value
    End Set

End Property

Public ReadOnly Property EditingControlCursor() As Cursor _
    Implements IDataGridViewEditingControl.EditingPanelCursor

    Get
        Return MyBase.Cursor
    End Get

End Property

Protected Overrides Sub OnValueChanged(ByVal eventargs As EventArgs)

    ' Notify the DataGridView that the contents of the cell have changed.
    valueIsChanged = True
    Me.EditingControlDataGridView.NotifyCurrentCellDirty(True)
    MyBase.OnValueChanged(eventargs)

End Sub

End Class

Public Class Form1
    Inherits Form

    Private dataGridView1 As New DataGridView()

    <STAThreadAttribute()> _
    Public Shared Sub Main()
        Application.Run(New Form1())
    End Sub

    Public Sub New()
        Me.dataGridView1.Dock = DockStyle.Fill
        Me.Controls.Add(Me.dataGridView1)
        Me.Text = "DataGridView calendar column demo"
    End Sub

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs) _

```

```
Handles Me.Load

    Dim col As New CalendarColumn()
    Me.dataGridView1.Columns.Add(col)
    Me.dataGridView1.RowCount = 5
    Dim row As DataGridViewRow
    For Each row In Me.dataGridView1.Rows
        row.Cells(0).Value = DateTime.Now
    Next row

End Sub

End Class
```

Compiling the Code

The following example requires:

- References to the System and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[DataGridView](#)
[DataGridViewColumn](#)
[DataGridViewCell](#)
[DataGridViewTextBoxCell](#)
[IDataGridViewEditingControl](#)
[DateTimePicker](#)
[Customizing the Windows Forms DataGridView Control](#)
[DataGridView Control Architecture](#)
[Column Types in the Windows Forms DataGridView Control](#)

Performance Tuning in the Windows Forms DataGridView Control

5/4/2018 • 1 min to read • [Edit Online](#)

When working with large amounts of data, the `DataGridView` control can consume a large amount of memory in overhead, unless you use it carefully. On clients with limited memory, you can avoid some of this overhead by avoiding features that have a high memory cost. You can also manage some or all of the data maintenance and retrieval tasks yourself using virtual mode in order to customize the memory usage for your scenario.

In This Section

[Best Practices for Scaling the Windows Forms DataGridView Control](#)

Describes how to use the `DataGridView` control in a way that avoids unnecessary memory usage and performance penalties when working with large amounts of data.

[Virtual Mode in the Windows Forms DataGridView Control](#)

Describes how to use virtual mode to supplement or replace the standard data-binding mechanism.

[Walkthrough: Implementing Virtual Mode in the Windows Forms DataGridView Control](#)

Describes how to implement handlers for several virtual-mode events. Also demonstrates how to implement row-level rollback and commit for user edits.

[Implementing Virtual Mode with Just-In-Time Data Loading in the Windows Forms DataGridView Control](#)

Describes how to load data on demand, which is useful when you have more data to display than the available client memory can store.

Reference

[DataGridView](#)

Provides reference documentation for the `DataGridView` control.

[VirtualMode](#)

Provides reference documentation for the `VirtualMode` property.

See Also

[DataGridView Control](#)

[Data Display Modes in the Windows Forms DataGridView Control](#)

Best Practices for Scaling the Windows Forms DataGridView Control

5/4/2018 • 8 min to read • [Edit Online](#)

The [DataGridView](#) control is designed to provide maximum scalability. If you need to display large amounts of data, you should follow the guidelines described in this topic to avoid consuming large amounts of memory or degrading the responsiveness of the user interface (UI). This topic discusses the following issues:

- Using cell styles efficiently
- Using shortcut menus efficiently
- Using automatic resizing efficiently
- Using the selected cells, rows, and columns collections efficiently
- Using shared rows
- Preventing rows from becoming unshared

If you have special performance needs, you can implement virtual mode and provide your own data management operations. For more information, see [Data Display Modes in the Windows Forms DataGridView Control](#).

Using Cell Styles Efficiently

Each cell, row, and column can have its own style information. Style information is stored in [DataGridViewCellStyle](#) objects. Creating cell style objects for many individual [DataGridView](#) elements can be inefficient, especially when working with large amounts of data. To avoid a performance impact, use the following guidelines:

- Avoid setting cell style properties for individual [DataGridViewCell](#) or [DataGridViewRow](#) objects. This includes the row object specified by the [RowTemplate](#) property. Each new row that is cloned from the row template will receive its own copy of the template's cell style object. For maximum scalability, set cell style properties at the [DataGridView](#) level. For example, set the [DataGridView.DefaultCellStyle](#) property rather than the [DataGridViewCell.Style](#) property.
- If some cells require formatting other than default formatting, use the same [DataGridViewCellStyle](#) instance across groups of cells, rows, or columns. Avoid directly setting properties of type [DataGridViewCellStyle](#) on individual cells, rows, and columns. For an example of cell style sharing, see [How to: Set Default Cell Styles for the Windows Forms DataGridView Control](#). You can also avoid a performance penalty when setting cell styles individually by handling the [CellFormatting](#) event handler. For an example, see [How to: Customize Data Formatting in the Windows Forms DataGridView Control](#).
- When determining a cell's style, use the [DataGridViewCell.InheritedStyle](#) property rather than the [DataGridViewCell.Style](#) property. Accessing the [Style](#) property creates a new instance of the [DataGridViewCellStyle](#) class if the property has not already been used. Additionally, this object might not contain the complete style information for the cell if some styles are inherited from the row, column, or control. For more information about cell style inheritance, see [Cell Styles in the Windows Forms DataGridView Control](#).

Using Shortcut Menus Efficiently

Each cell, row, and column can have its own shortcut menu. Shortcut menus in the [DataGridView](#) control are represented by [ContextMenuStrip](#) controls. Just as with cell style objects, creating shortcut menus for many individual [DataGridView](#) elements will negatively impact performance. To avoid this penalty, use the following guidelines:

- Avoid creating shortcut menus for individual cells and rows. This includes the row template, which is cloned along with its shortcut menu when new rows are added to the control. For maximum scalability, use only the control's [ContextMenuStrip](#) property to specify a single shortcut menu for the entire control.
- If you require multiple shortcut menus for multiple rows or cells, handle the [CellContextMenuStripNeeded](#) or [RowContextMenuStripNeeded](#) events. These events let you manage the shortcut menu objects yourself, allowing you to tune performance.

Using Automatic Resizing Efficiently

Rows, columns, and headers can be automatically resized as cell content changes so that the entire contents of cells are displayed without clipping. Changing sizing modes can also resize rows, columns, and headers. To determine the correct size, the [DataGridView](#) control must examine the value of each cell that it must accommodate. When working with large data sets, this analysis can negatively impact the performance of the control when automatic resizing occurs. To avoid performance penalties, use the following guidelines:

- Avoid using automatic sizing on a [DataGridView](#) control with a large set of rows. If you do use automatic sizing, only resize based on the displayed rows. Use only the displayed rows in virtual mode as well.
 - For rows and columns, use the [DisplayedCells](#) or [DisplayedCellsExceptHeaders](#) field of the [DataGridViewAutoSizeRowsMode](#), [DataGridViewAutoSizeColumnsMode](#), and [DataGridViewAutoSizeColumnMode](#) enumerations.
 - For row headers, use the [AutoSizeToDisplayedHeaders](#) or [AutoSizeToFirstHeader](#) field of the [DataGridViewRowHeadersWidthSizeMode](#) enumeration.
- For maximum scalability, turn off automatic sizing and use programmatic resizing.

For more information, see [Sizing Options in the Windows Forms DataGridView Control](#).

Using the Selected Cells, Rows, and Columns Collections Efficiently

The [SelectedCells](#) collection does not perform efficiently with large selections. The [SelectedRows](#) and [SelectedColumns](#) collections can also be inefficient, although to a lesser degree because there are many fewer rows than cells in a typical [DataGridView](#) control, and many fewer columns than rows. To avoid performance penalties when working with these collections, use the following guidelines:

- To determine whether all the cells in the [DataGridView](#) have been selected before you access the contents of the [SelectedCells](#) collection, check the return value of the [AreAllCellsSelected](#) method. Note, however, that this method can cause rows to become unshared. For more information, see the next section.
- Avoid using the [Count](#) property of the [System.Windows.Forms.DataGridViewSelectedCellCollection](#) to determine the number of selected cells. Instead, use the [DataGridView.GetCellCount](#) method and pass in the [DataGridViewElementStates.Selected](#) value. Similarly, use the [DataGridViewRowCollection.GetRowCount](#) and [DataGridViewColumnCollection.GetColumnCount](#) methods to determine the number of selected elements, rather than accessing the selected row and column collections.
- Avoid cell-based selection modes. Instead, set the [DataGridView.SelectionMode](#) property to [DataGridViewSelectionMode.FullRowSelect](#) or [DataGridViewSelectionMode.FullColumnSelect](#).

Using Shared Rows

Efficient memory use is achieved in the [DataGridView](#) control through shared rows. Rows will share as much information about their appearance and behavior as possible by sharing instances of the [DataGridViewRow](#) class.

While sharing row instances saves memory, rows can easily become unshared. For example, whenever a user interacts directly with a cell, its row becomes unshared. Because this cannot be avoided, the guidelines in this topic are useful only when working with very large amounts of data and only when users will interact with a relatively small part of the data each time your program is run.

A row cannot be shared in an unbound [DataGridView](#) control if any of its cells contain values. When the [DataGridView](#) control is bound to an external data source or when you implement virtual mode and provide your own data source, the cell values are stored outside the control rather than in cell objects, allowing the rows to be shared.

A row object can only be shared if the state of all its cells can be determined from the state of the row and the states of the columns containing the cells. If you change the state of a cell so that it can no longer be deduced from the state of its row and column, the row cannot be shared.

For example, a row cannot be shared in any of the following situations:

- The row contains a single selected cell that is not in a selected column.
- The row contains a cell with its [ToolTipText](#) or [ContextMenuStrip](#) properties set.
- The row contains a [DataGridViewComboBoxCell](#) with its [Items](#) property set.

In bound mode or virtual mode, you can provide ToolTips and shortcut menus for individual cells by handling the [CellToolTipTextNeeded](#) and [CellContextMenuStripNeeded](#) events.

The [DataGridView](#) control will automatically attempt to use shared rows whenever rows are added to the [DataGridViewRowCollection](#). Use the following guidelines to ensure that rows are shared:

- Avoid calling the `Add(Object[])` overload of the [Add](#) method and the `Insert(Object[])` overload of the [Insert](#) method of the [DataGridView.Rows](#) collection. These overloads automatically create unshared rows.
- Be sure that the row specified in the [DataGridView.RowTemplate](#) property can be shared in the following cases:
 - When calling the `Add()` or `Add(Int32)` overloads of the [Add](#) method or the `Insert(Int32, Int32)` overload of the [Insert](#) method of the [DataGridView.Rows](#) collection.
 - When increasing the value of the [DataGridView.RowCount](#) property.
 - When setting the [DataGridView.DataSource](#) property.
- Be sure that the row indicated by the `indexSource` parameter can be shared when calling the [AddCopy](#), [AddCopies](#), [InsertCopy](#), and [InsertCopies](#) methods of the [DataGridView.Rows](#) collection.
- Be sure that the specified row or rows can be shared when calling the `Add(DataGridViewRow)` overload of the [Add](#) method, the [AddRange](#) method, the `Insert(Int32, DataGridViewRow)` overload of the [Insert](#) method, and the [InsertRange](#) method of the [DataGridView.Rows](#) collection.

To determine whether a row is shared, use the [DataGridViewRowCollection.SharedRow](#) method to retrieve the row object, and then check the object's [Index](#) property. Shared rows always have an [Index](#) property value of -1.

Preventing Rows from Becoming Unshared

Shared rows can become unshared as a result of code or user action. To avoid a performance impact, you should avoid causing rows to become unshared. During application development, you can handle the [RowUnshared](#) event to determine when rows become unshared. This is useful when debugging row-sharing problems.

To prevent rows from becoming unshared, use the following guidelines:

- Avoid indexing the [Rows](#) collection or iterating through it with a `foreach` loop. You will not typically need to access rows directly. [DataGridView](#) methods that operate on rows take row index arguments rather than row instances. Additionally, handlers for row-related events receive event argument objects with row properties that you can use to manipulate rows without causing them to become unshared.
- If you need to access a row object, use the [DataGridViewRowCollection.SharedRow](#) method and pass in the row's actual index. Note, however, that modifying a shared row object retrieved through this method will modify all the rows that share this object. The row for new records is not shared with other rows, however, so it will not be affected when you modify any other row. Note also that different rows represented by a shared row may have different shortcut menus. To retrieve the correct shortcut menu from a shared row instance, use the [GetContextMenuStrip](#) method and pass in the row's actual index. If you access the shared row's [ContextMenuStrip](#) property instead, it will use the shared row index of -1 and will not retrieve the correct shortcut menu.
- Avoid indexing the [DataGridViewRow.Cells](#) collection. Accessing a cell directly will cause its parent row to become unshared, instantiating a new [DataGridViewRow](#). Handlers for cell-related events receive event argument objects with cell properties that you can use to manipulate cells without causing rows to become unshared. You can also use the [CurrentCellAddress](#) property to retrieve the row and column indexes of the current cell without accessing the cell directly.
- Avoid cell-based selection modes. These modes cause rows to become unshared. Instead, set the [DataGridView.SelectionMode](#) property to [DataGridViewSelectionMode.FullRowSelect](#) or [DataGridViewSelectionMode.FullColumnSelect](#).
- Do not handle the [DataGridViewRowCollection.CollectionChanged](#) or [DataGridView.RowStyleChanged](#) events. These events cause rows to become unshared. Also, do not call the [DataGridViewRowCollection.OnCollectionChanged](#) or [DataGridView.OnRowStateChanged](#) methods, which raise these events.
- Do not access the [DataGridView.SelectedCells](#) collection when the [DataGridView.SelectionMode](#) property value is [FullColumnSelect](#), [ColumnHeaderSelect](#), [FullRowSelect](#), or [RowHeaderSelect](#). This causes all selected rows to become unshared.
- Do not call the [DataGridView.AreAllCellsSelected](#) method. This method can cause rows to become unshared.
- Do not call the [DataGridView.SelectAll](#) method when the [DataGridView.SelectionMode](#) property value is [CellSelect](#). This causes all rows to become unshared.
- Do not set the [ReadOnly](#) or [Selected](#) property of a cell to `false` when the corresponding property in its column is set to `true`. This causes all rows to become unshared.
- Do not access the [DataGridViewRowCollection.List](#) property. This causes all rows to become unshared.
- Do not call the `Sort(IComparer)` overload of the [Sort](#) method. Sorting with a custom comparer causes all rows to become unshared.

See Also

[DataGridView](#)

[Performance Tuning in the Windows Forms DataGridView Control](#)

[Virtual Mode in the Windows Forms DataGridView Control](#)

[Data Display Modes in the Windows Forms DataGridView Control](#)

[Cell Styles in the Windows Forms DataGridView Control](#)

[How to: Set Default Cell Styles for the Windows Forms DataGridView Control](#)

[Sizing Options in the Windows Forms DataGridView Control](#)

Virtual Mode in the Windows Forms DataGridView Control

5/4/2018 • 4 min to read • [Edit Online](#)

With virtual mode, you can manage the interaction between the [DataGridView](#) control and a custom data cache. To implement virtual mode, set the [VirtualMode](#) property to `true` and handle one or more of the events described in this topic. You will typically handle at least the [CellValueNeeded](#) event, which enables the control look up values in the data cache.

Bound Mode and Virtual Mode

Virtual mode is necessary only when you need to supplement or replace bound mode. In bound mode, you set the [DataSource](#) property and the control automatically loads the data from the specified source and submits user changes back to it. You can control which of the bound columns are displayed, and the data source itself typically handles operations such as sorting.

Supplementing Bound Mode

You can supplement bound mode by displaying unbound columns along with the bound columns. This is sometimes called "mixed mode" and is useful for displaying things like calculated values or user-interface (UI) controls.

Because unbound columns are outside the data source, they are ignored by the data source's sorting operations. Therefore, when you enable sorting in mixed mode, you must manage the unbound data in a local cache and implement virtual mode to let the [DataGridView](#) control interact with it.

For more information about using virtual mode to maintain the values in unbound columns, see the examples in the [DataGridViewCheckBoxColumn.ThreeState](#) property and [System.Windows.Forms.DataGridViewComboBoxColumn](#) class reference topics.

Replacing Bound Mode

If bound mode does not meet your performance needs, you can manage all your data in a custom cache through virtual-mode event handlers. For example, you can use virtual mode to implement a just-in-time data loading mechanism that retrieves only as much data from a networked database as is necessary for optimal performance. This scenario is particularly useful when working with large amounts of data over a slow network connection or with client machines that have a limited amount of RAM or storage space.

For more information about using virtual mode in a just-in-time scenario, see [Implementing Virtual Mode with Just-In-Time Data Loading in the Windows Forms DataGridView Control](#).

Virtual-Mode Events

If your data is read-only, the [CellValueNeeded](#) event may be the only event you will need to handle. Additional virtual-mode events let you enable specific functionality like user edits, row addition and deletion, and row-level transactions.

Some standard [DataGridView](#) events (such as events that occur when users add or delete rows, or when cell values are edited, parsed, validated, or formatted) are useful in virtual mode, as well. You can also handle events that let you maintain values not typically stored in a bound data source, such as cell ToolTip text, cell and row

error text, cell and row shortcut menu data, and row height data.

For more information about implementing virtual mode to manage read/write data with a row-level commit scope, see [Walkthrough: Implementing Virtual Mode in the Windows Forms DataGridView Control](#).

For an example that implements virtual mode with a cell-level commit scope, see the [VirtualMode](#) property reference topic.

The following events occur only when the [VirtualMode](#) property is set to `true`.

EVENT	DESCRIPTION
CellValueNeeded	Used by the control to retrieve a cell value from the data cache for display. This event occurs only for cells in unbound columns.
CellValuePushed	Used by the control to commit user input for a cell to the data cache. This event occurs only for cells in unbound columns. Call the UpdateCellValue method when changing a cached value outside of a CellValuePushed event handler to ensure that the current value is displayed in the control and to apply any automatic sizing modes currently in effect.
NewRowNeeded	Used by the control to indicate the need for a new row in the data cache.
RowDirtyStateNeeded	Used by the control to determine whether a row has any uncommitted changes.
CancelRowEdit	Used by the control to indicate that a row should revert to its cached values.

The following events are useful in virtual mode, but can be used regardless of the [VirtualMode](#) property setting.

EVENTS	DESCRIPTION
UserDeletingRow	Used by the control to indicate when rows are deleted or added, letting you update the data cache accordingly.
UserDeletedRow	
RowsRemoved	
RowsAdded	
CellFormatting	Used by the control to format cell values for display and to parse and validate user input.
CellParsing	
CellValidating	
CellValidated	
RowValidating	
RowValidated	

EVENTS	DESCRIPTION
CellToolTipTextNeeded 	<p>Used by the control to retrieve cell ToolTip text when the DataSource property is set or the VirtualMode property is <code>true</code>.</p> <p>Cell ToolTips are displayed only when the ShowCellToolTips property value is <code>true</code>.</p>
CellErrorTextNeeded RowErrorTextNeeded 	<p>Used by the control to retrieve cell or row error text when the DataSource property is set or the VirtualMode property is <code>true</code>.</p> <p>Call the UpdateCellErrorText method or the UpdateRowErrorText method when you change the cell or row error text to ensure that the current value is displayed in the control.</p> <p>Cell and row error glyphs are displayed when the ShowCellErrors and ShowRowErrors property values are <code>true</code>.</p>
CellContextMenuStripNeeded RowContextMenuStripNeeded 	<p>Used by the control to retrieve a cell or row ContextMenuStrip when the control DataSource property is set or the VirtualMode property is <code>true</code>.</p>
RowHeightInfoNeeded RowHeightInfoPushed 	<p>Used by the control to retrieve or store row height information in the data cache. Call the UpdateRowHeightInfo method when changing the cached row height information outside of a RowHeightInfoPushed event handler to ensure that the current value is used in the display of the control.</p>

Best Practices in Virtual Mode

If you are implementing virtual mode in order to work efficiently with large amounts of data, you will also want to ensure that you are working efficiently with the [DataGridView](#) control itself. For more information about the efficient use of cell styles, automatic sizing, selections, and row sharing, see [Best Practices for Scaling the Windows Forms DataGridView Control](#).

See Also

[DataGridView](#)

[VirtualMode](#)

[Performance Tuning in the Windows Forms DataGridView Control](#)

[Best Practices for Scaling the Windows Forms DataGridView Control](#)

[Walkthrough: Implementing Virtual Mode in the Windows Forms DataGridView Control](#)

[Implementing Virtual Mode with Just-In-Time Data Loading in the Windows Forms DataGridView Control](#)

Walkthrough: Implementing Virtual Mode in the Windows Forms DataGridView Control

5/4/2018 • 17 min to read • [Edit Online](#)

When you want to display very large quantities of tabular data in a [DataGridView](#) control, you can set the [VirtualMode](#) property to `true` and explicitly manage the control's interaction with its data store. This lets you fine-tune the performance of the control in this situation.

The [DataGridView](#) control provides several events that you can handle to interact with a custom data store. This walkthrough guides you through the process of implementing these event handlers. The code example in this topic uses a very simple data source for illustration purposes. In a production setting, you will typically load only the rows you need to display into a cache, and handle [DataGridView](#) events to interact with and update the cache. For more information, see [Implementing Virtual Mode with Just-In-Time Data Loading in the Windows Forms DataGridView Control](#)

To copy the code in this topic as a single listing, see [How to: Implement Virtual Mode in the Windows Forms DataGridView Control](#).

Creating the Form

To implement virtual mode

1. Create a class that derives from [Form](#) and contains a [DataGridView](#) control.

The following code contains some basic initialization. It declares some variables that will be used in later steps, provides a `Main` method, and provides a simple form layout in the class constructor.

```
#using <System.Drawing.dll>
#using <System.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

public ref class Customer
{
private:
    String^ companyNameValue;
    String^ contactNameValue;

public:
    Customer()
    {

        // Leave fields empty.
    }

    Customer( String^ companyName, String^ contactName )
    {
        companyNameValue = companyName;
        contactNameValue = contactName;
    }

    property String^ CompanyName
    {
        String^ get()
    }
}
```

```

    }

    return companyNameValue;
}

void set( String^ value )
{
    companyNameValue = value;
}

}

property String^ ContactName
{
    String^ get()
    {
        return contactNameValue;
    }

    void set( String^ value )
    {
        contactNameValue = value;
    }
}

};

public ref class Form1: public Form
{
private:
    DataGridView^ dataGridView1;

    // Declare an ArrayList to serve as the data store.
    System::Collections::ArrayList^ customers;

    // Declare a Customer object to store data for a row being edited.
    Customer^ customerInEdit;

    // Declare a variable to store the index of a row being edited.
    // A value of -1 indicates that there is no row currently in edit.
    int rowInEdit;

    // Declare a variable to indicate the commit scope.
    // Set this value to false to use cell-level commit scope.
    bool rowScopeCommit;

public:
    static void Main()
    {
        Application::Run( gcnew Form1 );
    }

    Form1()
    {
        dataGridView1 = gcnew DataGridView;
        customers = gcnew System::Collections::ArrayList;
        rowInEdit = -1;
        rowScopeCommit = true;

        // Initialize the form.
        this->dataGridView1->Dock = DockStyle::Fill;
        this->Controls->Add( this->dataGridView1 );
        this->Load += gcnew EventHandler( this, &Form1::Form1_Load );
    }

private:

```

```
using System;
using System.Windows.Forms;

public class Form1 : Form
{
    private DataGridView dataGridView1 = new DataGridView();

    // Declare an ArrayList to serve as the data store.
    private System.Collections.ArrayList customers =
        new System.Collections.ArrayList();

    // Declare a Customer object to store data for a row being edited.
    private Customer customerInEdit;

    // Declare a variable to store the index of a row being edited.
    // A value of -1 indicates that there is no row currently in edit.
    private int rowInEdit = -1;

    // Declare a variable to indicate the commit scope.
    // Set this value to false to use cell-level commit scope.
    private bool rowScopeCommit = true;

    [STAThreadAttribute()]
    public static void Main()
    {
        Application.Run(new Form1());
    }

    public Form1()
    {
        // Initialize the form.
        this.dataGridView1.Dock = DockStyle.Fill;
        this.Controls.Add(this.dataGridView1);
        this.Load += new EventHandler(Form1_Load);
        this.Text = "DataGridView virtual-mode demo (row-level commit scope)";
    }
}
```

```

Imports System
Imports System.Windows.Forms

Public Class Form1
    Inherits Form

    Private WithEvents dataGridView1 As New DataGridView()

    ' Declare an ArrayList to serve as the data store.
    Private customers As New System.Collections.ArrayList()

    ' Declare a Customer object to store data for a row being edited.
    Private customerInEdit As Customer

    ' Declare a variable to store the index of a row being edited.
    ' A value of -1 indicates that there is no row currently in edit.
    Private rowInEdit As Integer = -1

    ' Declare a variable to indicate the commit scope.
    ' Set this value to false to use cell-level commit scope.
    Private rowScopeCommit As Boolean = True

    <STAThreadAttribute()> _
    Public Shared Sub Main()
        Application.Run(New Form1())
    End Sub

    Public Sub New()
        ' Initialize the form.
        Me.dataGridView1.Dock = DockStyle.Fill
        Me.Controls.Add(Me.dataGridView1)
        Me.Text = "DataGridView virtual-mode demo (row-level commit scope)"
    End Sub

```

};

```

int main()
{
    Form1::Main();
}

```

}

End Class

2. Implement a handler for your form's [Load](#) event that initializes the [DataGridView](#) control and populates the data store with sample values.

```

void Form1_Load( Object^ /*sender*/, EventArgs^ /*e*/ )
{
    // Enable virtual mode.
    this->dataGridView1->VirtualMode = true;

    // Connect the virtual-mode events to event handlers.
    this->dataGridView1->CellValueNeeded += gcnew
        DataGridViewCellValueEventHandler( this, &Form1::dataGridView1_CellValueNeeded );
    this->dataGridView1->CellValuePushed += gcnew
        DataGridViewCellValueEventHandler( this, &Form1::dataGridView1_CellValuePushed );
    this->dataGridView1->NewRowNeeded += gcnew
        DataGridViewRowEventHandler( this, &Form1::dataGridView1_NewRowNeeded );
    this->dataGridView1->RowValidated += gcnew
        DataGridViewCellEventHandler( this, &Form1::dataGridView1_RowValidated );
    this->dataGridView1->RowDirtyStateNeeded += gcnew
        QuestionEventHandler( this, &Form1::dataGridView1_RowDirtyStateNeeded );
    this->dataGridView1->CancelRowEdit += gcnew
        QuestionEventHandler( this, &Form1::dataGridView1_CancelRowEdit );
    this->dataGridView1->UserDeletingRow += gcnew
        DataGridViewRowCancelEventHandler( this, &Form1::dataGridView1_UserDeletingRow );

    // Add columns to the DataGridView.
    DataGridViewTextBoxColumn^ companyNameColumn = gcnew DataGridViewTextBoxColumn;
    companyNameColumn->HeaderText = L"Company Name";
    companyNameColumn->Name = L"Company Name";
    DataGridViewTextBoxColumn^ contactNameColumn = gcnew DataGridViewTextBoxColumn;
    contactNameColumn->HeaderText = L>Contact Name";
    contactNameColumn->Name = L>Contact Name";
    this->dataGridView1->Columns->Add( companyNameColumn );
    this->dataGridView1->Columns->Add( contactNameColumn );
    this->dataGridView1->AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode::AllCells;

    // Add some sample entries to the data store.
    this->customers->Add( gcnew Customer( L"Bon app'",L"Laurence Lebihan" ) );
    this->customers->Add( gcnew Customer( L"Bottom-Dollar Markets",L"Elizabeth Lincoln" ) );
    this->customers->Add( gcnew Customer( L"B's Beverages",L"Victoria Ashworth" ) );

    // Set the row count, including the row for new records.
    this->dataGridView1->RowCount = 4;
}

```

```
private void Form1_Load(object sender, EventArgs e)
{
    // Enable virtual mode.
    this.dataGridView1.VirtualMode = true;

    // Connect the virtual-mode events to event handlers.
    this.dataGridView1.CellValueNeeded += new
        DataGridViewCellValueEventHandler(dataGridView1_CellValueNeeded);
    this.dataGridView1.CellValuePushed += new
        DataGridViewCellValueEventHandler(dataGridView1_CellValuePushed);
    this.dataGridView1.NewRowNeeded += new
        DataGridViewRowEventHandler(dataGridView1_NewRowNeeded);
    this.dataGridView1.RowValidated += new
        DataGridViewCellEventHandler(dataGridView1_RowValidated);
    this.dataGridView1.RowDirtyStateChanged += new
        QuestionEventHandler(dataGridView1_RowDirtyStateChanged);
    this.dataGridView1.CancelRowEdit += new
        QuestionEventHandler(dataGridView1_CancelRowEdit);
    this.dataGridView1.UserDeletingRow += new
        DataGridViewRowCancelEventHandler(dataGridView1_UserDeletingRow);

    // Add columns to the DataGridView.
    DataGridViewTextBoxColumn companyNameColumn = new
        DataGridViewTextBoxColumn();
    companyNameColumn.HeaderText = "Company Name";
    companyNameColumn.Name = "Company Name";
    DataGridViewTextBoxColumn contactNameColumn = new
        DataGridViewTextBoxColumn();
    contactNameColumn.HeaderText = "Contact Name";
    contactNameColumn.Name = "Contact Name";
    this.dataGridView1.Columns.Add(companyNameColumn);
    this.dataGridView1.Columns.Add(contactNameColumn);
    this.dataGridView1.AutoSizeColumnsMode =
        DataGridViewAutoSizeColumnsMode.AllCells;

    // Add some sample entries to the data store.
    this.customers.Add(new Customer(
        "Bon app'", "Laurence Lebihan"));
    this.customers.Add(new Customer(
        "Bottom-Dollar Markets", "Elizabeth Lincoln"));
    this.customers.Add(new Customer(
        "B's Beverages", "Victoria Ashworth"));

    // Set the row count, including the row for new records.
    this.dataGridView1.RowCount = 4;
}
```

```

Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs) _
Handles Me.Load

    ' Enable virtual mode.
    Me.dataGridView1.VirtualMode = True

    ' Add columns to the DataGridView.
    Dim companyNameColumn As New DataGridViewTextBoxColumn()
    With companyNameColumn
        .HeaderText = "Company Name"
        .Name = "Company Name"
    End With
    Dim contactNameColumn As New DataGridViewTextBoxColumn()
    With contactNameColumn
        .HeaderText = "Contact Name"
        .Name = "Contact Name"
    End With
    Me.dataGridView1.Columns.Add(companyNameColumn)
    Me.dataGridView1.Columns.Add(contactNameColumn)
    Me.dataGridView1.AutoSizeColumnsMode = _
        DataGridViewAutoSizeColumnsMode.AllCells

    ' Add some sample entries to the data store.
    Me.customers.Add(New Customer("Bon app'", "Laurence Lebihan"))
    Me.customers.Add(New Customer("Bottom-Dollar Markets", _
        "Elizabeth Lincoln"))
    Me.customers.Add(New Customer("B's Beverages", "Victoria Ashworth"))

    ' Set the row count, including the row for new records.
    Me.dataGridView1.RowCount = 4

End Sub

```

3. Implement a handler for the [CellValueNeeded](#) event that retrieves the requested cell value from the data store or the [Customer](#) object currently in edit.

This event occurs whenever the [DataGridView](#) control needs to paint a cell.

```
void dataGridView1_CellValueNeeded( Object^ /*sender*/,
    System::Windows::Forms::DataGridViewCellValueEventArgs^ e )
{
    Customer^ customerTmp = nullptr;

    // Store a reference to the Customer object for the row being painted.
    if ( e->RowIndex == rowInEdit )
    {
        customerTmp = this->customerInEdit;
    }
    else
    {
        customerTmp = dynamic_cast<Customer^>(this->customers[ e->RowIndex ]);
    }

    // Set the cell value to paint using the Customer object retrieved.
    int switchcase = 0;
    if ( (this->dataGridView1->Columns[ e->ColumnIndex ]->Name)->Equals( L"Company Name" ) )
        switchcase = 1;
    else
        if ( (this->dataGridView1->Columns[ e->ColumnIndex ]->Name)->Equals( L>Contact Name" ) )
            switchcase = 2;

    switch ( switchcase )
    {
        case 1:
            e->Value = customerTmp->CompanyName;
            break;

        case 2:
            e->Value = customerTmp->ContactName;
            break;
    }
}
```

```

private void dataGridView1_CellValueNeeded(object sender,
    System.Windows.Forms.DataGridViewCellValueEventArgs e)
{
    // If this is the row for new records, no values are needed.
    if (e.RowIndex == this.dataGridView1.RowCount - 1) return;

    Customer customerTmp = null;

    // Store a reference to the Customer object for the row being painted.
    if (e.RowIndex == rowInEdit)
    {
        customerTmp = this.customerInEdit;
    }
    else
    {
        customerTmp = (Customer)this.customers[e.RowIndex];
    }

    // Set the cell value to paint using the Customer object retrieved.
    switch (this.dataGridView1.Columns[e.ColumnIndex].Name)
    {
        case "Company Name":
            e.Value = customerTmp.CompanyName;
            break;

        case "Contact Name":
            e.Value = customerTmp.ContactName;
            break;
    }
}

```

```

Private Sub dataGridView1_CellValueNeeded(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DataGridViewCellValueEventArgs) _
Handles dataGridView1.CellValueNeeded

    ' If this is the row for new records, no values are needed.
    If e.RowIndex = Me.dataGridView1.RowCount - 1 Then
        Return
    End If

    Dim customerTmp As Customer = Nothing

    ' Store a reference to the Customer object for the row being painted.
    If e.RowIndex = rowInEdit Then
        customerTmp = Me.customerInEdit
    Else
        customerTmp = CType(Me.customers(e.RowIndex), Customer)
    End If

    ' Set the cell value to paint using the Customer object retrieved.
    Select Case Me.dataGridView1.Columns(e.ColumnIndex).Name
        Case "Company Name"
            e.Value = customerTmp.CompanyName

        Case "Contact Name"
            e.Value = customerTmp.ContactName
    End Select

End Sub

```

4. Implement a handler for the **CellValuePushed** event that stores an edited cell value in the **Customer** object representing the edited row. This event occurs whenever the user commits a cell value change.

```

void dataGridView1_CellValuePushed( Object^ /*sender*/,
    System::Windows::Forms::DataGridViewCellValueEventArgs^ e )
{
    Customer^ customerTmp = nullptr;

    // Store a reference to the Customer object for the row being edited.
    if ( e->RowIndex < this->customers->Count )
    {

        // If the user is editing a new row, create a new Customer object.
        if ( this->customerInEdit == nullptr )
        {
            this->customerInEdit = gcnew Customer(
                (dynamic_cast<Customer^>(this->customers[ e->RowIndex ]))->CompanyName,
                (dynamic_cast<Customer^>(this->customers[ e->RowIndex ]))->ContactName );
        }

        customerTmp = this->customerInEdit;
        this->rowInEdit = e->RowIndex;
    }
    else
    {
        customerTmp = this->customerInEdit;
    }

    // Set the appropriate Customer property to the cell value entered.
    int switchcase = 0;
    if ( (this->dataGridView1->Columns[ e->ColumnIndex ]->Name)->Equals( L"Company Name" ) )
        switchcase = 1;
    else
        if ( (this->dataGridView1->Columns[ e->ColumnIndex ]->Name)->Equals( L>Contact Name" ) )
            switchcase = 2;

    switch ( switchcase )
    {
        case 1:
            customerTmp->CompanyName = dynamic_cast<String^>(e->Value);
            break;

        case 2:
            customerTmp->ContactName = dynamic_cast<String^>(e->Value);
            break;
    }
}

```

```
private void dataGridView1_CellValuePushed(object sender,
    System.Windows.Forms.DataGridViewCellValueEventArgs e)
{
    Customer customerTmp = null;

    // Store a reference to the Customer object for the row being edited.
    if (e.RowIndex < this.customers.Count)
    {
        // If the user is editing a new row, create a new Customer object.
        if (this.customerInEdit == null)
        {
            this.customerInEdit = new Customer(
                ((Customer)this.customers[e.RowIndex]).CompanyName,
                ((Customer)this.customers[e.RowIndex]).ContactName);
        }
        customerTmp = this.customerInEdit;
        this.rowInEdit = e.RowIndex;
    }
    else
    {
        customerTmp = this.customerInEdit;
    }

    // Set the appropriate Customer property to the cell value entered.
    String newValue = e.Value as String;
    switch (this.dataGridView1.Columns[e.ColumnIndex].Name)
    {
        case "Company Name":
            customerTmp.CompanyName = newValue;
            break;

        case "Contact Name":
            customerTmp.ContactName = newValue;
            break;
    }
}
```

```

Private Sub dataGridView1_CellValuePushed(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DataGridViewCellValueEventArgs) _
Handles dataGridView1.CellValuePushed

    Dim customerTmp As Customer = Nothing

    ' Store a reference to the Customer object for the row being edited.
    If e.RowIndex < Me.customers.Count Then

        ' If the user is editing a new row, create a new Customer object.
        If Me.customerInEdit Is Nothing Then
            Me.customerInEdit = New Customer( _
                CType(Me.customers(e.RowIndex), Customer).CompanyName, _
                CType(Me.customers(e.RowIndex), Customer).ContactName)
        End If
        customerTmp = Me.customerInEdit
        Me.rowInEdit = e.RowIndex

    Else
        customerTmp = Me.customerInEdit
    End If

    ' Set the appropriate Customer property to the cell value entered.
    Dim newValue As String = TryCast(e.Value, String)
    Select Case Me.dataGridView1.Columns(e.ColumnIndex).Name
        Case "Company Name"
            customerTmp.CompanyName = newValue
        Case "Contact Name"
            customerTmp.ContactName = newValue
    End Select

End Sub

```

5. Implement a handler for the [NewRowNeeded](#) event that creates a new `Customer` object representing a newly created row.

This event occurs whenever the user enters the row for new records.

```

void dataGridView1_NewRowNeeded( Object^ /*sender*/, 
    System::Windows::Forms::DataGridViewRowEventArgs^ /*e*/ )
{
    // Create a new Customer object when the user edits
    // the row for new records.
    this->customerInEdit = gcnew Customer;
    this->rowInEdit = this->dataGridView1->Rows->Count - 1;
}

```

```

private void dataGridView1_NewRowNeeded(object sender,
    System.Windows.Forms.DataGridViewRowEventArgs e)
{
    // Create a new Customer object when the user edits
    // the row for new records.
    this.customerInEdit = new Customer();
    this.rowInEdit = this.dataGridView1.Rows.Count - 1;
}

```

```

Private Sub dataGridView1_NewRowNeeded(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DataGridViewRowEventArgs) _
Handles dataGridView1.NewRowNeeded

    ' Create a new Customer object when the user edits
    ' the row for new records.
    Me.customerInEdit = New Customer()
    Me.rowInEdit = Me.dataGridView1.Rows.Count - 1

End Sub

```

6. Implement a handler for the [RowValidated](#) event that saves new or modified rows to the data store.

This event occurs whenever the user changes the current row.

```

void dataGridView1_RowValidated( Object^ /*sender*/,
    System::Windows::Forms::DataGridViewCellEventArgs^ e )
{

    // Save row changes if any were made and release the edited
    // Customer object if there is one.
    if ( e->RowIndex >= this->customers->Count && e->RowIndex != this->dataGridView1->Rows->Count - 1
)
{
    {

        // Add the new Customer object to the data store.
        this->customers->Add( this->customerInEdit );
        this->customerInEdit = nullptr;
        this->rowInEdit = -1;
    }
    else
    if ( this->customerInEdit != nullptr && e->RowIndex < this->customers->Count )
    {

        // Save the modified Customer object in the data store.
        this->customers[ e->RowIndex ] = this->customerInEdit;
        this->customerInEdit = nullptr;
        this->rowInEdit = -1;
    }
    else
    if ( this->dataGridView1->ContainsFocus )
    {
        this->customerInEdit = nullptr;
        this->rowInEdit = -1;
    }
}

```

```

private void dataGridView1_RowValidated(object sender,
    System.Windows.Forms.DataGridViewCellEventArgs e)
{
    // Save row changes if any were made and release the edited
    // Customer object if there is one.
    if (e.RowIndex >= this.customers.Count &&
        e.RowIndex != this.dataGridView1.Rows.Count - 1)
    {
        // Add the new Customer object to the data store.
        this.customers.Add(this.customerInEdit);
        this.customerInEdit = null;
        this.rowInEdit = -1;
    }
    else if (this.customerInEdit != null &&
        e.RowIndex < this.customers.Count)
    {
        // Save the modified Customer object in the data store.
        this.customers[e.RowIndex] = this.customerInEdit;
        this.customerInEdit = null;
        this.rowInEdit = -1;
    }
    else if (this.dataGridView1.ContainsFocus)
    {
        this.customerInEdit = null;
        this.rowInEdit = -1;
    }
}

```

```

Private Sub dataGridView1_RowValidated(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DataGridViewCellEventArgs) _
Handles dataGridView1.RowValidated

    ' Save row changes if any were made and release the edited
    ' Customer object if there is one.
    If e.RowIndex >= Me.customers.Count AndAlso _
        e.RowIndex <> Me.dataGridView1.Rows.Count - 1 Then

        ' Add the new Customer object to the data store.
        Me.customers.Add(Me.customerInEdit)
        Me.customerInEdit = Nothing
        Me.rowInEdit = -1

    ElseIf (Me.customerInEdit IsNot Nothing) AndAlso _
        e.RowIndex < Me.customers.Count Then

        ' Save the modified Customer object in the data store.
        Me.customers(e.RowIndex) = Me.customerInEdit
        Me.customerInEdit = Nothing
        Me.rowInEdit = -1

    ElseIf Me.dataGridView1.ContainsFocus Then

        Me.customerInEdit = Nothing
        Me.rowInEdit = -1

    End If

End Sub

```

7. Implement a handler for the [RowDirtyStateChanged](#) event that indicates whether the [CancelRowEdit](#) event will occur when the user signals row reversion by pressing ESC twice in edit mode or once outside of edit mode.

By default, [CancelRowEdit](#) occurs upon row reversion when any cells in the current row have been

modified unless the `QuestionEventArgs.Response` property is set to `true` in the `RowDirtyStateNeeded` event handler. This event is useful when the commit scope is determined at run time.

```
void dataGridView1_RowDirtyStateNeeded( Object^ /*sender*/,
                                         System::Windows::Forms::QuestionEventArgs^ e )
{
    if ( !rowScopeCommit )
    {
        // In cell-level commit scope, indicate whether the value
        // of the current cell has been modified.
        e->Response = this->dataGridView1->IsCurrentCellDirty;
    }
}
```

```
private void dataGridView1_RowDirtyStateNeeded(object sender,
                                              System.Windows.Forms.QuestionEventArgs e)
{
    if (!rowScopeCommit)
    {
        // In cell-level commit scope, indicate whether the value
        // of the current cell has been modified.
        e.Response = this.dataGridView1.IsCurrentCellDirty;
    }
}
```

```
Private Sub dataGridView1_RowDirtyStateNeeded(ByVal sender As Object, _
                                              ByVal e As System.Windows.Forms.QuestionEventArgs) _
Handles dataGridView1.RowDirtyStateNeeded

    If Not rowScopeCommit Then

        ' In cell-level commit scope, indicate whether the value
        ' of the current cell has been modified.
        e.Response = Me.dataGridView1.IsCurrentCellDirty

    End If

End Sub
```

8. Implement a handler for the `CancelRowEdit` event that discards the values of the `Customer` object representing the current row.

This event occurs when the user signals row reversion by pressing ESC twice in edit mode or once outside of edit mode. This event does not occur if no cells in the current row have been modified or if the value of the `QuestionEventArgs.Response` property has been set to `false` in a `RowDirtyStateNeeded` event handler.

```

void dataGridView1_CancelRowEdit( Object^ /*sender*/,
    System::Windows::Forms::QuestionEventArgs^ /*e*/ )
{
    if ( this->rowInEdit == this->dataGridView1->Rows->Count - 2 &&
        this->rowInEdit == this->customers->Count )
    {

        // If the user has canceled the edit of a newly created row,
        // replace the corresponding Customer object with a new, empty one.
        this->customerInEdit = gcnew Customer;
    }
    else
    {

        // If the user has canceled the edit of an existing row,
        // release the corresponding Customer object.
        this->customerInEdit = nullptr;
        this->rowInEdit = -1;
    }
}

```

```

private void dataGridView1_CancelRowEdit(object sender,
    System.Windows.Forms.QuestionEventArgs e)
{
    if (this.rowInEdit == this.dataGridView1.Rows.Count - 2 &&
        this.rowInEdit == this.customers.Count)
    {
        // If the user has canceled the edit of a newly created row,
        // replace the corresponding Customer object with a new, empty one.
        this.customerInEdit = new Customer();
    }
    else
    {
        // If the user has canceled the edit of an existing row,
        // release the corresponding Customer object.
        this.customerInEdit = null;
        this.rowInEdit = -1;
    }
}

```

```

Private Sub dataGridView1_CancelRowEdit(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.QuestionEventArgs) _
    Handles dataGridView1.CancelRowEdit

    If Me.rowInEdit = Me.dataGridView1.Rows.Count - 2 AndAlso _
        Me.rowInEdit = Me.customers.Count Then

        ' If the user has canceled the edit of a newly created row,
        ' replace the corresponding Customer object with a new, empty one.
        Me.customerInEdit = New Customer()

    Else

        ' If the user has canceled the edit of an existing row,
        ' release the corresponding Customer object.
        Me.customerInEdit = Nothing
        Me.rowInEdit = -1

    End If

End Sub

```

9. Implement a handler for the `UserDeletingRow` event that deletes an existing `Customer` object from the data store or discards an unsaved `Customer` object representing a newly created row.

This event occurs whenever the user deletes a row by clicking a row header and pressing the DELETE key.

```
void dataGridView1_UserDeletingRow( Object^ /*sender*/,
    System::Windows::Forms::DataGridViewRowEventArgs^ e )
{
    if ( e->Row->Index < this->customers->Count )
    {

        // If the user has deleted an existing row, remove the
        // corresponding Customer object from the data store.
        this->customers->RemoveAt( e->Row->Index );
    }

    if ( e->Row->Index == this->rowInEdit )
    {

        // If the user has deleted a newly created row, release
        // the corresponding Customer object.
        this->rowInEdit = -1;
        this->customerInEdit = nullptr;
    }
}
```

```
private void dataGridView1_UserDeletingRow(object sender,
    System.Windows.Forms.DataGridViewRowEventArgs e)
{
    if (e.Row.Index < this.customers.Count)
    {
        // If the user has deleted an existing row, remove the
        // corresponding Customer object from the data store.
        this.customers.RemoveAt(e.Row.Index);
    }

    if (e.Row.Index == this.rowInEdit)
    {
        // If the user has deleted a newly created row, release
        // the corresponding Customer object.
        this.rowInEdit = -1;
        this.customerInEdit = null;
    }
}
```

```
Private Sub dataGridView1_UserDeletingRow(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DataGridViewRowCancelEventArgs) _
Handles dataGridView1.UserDeletingRow

    If e.Row.Index < Me.customers.Count Then

        ' If the user has deleted an existing row, remove the
        ' corresponding Customer object from the data store.
        Me.customers.RemoveAt(e.Row.Index)

    End If

    If e.Row.Index = Me.rowInEdit Then

        ' If the user has deleted a newly created row, release
        ' the corresponding Customer object.
        Me.rowInEdit = -1
        Me.customerInEdit = Nothing

    End If

End Sub
```

10. Implement a simple `Customers` class to represent the data items used by this code example.

```
public ref class Customer
{
private:
    String^ companyNameValue;
    String^ contactNameValue;

public:
    Customer()
    {

        // Leave fields empty.
    }

    Customer( String^ companyName, String^ contactName )
    {
        companyNameValue = companyName;
        contactNameValue = contactName;
    }

    property String^ CompanyName
    {
        String^ get()
        {
            return companyNameValue;
        }

        void set( String^ value )
        {
            companyNameValue = value;
        }
    }

    property String^ ContactName
    {
        String^ get()
        {
            return contactNameValue;
        }

        void set( String^ value )
        {
            contactNameValue = value;
        }
    }
};

};
```

```
public class Customer
{
    private String companyNameValue;
    private String contactNameValue;

    public Customer()
    {
        // Leave fields empty.
    }

    public Customer(String companyName, String contactName)
    {
        companyNameValue = companyName;
        contactNameValue = contactName;
    }

    public String CompanyName
    {
        get
        {
            return companyNameValue;
        }
        set
        {
            companyNameValue = value;
        }
    }

    public String ContactName
    {
        get
        {
            return contactNameValue;
        }
        set
        {
            contactNameValue = value;
        }
    }
}
```

```

Public Class Customer

    Private companyNameValue As String
    Private contactNameValue As String

    Public Sub New()
        ' Leave fields empty.
    End Sub

    Public Sub New(ByVal companyName As String, ByVal contactName As String)
        companyNameValue = companyName
        contactNameValue = contactName
    End Sub

    Public Property CompanyName() As String
        Get
            Return companyNameValue
        End Get
        Set(ByVal value As String)
            companyNameValue = value
        End Set
    End Property

    Public Property ContactName() As String
        Get
            Return contactNameValue
        End Get
        Set(ByVal value As String)
            contactNameValue = value
        End Set
    End Property

End Class

```

Testing the Application

You can now test the form to make sure it behaves as expected.

To test the form

- Compile and run the application.

You will see a [DataGridView](#) control populated with three customer records. You can modify the values of multiple cells in a row and press ESC twice in edit mode and once outside of edit mode to revert the entire row to its original values. When you modify, add, or delete rows in the control, `Customer` objects in the data store are modified, added, or deleted as well.

Next Steps

This application gives you a basic understanding of the events you must handle to implement virtual mode in the [DataGridView](#) control. You can improve this basic application in a number of ways:

- Implement a data store that caches values from an external database. The cache should retrieve and discard values as necessary so that it only contains what is necessary for display while consuming a small amount of memory on the client computer.
- Fine-tune the performance of the data store depending on your requirements. For example, you might want to compensate for slow network connections rather than client-computer memory limitations by using a larger cache size and minimizing the number of database queries.

For more information about caching values from an external database, see [How to: Implement Virtual Mode with Just-In-Time Data Loading in the Windows Forms DataGridView Control](#).

See Also

[DataGridView](#)

[VirtualMode](#)

[CellValueNeeded](#)

[CellValuePushed](#)

[NewRowNeeded](#)

[RowValidated](#)

[RowDirtyStateNeeded](#)

[CancelRowEdit](#)

[UserDeletingRow](#)

[Performance Tuning in the Windows Forms DataGridView Control](#)

[Best Practices for Scaling the Windows Forms DataGridView Control](#)

[Implementing Virtual Mode with Just-In-Time Data Loading in the Windows Forms DataGridView Control](#)

[How to: Implement Virtual Mode in the Windows Forms DataGridView Control](#)

How to: Implement Virtual Mode in the Windows Forms DataGridView Control

5/4/2018 • 13 min to read • [Edit Online](#)

The following code example demonstrates how to manage large sets of data using a [DataGridView](#) control with its [VirtualMode](#) property set to `true`.

For a complete explanation of this code example, see [Walkthrough: Implementing Virtual Mode in the Windows Forms DataGridView Control](#).

Example

```
#using <System.Drawing.dll>
#using <System.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

public ref class Customer
{
private:
    String^ companyNameValue;
    String^ contactNameValue;

public:
    Customer()
    {

        // Leave fields empty.
    }

    Customer( String^ companyName, String^ contactName )
    {
        companyNameValue = companyName;
        contactNameValue = contactName;
    }

    property String^ CompanyName
    {
        String^ get()
        {
            return companyNameValue;
        }

        void set( String^ value )
        {
            companyNameValue = value;
        }
    }

    property String^ ContactName
    {
        String^ get()
        {
            return contactNameValue;
        }
    }
}
```

```

        void set( String^ value )
    {
        contactNameValue = value;
    }

}

};

public ref class Form1: public Form
{
private:
    DataGridView^ dataGridView1;

    // Declare an ArrayList to serve as the data store.
    System::Collections::ArrayList^ customers;

    // Declare a Customer object to store data for a row being edited.
    Customer^ customerInEdit;

    // Declare a variable to store the index of a row being edited.
    // A value of -1 indicates that there is no row currently in edit.
    int rowInEdit;

    // Declare a variable to indicate the commit scope.
    // Set this value to false to use cell-level commit scope.
    bool rowScopeCommit;

public:
    static void Main()
    {
        Application::Run( gcnew Form1 );
    }

    Form1()
    {
        dataGridView1 = gcnew DataGridView;
        customers = gcnew System::Collections::ArrayList;
        rowInEdit = -1;
        rowScopeCommit = true;

        // Initialize the form.
        this->dataGridView1->Dock = DockStyle::Fill;
        this->Controls->Add( this->dataGridView1 );
        this->Load += gcnew EventHandler( this, &Form1::Form1_Load );
    }

private:
    void Form1_Load( Object^ /*sender*/, EventArgs^ /*e*/ )
    {

        // Enable virtual mode.
        this->dataGridView1->VirtualMode = true;

        // Connect the virtual-mode events to event handlers.
        this->dataGridView1->CellValueNeeded += gcnew
            DataGridViewCellValueEventHandler( this, &Form1::dataGridView1_CellValueNeeded );
        this->dataGridView1->CellValuePushed += gcnew
            DataGridViewCellValueEventHandler( this, &Form1::dataGridView1_CellValuePushed );
        this->dataGridView1->NewRowNeeded += gcnew
            DataGridViewRowEventHandler( this, &Form1::dataGridView1_NewRowNeeded );
        this->dataGridView1->RowValidated += gcnew
            DataGridViewCellEventHandler( this, &Form1::dataGridView1_RowValidated );
        this->dataGridView1->RowDirtyStateNeeded += gcnew
            QuestionEventHandler( this, &Form1::dataGridView1_RowDirtyStateNeeded );
        this->dataGridView1->CancelRowEdit += gcnew
            QuestionEventHandler( this, &Form1::dataGridView1_CancelRowEdit );
        this->dataGridView1->UserDeletingRow += gcnew
    }
}

```

```

    DataGridViewRowCancelEventHandler( this, &Form1::dataGridView1_UserDeletingRow );

    // Add columns to the DataGridView.
    DataGridViewTextBoxColumn^ companyNameColumn = gcnew DataGridViewTextBoxColumn;
    companyNameColumn->HeaderText = L"Company Name";
    companyNameColumn->Name = L"Company Name";
    DataGridViewTextBoxColumn^ contactNameColumn = gcnew DataGridViewTextBoxColumn;
    contactNameColumn->HeaderText = L>Contact Name";
    contactNameColumn->Name = L>Contact Name";
    this->dataGridView1->Columns->Add( companyNameColumn );
    this->dataGridView1->Columns->Add( contactNameColumn );
    this->dataGridView1->AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode::AllCells;

    // Add some sample entries to the data store.
    this->customers->Add( gcnew Customer( L"Bon app'",L"Laurence Lebihan" ) );
    this->customers->Add( gcnew Customer( L"Bottom-Dollar Markets",L"Elizabeth Lincoln" ) );
    this->customers->Add( gcnew Customer( L"B's Beverages",L"Victoria Ashworth" ) );

    // Set the row count, including the row for new records.
    this->dataGridView1->RowCount = 4;
}

void dataGridView1_CellValueNeeded( Object^ /*sender*/,
    System::Windows::Forms::DataGridViewCellValueEventArgs^ e )
{
    Customer^ customerTmp = nullptr;

    // Store a reference to the Customer object for the row being painted.
    if ( e->RowIndex == rowInEdit )
    {
        customerTmp = this->customerInEdit;
    }
    else
    {
        customerTmp = dynamic_cast<Customer^>(this->customers[ e->RowIndex ]);
    }

    // Set the cell value to paint using the Customer object retrieved.
    int switchcase = 0;
    if ( (this->dataGridView1->Columns[ e->ColumnIndex ]->Name)->Equals( L"Company Name" ) )
        switchcase = 1;
    else
    if ( (this->dataGridView1->Columns[ e->ColumnIndex ]->Name)->Equals( L>Contact Name" ) )
        switchcase = 2;

    switch ( switchcase )
    {
        case 1:
            e->Value = customerTmp->CompanyName;
            break;

        case 2:
            e->Value = customerTmp->ContactName;
            break;
    }
}

void dataGridView1_CellValuePushed( Object^ /*sender*/,
    System::Windows::Forms::DataGridViewCellValueEventArgs^ e )
{
    Customer^ customerTmp = nullptr;

    // Store a reference to the Customer object for the row being edited.
    if ( e->RowIndex < this->customers->Count )
    {

        // If the user is editing a new row, create a new Customer object.

```

```

    if ( this->customerInEdit == nullptr )
    {
        this->customerInEdit = gcnew Customer(
            (dynamic_cast<Customer^>(this->customers[ e->RowIndex ]))->CompanyName,
            (dynamic_cast<Customer^>(this->customers[ e->RowIndex ]))->ContactName) );
    }

    customerTmp = this->customerInEdit;
    this->rowInEdit = e->RowIndex;
}
else
{
    customerTmp = this->customerInEdit;
}

// Set the appropriate Customer property to the cell value entered.
int switchcase = 0;
if ( (this->dataGridView1->Columns[ e->ColumnIndex ]->Name)->Equals( L"Company Name" ) )
    switchcase = 1;
else
if ( (this->dataGridView1->Columns[ e->ColumnIndex ]->Name)->Equals( L>Contact Name" ) )
    switchcase = 2;

switch ( switchcase )
{
    case 1:
        customerTmp->CompanyName = dynamic_cast<String^>(e->Value);
        break;

    case 2:
        customerTmp->ContactName = dynamic_cast<String^>(e->Value);
        break;
}
}

void dataGridView1_NewRowNeeded( Object^ /*sender*/,
    System::Windows::Forms::DataGridViewEventArgs^ /*e*/ )
{
}

// Create a new Customer object when the user edits
// the row for new records.
this->customerInEdit = gcnew Customer;
this->rowInEdit = this->dataGridView1->Rows->Count - 1;
}

void dataGridView1_RowValidated( Object^ /*sender*/,
    System::Windows::Forms::DataGridViewCellEventArgs^ e )
{
}

// Save row changes if any were made and release the edited
// Customer object if there is one.
if ( e->RowIndex >= this->customers->Count && e->RowIndex != this->dataGridView1->Rows->Count - 1 )
{
    // Add the new Customer object to the data store.
    this->customers->Add( this->customerInEdit );
    this->customerInEdit = nullptr;
    this->rowInEdit = -1;
}
else
if ( this->customerInEdit != nullptr && e->RowIndex < this->customers->Count )
{
    // Save the modified Customer object in the data store.
    this->customers[ e->RowIndex ] = this->customerInEdit;
}

```

```

    this->customerInEdit = nullptr;
    this->rowInEdit = -1;
}
else
{
    if ( this->dataGridView1->ContainsFocus )
    {
        this->customerInEdit = nullptr;
        this->rowInEdit = -1;
    }
}

void dataGridView1_RowDirtyStateChanged( Object^ /*sender*/,
                                         System::Windows::Forms::QuestionEventArgs^ e )
{
    if ( !rowScopeCommit )
    {

        // In cell-level commit scope, indicate whether the value
        // of the current cell has been modified.
        e->Response = this->dataGridView1->IsCurrentCellDirty;
    }
}

void dataGridView1_CancelRowEdit( Object^ /*sender*/,
                                 System::Windows::Forms::QuestionEventArgs^ /*e*/ )
{
    if ( this->rowInEdit == this->dataGridView1->Rows->Count - 2 &&
         this->rowInEdit == this->customers->Count )
    {

        // If the user has canceled the edit of a newly created row,
        // replace the corresponding Customer object with a new, empty one.
        this->customerInEdit = gcnew Customer;
    }
    else
    {

        // If the user has canceled the edit of an existing row,
        // release the corresponding Customer object.
        this->customerInEdit = nullptr;
        this->rowInEdit = -1;
    }
}

void dataGridView1_UserDeletingRow( Object^ /*sender*/,
                                  System::Windows::Forms::DataGridViewRowCancelEventArgs^ e )
{
    if ( e->Row->Index < this->customers->Count )
    {

        // If the user has deleted an existing row, remove the
        // corresponding Customer object from the data store.
        this->customers->RemoveAt( e->Row->Index );
    }

    if ( e->Row->Index == this->rowInEdit )
    {

        // If the user has deleted a newly created row, release
        // the corresponding Customer object.
        this->rowInEdit = -1;
        this->customerInEdit = nullptr;
    }
}
};


```

```
int main()
{
    Form1::Main();
}
```

```
using System;
using System.Windows.Forms;

public class Form1 : Form
{
    private DataGridView dataGridView1 = new DataGridView();

    // Declare an ArrayList to serve as the data store.
    private System.Collections.ArrayList customers =
        new System.Collections.ArrayList();

    // Declare a Customer object to store data for a row being edited.
    private Customer customerInEdit;

    // Declare a variable to store the index of a row being edited.
    // A value of -1 indicates that there is no row currently in edit.
    private int rowInEdit = -1;

    // Declare a variable to indicate the commit scope.
    // Set this value to false to use cell-level commit scope.
    private bool rowScopeCommit = true;

    [STAThreadAttribute()]
    public static void Main()
    {
        Application.Run(new Form1());
    }

    public Form1()
    {
        // Initialize the form.
        this.dataGridView1.Dock = DockStyle.Fill;
        this.Controls.Add(this.dataGridView1);
        this.Load += new EventHandler(Form1_Load);
        this.Text = "DataGridView virtual-mode demo (row-level commit scope)";
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        // Enable virtual mode.
        this.dataGridView1.VirtualMode = true;

        // Connect the virtual-mode events to event handlers.
        this.dataGridView1.CellValueNeeded += new
            DataGridViewCellValueEventHandler(dataGridView1_CellValueNeeded);
        this.dataGridView1.CellValuePushed += new
            DataGridViewCellValueEventHandler(dataGridView1_CellValuePushed);
        this.dataGridView1.NewRowNeeded += new
            DataGridViewRowEventHandler(dataGridView1_NewRowNeeded);
        this.dataGridView1.RowValidated += new
            DataGridViewCellEventHandler(dataGridView1_RowValidated);
        this.dataGridView1.RowDirtyStateNeeded += new
            QuestionEventHandler(dataGridView1_RowDirtyStateNeeded);
        this.dataGridView1.CancelRowEdit += new
            QuestionEventHandler(dataGridView1_CancelRowEdit);
        this.dataGridView1.UserDeletingRow += new
            DataGridViewRowCancelEventHandler(dataGridView1_UserDeletingRow);

        // Add columns to the DataGridView.
        DataGridViewTextBoxColumn companyNameColumn = new
```

```

        DataGridTextBoxColumn companyColumnName = new
            DataGridViewTextBoxColumn();
        companyColumnName.HeaderText = "Company Name";
        companyColumnName.Name = "Company Name";
        DataGridViewTextBoxColumn contactNameColumn = new
            DataGridViewTextBoxColumn();
        contactNameColumn.HeaderText = "Contact Name";
        contactNameColumn.Name = "Contact Name";
        this.dataGridView1.Columns.Add(companyColumnName);
        this.dataGridView1.Columns.Add(contactNameColumn);
        this.dataGridView1.AutoSizeColumnsMode =
            DataGridViewAutoSizeColumnsMode.AllCells;

        // Add some sample entries to the data store.
        this.customers.Add(new Customer(
            "Bon app'", "Laurence Lebihan"));
        this.customers.Add(new Customer(
            "Bottom-Dollar Markets", "Elizabeth Lincoln"));
        this.customers.Add(new Customer(
            "B's Beverages", "Victoria Ashworth"));

        // Set the row count, including the row for new records.
        this.dataGridView1.RowCount = 4;
    }

    private void dataGridView1_CellValueNeeded(object sender,
        System.Windows.Forms.DataGridViewCellValueEventArgs e)
    {
        // If this is the row for new records, no values are needed.
        if (e.RowIndex == this.dataGridView1.RowCount - 1) return;

        Customer customerTmp = null;

        // Store a reference to the Customer object for the row being painted.
        if (e.RowIndex == rowInEdit)
        {
            customerTmp = this.customerInEdit;
        }
        else
        {
            customerTmp = (Customer)this.customers[e.RowIndex];
        }

        // Set the cell value to paint using the Customer object retrieved.
        switch (this.dataGridView1.Columns[e.ColumnIndex].Name)
        {
            case "Company Name":
                e.Value = customerTmp.CompanyName;
                break;

            case "Contact Name":
                e.Value = customerTmp.ContactName;
                break;
        }
    }

    private void dataGridView1_CellValuePushed(object sender,
        System.Windows.Forms.DataGridViewCellValueEventArgs e)
    {
        Customer customerTmp = null;

        // Store a reference to the Customer object for the row being edited.
        if (e.RowIndex < this.customers.Count)
        {
            // If the user is editing a new row, create a new Customer object.
            if (this.customerInEdit == null)
            {
                this.customerInEdit = new Customer(
                    ((Customer)this.customers[e.RowIndex]).CompanyName,
                    //((Customer)this.customers[e.RowIndex]).ContactName));
            }
        }
    }
}

```

```

        ((Customer)this.customers[e.RowIndex]).ContactName);
    }
    customerTmp = this.customerInEdit;
    this.rowInEdit = e.RowIndex;
}
else
{
    customerTmp = this.customerInEdit;
}

// Set the appropriate Customer property to the cell value entered.
String newValue = e.Value as String;
switch (this.dataGridView1.Columns[e.ColumnIndex].Name)
{
    case "Company Name":
        customerTmp.CompanyName = newValue;
        break;

    case "Contact Name":
        customerTmp.ContactName = newValue;
        break;
}
}

private void dataGridView1_NewRowNeeded(object sender,
    System.Windows.Forms.DataGridViewRowEventArgs e)
{
    // Create a new Customer object when the user edits
    // the row for new records.
    this.customerInEdit = new Customer();
    this.rowInEdit = this.dataGridView1.Rows.Count - 1;
}

private void dataGridView1_RowValidated(object sender,
    System.Windows.Forms.DataGridViewCellEventArgs e)
{
    // Save row changes if any were made and release the edited
    // Customer object if there is one.
    if (e.RowIndex >= this.customers.Count &&
        e.RowIndex != this.dataGridView1.Rows.Count - 1)
    {
        // Add the new Customer object to the data store.
        this.customers.Add(this.customerInEdit);
        this.customerInEdit = null;
        this.rowInEdit = -1;
    }
    else if (this.customerInEdit != null &&
        e.RowIndex < this.customers.Count)
    {
        // Save the modified Customer object in the data store.
        this.customers[e.RowIndex] = this.customerInEdit;
        this.customerInEdit = null;
        this.rowInEdit = -1;
    }
    else if (this.dataGridView1.ContainsFocus)
    {
        this.customerInEdit = null;
        this.rowInEdit = -1;
    }
}

private void dataGridView1_RowDirtyStateChanged(object sender,
    System.Windows.Forms.QuestionEventArgs e)
{
    if (!rowScopeCommit)
    {
        // In cell-level commit scope, indicate whether the value
        // of the current cell has been modified.
        e.Response = this.dataGridView1.IsCurrentCellDirty;
    }
}

```

```

        }

    }

    private void dataGridView1_CancelRowEdit(object sender,
        System.Windows.Forms.QuestionEventArgs e)
    {
        if (this.rowInEdit == this.dataGridView1.Rows.Count - 2 &&
            this.rowInEdit == this.customers.Count)
        {
            // If the user has canceled the edit of a newly created row,
            // replace the corresponding Customer object with a new, empty one.
            this.customerInEdit = new Customer();
        }
        else
        {
            // If the user has canceled the edit of an existing row,
            // release the corresponding Customer object.
            this.customerInEdit = null;
            this.rowInEdit = -1;
        }
    }

    private void dataGridView1_UserDeletingRow(object sender,
        System.Windows.Forms.DataGridViewRowCancelEventArgs e)
    {
        if (e.Row.Index < this.customers.Count)
        {
            // If the user has deleted an existing row, remove the
            // corresponding Customer object from the data store.
            this.customers.RemoveAt(e.Row.Index);
        }

        if (e.Row.Index == this.rowInEdit)
        {
            // If the user has deleted a newly created row, release
            // the corresponding Customer object.
            this.rowInEdit = -1;
            this.customerInEdit = null;
        }
    }
}

public class Customer
{
    private String companyNameValue;
    private String contactNameValue;

    public Customer()
    {
        // Leave fields empty.
    }

    public Customer(String companyName, String contactName)
    {
        companyNameValue = companyName;
        contactNameValue = contactName;
    }

    public String CompanyName
    {
        get
        {
            return companyNameValue;
        }
        set
        {
            companyNameValue = value;
        }
    }
}

```

```

public String ContactName
{
    get
    {
        return contactNameValue;
    }
    set
    {
        contactNameValue = value;
    }
}

```

```

Imports System
Imports System.Windows.Forms

Public Class Form1
    Inherits Form

    Private WithEvents dataGridView1 As New DataGridView()

    ' Declare an ArrayList to serve as the data store.
    Private customers As New System.Collections.ArrayList()

    ' Declare a Customer object to store data for a row being edited.
    Private customerInEdit As Customer

    ' Declare a variable to store the index of a row being edited.
    ' A value of -1 indicates that there is no row currently in edit.
    Private rowInEdit As Integer = -1

    ' Declare a variable to indicate the commit scope.
    ' Set this value to false to use cell-level commit scope.
    Private rowScopeCommit As Boolean = True

    <STAThreadAttribute()> _
    Public Shared Sub Main()
        Application.Run(New Form1())
    End Sub

    Public Sub New()
        ' Initialize the form.
        Me.dataGridView1.Dock = DockStyle.Fill
        Me.Controls.Add(Me.dataGridView1)
        Me.Text = "DataGridView virtual-mode demo (row-level commit scope)"
    End Sub

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs) _
        Handles Me.Load

        ' Enable virtual mode.
        Me.dataGridView1.VirtualMode = True

        ' Add columns to the DataGridView.
        Dim companyNameColumn As New DataGridViewTextBoxColumn()
        With companyNameColumn
            .HeaderText = "Company Name"
            .Name = "Company Name"
        End With
        Dim contactNameColumn As New DataGridViewTextBoxColumn()
        With contactNameColumn
            .HeaderText = "Contact Name"
            .Name = "Contact Name"
        End With
        Me.dataGridView1.Columns.Add(companyNameColumn)
        Me.dataGridView1.Columns.Add(contactNameColumn)
    End Sub

```

```

Me.dataGridView1.AutoSizeColumnsMode = _
    DataGridViewAutoSizeColumnsMode.AllCells

    ' Add some sample entries to the data store.
Me.customers.Add(New Customer("Bon app'", "Laurence Lebihan"))
Me.customers.Add(New Customer("Bottom-Dollar Markets", _
    "Elizabeth Lincoln"))
Me.customers.Add(New Customer("B's Beverages", "Victoria Ashworth"))

    ' Set the row count, including the row for new records.
Me.dataGridView1.RowCount = 4

End Sub

Private Sub dataGridView1_CellValueNeeded(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DataGridViewCellValueEventArgs) _
Handles dataGridView1.CellValueNeeded

    ' If this is the row for new records, no values are needed.
If e.RowIndex = Me.dataGridView1.RowCount - 1 Then
    Return
End If

Dim customerTmp As Customer = Nothing

    ' Store a reference to the Customer object for the row being painted.
If e.RowIndex = rowInEdit Then
    customerTmp = Me.customerInEdit
Else
    customerTmp = CType(Me.customers(e.RowIndex), Customer)
End If

    ' Set the cell value to paint using the Customer object retrieved.
Select Case Me.dataGridView1.Columns(e.ColumnIndex).Name
    Case "Company Name"
        e.Value = customerTmp.CompanyName

    Case "Contact Name"
        e.Value = customerTmp.ContactName
End Select

End Sub

Private Sub dataGridView1_CellValuePushed(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DataGridViewCellValueEventArgs) _
Handles dataGridView1.CellValuePushed

Dim customerTmp As Customer = Nothing

    ' Store a reference to the Customer object for the row being edited.
If e.RowIndex < Me.customers.Count Then

    ' If the user is editing a new row, create a new Customer object.
    If Me.customerInEdit Is Nothing Then
        Me.customerInEdit = New Customer( _
            CType(Me.customers(e.RowIndex), Customer).CompanyName, _
            CType(Me.customers(e.RowIndex), Customer).ContactName)
    End If
    customerTmp = Me.customerInEdit
    Me.rowInEdit = e.RowIndex

Else
    customerTmp = Me.customerInEdit
End If

    ' Set the appropriate Customer property to the cell value entered.
Dim newValue As String = TryCast(e.Value, String)
Select Case Me.dataGridView1.Columns(e.ColumnIndex).Name
    Case "Company Name"

```

```

        customerTmp.CompanyName = newValue
    Case "Contact Name"
        customerTmp.ContactName = newValue
    End Select

End Sub

Private Sub dataGridView1_NewRowNeeded(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DataGridViewRowEventArgs) _
Handles dataGridView1.NewRowNeeded

    ' Create a new Customer object when the user edits
    ' the row for new records.
    Me.customerInEdit = New Customer()
    Me.rowInEdit = Me.dataGridView1.Rows.Count - 1

End Sub

Private Sub dataGridView1_RowValidated(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DataGridViewCellEventArgs) _
Handles dataGridView1.RowValidated

    ' Save row changes if any were made and release the edited
    ' Customer object if there is one.
    If e.RowIndex >= Me.customers.Count AndAlso _
        e.RowIndex <> Me.dataGridView1.Rows.Count - 1 Then

        ' Add the new Customer object to the data store.
        Me.customers.Add(Me.customerInEdit)
        Me.customerInEdit = Nothing
        Me.rowInEdit = -1

    ElseIf (Me.customerInEdit IsNot Nothing) AndAlso _
        e.RowIndex < Me.customers.Count Then

        ' Save the modified Customer object in the data store.
        Me.customers(e.RowIndex) = Me.customerInEdit
        Me.customerInEdit = Nothing
        Me.rowInEdit = -1

    ElseIf Me.dataGridView1.ContainsFocus Then

        Me.customerInEdit = Nothing
        Me.rowInEdit = -1

    End If

End Sub

Private Sub dataGridView1_RowDirtyStateChanged(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.QuestionEventArgs) _
Handles dataGridView1.RowDirtyStateChanged

    If Not rowScopeCommit Then

        ' In cell-level commit scope, indicate whether the value
        ' of the current cell has been modified.
        e.Response = Me.dataGridView1.IsCurrentCellDirty

    End If

End Sub

Private Sub dataGridView1_CancelRowEdit(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.QuestionEventArgs) _
Handles dataGridView1.CancelRowEdit

    If Me.rowInEdit = Me.dataGridView1.Rows.Count - 2 AndAlso _
        Me.rowInEdit = Me.customers.Count Then

```

```

' If the user has canceled the edit of a newly created row,
' replace the corresponding Customer object with a new, empty one.
Me.customerInEdit = New Customer()

Else

    ' If the user has canceled the edit of an existing row,
    ' release the corresponding Customer object.
    Me.customerInEdit = Nothing
    Me.rowInEdit = -1

End If

End Sub

Private Sub dataGridView1_UserDeletingRow(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DataGridViewRowCancelEventArgs) _
Handles dataGridView1.UserDeletingRow

If e.Row.Index < Me.customers.Count Then

    ' If the user has deleted an existing row, remove the
    ' corresponding Customer object from the data store.
    Me.customers.RemoveAt(e.Row.Index)

End If

If e.Row.Index = Me.rowInEdit Then

    ' If the user has deleted a newly created row, release
    ' the corresponding Customer object.
    Me.rowInEdit = -1
    Me.customerInEdit = Nothing

End If

End Sub

End Class

Public Class Customer

Private companyNameValue As String
Private contactNameValue As String

Public Sub New()
    ' Leave fields empty.
End Sub

Public Sub New(ByVal companyName As String, ByVal contactName As String)
    companyNameValue = companyName
    contactNameValue = contactName
End Sub

Public Property CompanyName() As String
    Get
        Return companyNameValue
    End Get
    Set(ByVal value As String)
        companyNameValue = value
    End Set
End Property

Public Property ContactName() As String
    Get
        Return contactNameValue
    End Get
    Set(ByVal value As String)

```

```
    contactNameValue = value
End Set
End Property

End Class
```

Compiling the Code

This example requires:

- References to the System and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[DataGridView](#)

[VirtualMode](#)

[CellValueNeeded](#)

[CellValuePushed](#)

[NewRowNeeded](#)

[RowValidated](#)

[RowDirtyStateChanged](#)

[CancelRowEdit](#)

[UserDeletingRow](#)

[Walkthrough: Implementing Virtual Mode in the Windows Forms DataGridView Control](#)

[Performance Tuning in the Windows Forms DataGridView Control](#)

[Virtual Mode in the Windows Forms DataGridView Control](#)

Implementing Virtual Mode with Just-In-Time Data Loading in the Windows Forms DataGridView Control

5/4/2018 • 16 min to read • [Edit Online](#)

One reason to implement virtual mode in the [DataGridView](#) control is to retrieve data only as it is needed. This is called *just-in-time data loading*.

If you are working with a very large table in a remote database, for example, you might want to avoid startup delays by retrieving only the data that is necessary for display and retrieving additional data only when the user scrolls new rows into view. If the client computers running your application have a limited amount of memory available for storing data, you might also want to discard unused data when retrieving new values from the database.

The following sections describe how to use a [DataGridView](#) control with a just-in-time cache.

To copy the code in this topic as a single listing, see [How to: Implement Virtual Mode with Just-In-Time Data Loading in the Windows Forms DataGridView Control](#).

The Form

The following code example defines a form containing a read-only [DataGridView](#) control that interacts with a [Cache](#) object through a [CellValueNeeded](#) event handler. The [Cache](#) object manages the locally stored values and uses a [DataRetriever](#) object to retrieve values from the Orders table of the sample Northwind database. The [DataRetriever](#) object, which implements the [IDataPageRetriever](#) interface required by the [Cache](#) class, is also used to initialize the [DataGridView](#) control rows and columns.

The [IDataPageRetriever](#), [DataRetriever](#), and [Cache](#) types are described later in this topic.

NOTE

Storing sensitive information, such as a password, within the connection string can affect the security of your application. Using Windows Authentication (also known as integrated security) is a more secure way to control access to a database. For more information, see [Protecting Connection Information](#).

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Windows.Forms;

public class VirtualJustInTimeDemo : System.Windows.Forms.Form
{
    private DataGridView dataGridView1 = new DataGridView();
    private Cache memoryCache;

    // Specify a connection string. Replace the given value with a
    // valid connection string for a Northwind SQL Server sample
    // database accessible to your system.
    private string connectionString =
        "Initial Catalog=NorthWind;Data Source=localhost;" +
        "Integrated Security=SSPI;Persist Security Info=False";
```

```

private string table = "Orders";

protected override void OnLoad(EventArgs e)
{
    // Initialize the form.
    this.AutoSize = true;
    this.Controls.Add(this.dataGridView1);
    this.Text = "DataGridView virtual-mode just-in-time demo";

    // Complete the initialization of the DataGridView.
    this.dataGridView1.Size = new Size(800, 250);
    this.dataGridView1.Dock = DockStyle.Fill;
    this.dataGridView1.VirtualMode = true;
    this.dataGridView1.ReadOnly = true;
    this.dataGridView1.AllowUserToAddRows = false;
    this.dataGridView1.AllowUserToOrderColumns = false;
    this.dataGridView1.SelectionMode =
        DataGridViewSelectionMode.FullRowSelect;
    this.dataGridView1.CellValueNeeded += new
        DataGridViewCellValueEventHandler(dataGridView1_CellValueNeeded);

    // Create a DataRetriever and use it to create a Cache object
    // and to initialize the DataGridView columns and rows.
    try
    {
        DataRetriever retriever =
            new DataRetriever(connectionString, table);
        memoryCache = new Cache(retriever, 16);
        foreach (DataColumn column in retriever.Columns)
        {
            dataGridView1.Columns.Add(
                column.ColumnName, column.ColumnName);
        }
        this.dataGridView1.RowCount = retriever.RowCount;
    }
    catch (SqlException)
    {
        MessageBox.Show("Connection could not be established. " +
            "Verify that the connection string is valid.");
        Application.Exit();
    }

    // Adjust the column widths based on the displayed values.
    this.dataGridView1.AutoResizeColumns(
        DataGridViewAutoSizeColumnsMode.DisplayedCells);

    base.OnLoad(e);
}

private void dataGridView1_CellValueNeeded(object sender,
    DataGridViewCellValueEventArgs e)
{
    e.Value = memoryCache.RetrieveElement(e.RowIndex, e.ColumnIndex);
}

[STAThreadAttribute()]
public static void Main()
{
    Application.Run(new VirtualJustInTimeDemo());
}
}

```

```

Imports System.Data
Imports System.Data.SqlClient
Imports System.Drawing
Imports System.Windows.Forms

```

```

Public Class VirtualJustInTimeDemo
    Inherits System.Windows.Forms.Form

    Private WithEvents dataGridView1 As New DataGridView()
    Private memoryCache As Cache

    ' Specify a connection string. Replace the given value with a
    ' valid connection string for a Northwind SQL Server sample
    ' database accessible to your system.
    Private connectionString As String = _
        "Initial Catalog=NorthWind;Data Source=localhost;" & _
        "Integrated Security=SSPI;Persist Security Info=False"
    Private table As String = "Orders"

    Private Sub VirtualJustInTimeDemo_Load( _
        ByVal sender As Object, ByVal e As EventArgs) _
        Handles Me.Load

        ' Initialize the form.
        With Me
            .AutoSize = True
            .Controls.Add(Me.dataGridView1)
            .Text = "DataGridView virtual-mode just-in-time demo"
        End With

        ' Complete the initialization of the DataGridView.
        With Me.dataGridView1
            .Size = New Size(800, 250)
            .Dock = DockStyle.Fill
            .VirtualMode = True
            .ReadOnly = True
            .AllowUserToAddRows = False
            .AllowUserToOrderColumns = False
            .SelectionMode = DataGridViewSelectionMode.FullRowSelect
        End With

        ' Create a DataRetriever and use it to create a Cache object
        ' and to initialize the DataGridView columns and rows.
        Try
            Dim retriever As New DataRetriever(connectionString, table)
            memoryCache = New Cache(retriever, 16)
            For Each column As DataColumn In retriever.Columns
                dataGridView1.Columns.Add( _
                    column.ColumnName, column.ColumnName)
            Next
            Me.dataGridView1.RowCount = retriever.RowCount
        Catch ex As SqlException
            MessageBox.Show("Connection could not be established. " & _
                "Verify that the connection string is valid.")
            Application.Exit()
        End Try

        ' Adjust the column widths based on the displayed values.
        Me.dataGridView1.AutoResizeColumns( _
            DataGridViewAutoSizeColumnsMode.DisplayedCells)

    End Sub

    Private Sub dataGridView1_CellValueNeeded( _
        ByVal sender As Object, ByVal e As DataGridViewCellValueEventArgs) _
        Handles dataGridView1.CellValueNeeded

        e.Value = memoryCache.RetrieveElement(e.RowIndex, e.ColumnIndex)

    End Sub

    <STAThreadAttribute()> _
    Public Shared Sub Main()

```

```
Application.Run(New VirtualJustInTimeDemo())
End Sub

End Class
```

The IDataPageRetriever Interface

The following code example defines the `IDataPageRetriever` interface, which is implemented by the `DataRetriever` class. The only method declared in this interface is the `SupplyPageOfData` method, which requires an initial row index and a count of the number of rows in a single page of data. These values are used by the implementer to retrieve a subset of data from a data source.

A `Cache` object uses an implementation of this interface during construction to load two initial pages of data. Whenever an uncached value is needed, the cache discards one of these pages and requests a new page containing the value from the `IDataPageRetriever`.

```
public interface IDataPageRetriever
{
    DataTable SupplyPageOfData(int lowerPageBoundary, int rowsPerPage);
}
```

```
Public Interface IDataPageRetriever

    Function SupplyPageOfData( _
        ByVal lowerPageBoundary As Integer, ByVal rowsPerPage As Integer) _
        As DataTable

End Interface
```

The DataRetriever Class

The following code example defines the `DataRetriever` class, which implements the `IDataPageRetriever` interface to retrieve pages of data from a server. The `DataRetriever` class also provides `Columns` and `RowCount` properties, which the `DataGridView` control uses to create the necessary columns and to add the appropriate number of empty rows to the `Rows` collection. Adding the empty rows is necessary so that the control will behave as though it contains all the data in the table. This means that the scroll box in the scroll bar will have the appropriate size, and the user will be able to access any row in the table. The rows are filled by the `CellValueNeeded` event handler only when they are scrolled into view.

```
public class DataRetriever : IDataPageRetriever
{
    private string tableName;
    private SqlCommand command;

    public DataRetriever(string connectionString, string tableName)
    {
        SqlConnection connection = new SqlConnection(connectionString);
        connection.Open();
        command = connection.CreateCommand();
        this.tableName = tableName;
    }

    private int rowCountValue = -1;

    public int RowCount
    {
        get
        {
```

```

        // Return the existing value if it has already been determined.
        if (rowCountValue != -1)
        {
            return rowCountValue;
        }

        // Retrieve the row count from the database.
        command.CommandText = "SELECT COUNT(*) FROM " + tableName;
        rowCountValue = (int)command.ExecuteScalar();
        return rowCountValue;
    }
}

private DataColumnCollection columnsValue;

public DataColumnCollection Columns
{
    get
    {
        // Return the existing value if it has already been determined.
        if (columnsValue != null)
        {
            return columnsValue;
        }

        // Retrieve the column information from the database.
        command.CommandText = "SELECT * FROM " + tableName;
        SqlDataAdapter adapter = new SqlDataAdapter();
        adapter.SelectCommand = command;
        DataTable table = new DataTable();
        table.Locale = System.Globalization.CultureInfo.InvariantCulture;
        adapter.FillSchema(table, SchemaType.Source);
        columnsValue = table.Columns;
        return columnsValue;
    }
}

private string commaSeparatedListOfColumnNamesValue = null;

private string CommaSeparatedListOfColumnNames
{
    get
    {
        // Return the existing value if it has already been determined.
        if (commaSeparatedListOfColumnNamesValue != null)
        {
            return commaSeparatedListOfColumnNamesValue;
        }

        // Store a list of column names for use in the
        // SupplyPageOfData method.
        System.Text.StringBuilder commaSeparatedColumnNames =
            new System.Text.StringBuilder();
        bool firstColumn = true;
        foreach (DataColumn column in Columns)
        {
            if (!firstColumn)
            {
                commaSeparatedColumnNames.Append(", ");
            }
            commaSeparatedColumnNames.Append(column.ColumnName);
            firstColumn = false;
        }

        commaSeparatedListOfColumnNamesValue =
            commaSeparatedColumnNames.ToString();
        return commaSeparatedListOfColumnNamesValue;
    }
}

```

```

// Declare variables to be reused by the SupplyPageOfData method.
private string columnToSortBy;
private SqlDataAdapter adapter = new SqlDataAdapter();

public DataTable SupplyPageOfData(int lowerPageBoundary, int rowsPerPage)
{
    // Store the name of the ID column. This column must contain unique
    // values so the SQL below will work properly.
    if (columnToSortBy == null)
    {
        columnToSortBy = this.Columns[0].ColumnName;
    }

    if (!this.Columns[columnToSortBy].Unique)
    {
        throw new InvalidOperationException(String.Format(
            "Column {0} must contain unique values.", columnToSortBy));
    }

    // Retrieve the specified number of rows from the database, starting
    // with the row specified by the lowerPageBoundary parameter.
    command.CommandText = "Select Top " + rowsPerPage + " "
        + CommaSeparatedListOfColumnNames + " From " + tableName +
        " WHERE " + columnToSortBy + " NOT IN (SELECT TOP " +
        lowerPageBoundary + " " + columnToSortBy + " From " +
        tableName + " Order By " + columnToSortBy +
        ") Order By " + columnToSortBy;
    adapter.SelectCommand = command;

    DataTable table = new DataTable();
    table.Locale = System.Globalization.CultureInfo.InvariantCulture;
    adapter.Fill(table);
    return table;
}

}

```

```

Public Class DataRetriever
    Implements IDataPageRetriever

    Private tableName As String
    Private command As SqlCommand

    Public Sub New( _
        ByVal connectionString As String, ByVal tableName As String)

        Dim connection As New SqlConnection(connectionString)
        connection.Open()
        command = connection.CreateCommand()
        Me.tableName = tableName

    End Sub

    Private rowCountValue As Integer = -1

    Public ReadOnly Property RowCount() As Integer
        Get
            ' Return the existing value if it has already been determined.
            If Not rowCountValue = -1 Then
                Return rowCountValue
            End If

            ' Retrieve the row count from the database.
            command.CommandText = "SELECT COUNT(*) FROM " & tableName
            rowCountValue = CInt(command.ExecuteScalar())
            Return rowCountValue
        End Get
    End Property

```

```

    End Get
End Property

Private columnsValue As DataColumnCollection

Public ReadOnly Property Columns() As DataColumnCollection
Get
    ' Return the existing value if it has already been determined.
    If columnsValue IsNot Nothing Then
        Return columnsValue
    End If

    ' Retrieve the column information from the database.
    command.CommandText = "SELECT * FROM " & tableName
    Dim adapter As New SqlDataAdapter()
    adapter.SelectCommand = command
    Dim table As New DataTable()
    table.Locale = System.Globalization.CultureInfo.InvariantCulture
    adapter.FillSchema(table, SchemaType.Source)
    columnsValue = table.Columns
    Return columnsValue
End Get
End Property

Private commaSeparatedListOfColumnNamesValue As String = Nothing

Public ReadOnly Property CommaSeparatedListOfColumnNames() As String
Get
    ' Return the existing value if it has already been determined.
    If commaSeparatedListOfColumnNamesValue IsNot Nothing Then
        Return commaSeparatedListOfColumnNamesValue
    End If

    ' Store a list of column names for use in the
    ' SupplyPageOfData method.
    Dim commaSeparatedColumnNames As New System.Text.StringBuilder()
    Dim firstColumn As Boolean = True
    For Each column As DataColumn In Columns
        If Not firstColumn Then
            commaSeparatedColumnNames.Append(", ")
        End If
        commaSeparatedColumnNames.Append(column.ColumnName)
        firstColumn = False
    Next

    commaSeparatedListOfColumnNamesValue = _
        commaSeparatedColumnNames.ToString()
    Return commaSeparatedListOfColumnNamesValue
End Get
End Property

' Declare variables to be reused by the SupplyPageOfData method.
Private columnToSortBy As String
Private adapter As New SqlDataAdapter()

Public Function SupplyPageOfData( _
    ByVal lowerPageBoundary As Integer, ByVal rowsPerPage As Integer) _
As DataTable Implements IDataPageRetriever.SupplyPageOfData

    ' Store the name of the ID column. This column must contain unique
    ' values so the SQL below will work properly.
    If columnToSortBy Is Nothing Then
        columnToSortBy = Me.Columns(0).ColumnName
    End If

    If Not Me.Columns(columnToSortBy).Unique Then
        Throw New InvalidOperationException(String.Format( _
            "Column {0} must contain unique values.", columnToSortBy))
    End If

```

```

    ' Retrieve the specified number of rows from the database, starting
    ' with the row specified by the lowerPageBoundary parameter.
    command.CommandText = _
        "Select Top " & rowsPerPage & " " & _
        CommaSeparatedListOfColumnNames & " From " & tableName & _
        " WHERE " & columnToSortBy & " NOT IN (SELECT TOP " & _
        lowerPageBoundary & " " & columnToSortBy & " From " & _
        tableName & " Order By " & columnToSortBy & _
        ") Order By " & columnToSortBy
    adapter.SelectCommand = command

    Dim table As New DataTable()
    table.Locale = System.Globalization.CultureInfo.InvariantCulture
    adapter.Fill(table)
    Return table

End Function

End Class

```

The Cache Class

The following code example defines the `Cache` class, which manages two pages of data populated through an `IDataPageRetriever` implementation. The `Cache` class defines an inner `DataPage` structure, which contains a `DataTable` to store the values in a single cache page and which calculates the row indexes that represent the upper and lower boundaries of the page.

The `Cache` class loads two pages of data at construction time. Whenever the `CellValueNeeded` event requests a value, the `Cache` object determines if the value is available in one of its two pages and, if so, returns it. If the value is not available locally, the `Cache` object determines which of its two pages is farthest from the currently displayed rows and replaces the page with a new one containing the requested value, which it then returns.

Assuming that the number of rows in a data page is the same as the number of rows that can be displayed on screen at once, this model allows users paging through the table to efficiently return to the most recently viewed page.

```

public class Cache
{
    private static int RowsPerPage;

    // Represents one page of data.
    public struct DataPage
    {
        public DataTable table;
        private int lowestIndexValue;
        private int highestIndexValue;

        public DataPage(DataTable table, int rowIndex)
        {
            this.table = table;
            lowestIndexValue = MapToLowerBoundary(rowIndex);
            highestIndexValue = MapToUpperBoundary(rowIndex);
            System.Diagnostics.Debug.Assert(lowestIndexValue >= 0);
            System.Diagnostics.Debug.Assert(highestIndexValue >= 0);
        }

        public int LowestIndex
        {
            get
            {
                return lowestIndexValue;
            }
        }
    }
}

```

```

        }

    public int HighestIndex
    {
        get
        {
            return highestIndexValue;
        }
    }

    public static int MapToLowerBoundary(int rowIndex)
    {
        // Return the lowest index of a page containing the given index.
        return (rowIndex / RowsPerPage) * RowsPerPage;
    }

    private static int MapToUpperBoundary(int rowIndex)
    {
        // Return the highest index of a page containing the given index.
        return MapToLowerBoundary(rowIndex) + RowsPerPage - 1;
    }
}

private DataPage[] cachePages;
private IDataPageRetriever dataSupply;

public Cache(IDataPageRetriever dataSupplier, int rowsPerPage)
{
    dataSupply = dataSupplier;
    Cache.RowsPerPage = rowsPerPage;
    LoadFirstTwoPages();
}

// Sets the value of the element parameter if the value is in the cache.
private bool IfPageCached_ThenSetElement(int rowIndex,
    int columnIndex, ref string element)
{
    if (IsRowCachedInPage(0, rowIndex))
    {
        element = cachePages[0].table
            .Rows[rowIndex % RowsPerPage][columnIndex].ToString();
        return true;
    }
    else if (IsRowCachedInPage(1, rowIndex))
    {
        element = cachePages[1].table
            .Rows[rowIndex % RowsPerPage][columnIndex].ToString();
        return true;
    }

    return false;
}

public string RetrieveElement(int rowIndex, int columnIndex)
{
    string element = null;

    if (IfPageCached_ThenSetElement(rowIndex, columnIndex, ref element))
    {
        return element;
    }
    else
    {
        return RetrieveData_CacheIt_ThenReturnElement(
            rowIndex, columnIndex);
    }
}

private void LoadFirstTwoPages()

```

```

private void LoadData()
{
    cachePages = new DataPage[]{ 
        new DataPage(dataSupply.SupplyPageOfData(
            DataPage.MapToLowerBoundary(0), RowsPerPage), 0),
        new DataPage(dataSupply.SupplyPageOfData(
            DataPage.MapToLowerBoundary(RowsPerPage),
            RowsPerPage), RowsPerPage)};
    }

private string RetrieveData_CacheIt_ThenReturnElement(
    int rowIndex, int columnIndex)
{
    // Retrieve a page worth of data containing the requested value.
    DataTable table = dataSupply.SupplyPageOfData(
        DataPage.MapToLowerBoundary(rowIndex), RowsPerPage);

    // Replace the cached page furthest from the requested cell
    // with a new page containing the newly retrieved data.
    cachePages[GetIndexToUnusedPage(rowIndex)] = new DataPage(table, rowIndex);

    return RetrieveElement(rowIndex, columnIndex);
}

// Returns the index of the cached page most distant from the given index
// and therefore least likely to be reused.
private int GetIndexToUnusedPage(int rowIndex)
{
    if (rowIndex > cachePages[0].HighestIndex &&
        rowIndex > cachePages[1].HighestIndex)
    {
        int offsetFromPage0 = rowIndex - cachePages[0].HighestIndex;
        int offsetFromPage1 = rowIndex - cachePages[1].HighestIndex;
        if (offsetFromPage0 < offsetFromPage1)
        {
            return 1;
        }
        return 0;
    }
    else
    {
        int offsetFromPage0 = cachePages[0].LowestIndex - rowIndex;
        int offsetFromPage1 = cachePages[1].LowestIndex - rowIndex;
        if (offsetFromPage0 < offsetFromPage1)
        {
            return 1;
        }
        return 0;
    }
}

// Returns a value indicating whether the given row index is contained
// in the given DataPage.
private bool IsRowCachedInPage(int pageNumber, int rowIndex)
{
    return rowIndex <= cachePages[pageNumber].HighestIndex &&
        rowIndex >= cachePages[pageNumber].LowestIndex;
}
}

```

Public Class Cache

```

Private Shared RowsPerPage As Integer

' Represents one page of data.
Public Structure DataPage

```

```

Public table As DataTable
Private lowestIndexValue As Integer
Private highestIndexValue As Integer

Public Sub New(ByVal table As DataTable, ByVal rowIndex As Integer)

    Me.table = table
    lowestIndexValue = MapToLowerBoundary(rowIndex)
    highestIndexValue = MapToUpperBoundary(rowIndex)
    System.Diagnostics.Debug.Assert(lowestIndexValue >= 0)
    System.Diagnostics.Debug.Assert(highestIndexValue >= 0)

End Sub

Public ReadOnly Property LowestIndex() As Integer
    Get
        Return lowestIndexValue
    End Get
End Property

Public ReadOnly Property HighestIndex() As Integer
    Get
        Return highestIndexValue
    End Get
End Property

Public Shared Function MapToLowerBoundary( _
    ByVal rowIndex As Integer) As Integer

    ' Return the lowest index of a page containing the given index.
    Return (rowIndex \ RowsPerPage) * RowsPerPage

End Function

Private Shared Function MapToUpperBoundary( _
    ByVal rowIndex As Integer) As Integer

    ' Return the highest index of a page containing the given index.
    Return MapToLowerBoundary(rowIndex) + RowsPerPage - 1

End Function

End Structure

Private cachePages As DataPage()
Private dataSupply As IDataPageRetriever

Public Sub New(ByVal dataSupplier As IDataPageRetriever, _
    ByVal rowsPerPage As Integer)

    dataSupply = dataSupplier
    Cache.RowsPerPage = rowsPerPage
    LoadFirstTwoPages()

End Sub

' Sets the value of the element parameter if the value is in the cache.
Private Function IfPageCached_ThenSetElement(ByVal rowIndex As Integer, _
    ByVal columnIndex As Integer, ByRef element As String) As Boolean

    If IsRowCachedInPage(0, rowIndex) Then
        element = cachePages(0).table.Rows(rowIndex Mod RowsPerPage) _
            .Item(columnIndex).ToString()
        Return True
    ElseIf IsRowCachedInPage(1, rowIndex) Then
        element = cachePages(1).table.Rows(rowIndex Mod RowsPerPage) _
            .Item(columnIndex).ToString()
        Return True
    End If
End Function

```

```

        Return True
    End If

    Return False

End Function

Public Function RetrieveElement(ByVal rowIndex As Integer, _
                               ByVal columnIndex As Integer) As String

    Dim element As String = Nothing
    If IfPageCached_ThenSetElement(rowIndex, columnIndex, element) Then
        Return element
    Else
        Return RetrieveData_CacheIt_ThenReturnElement( _
            rowIndex, columnIndex)
    End If

End Function

Private Sub LoadFirstTwoPages()

    cachePages = New DataPage() { _
        New DataPage(dataSupply.SupplyPageOfData( _
            DataPage.MapToLowerBoundary(0), RowsPerPage), 0), _
        New DataPage(dataSupply.SupplyPageOfData( _
            DataPage.MapToLowerBoundary(RowsPerPage), _
            RowsPerPage), RowsPerPage) _
    }

End Sub

Private Function RetrieveData_CacheIt_ThenReturnElement( _
    ByVal rowIndex As Integer, ByVal columnIndex As Integer) As String

    ' Retrieve a page worth of data containing the requested value.
    Dim table As DataTable = dataSupply.SupplyPageOfData( _
        DataPage.MapToLowerBoundary(rowIndex), RowsPerPage)

    ' Replace the cached page furthest from the requested cell
    ' with a new page containing the newly retrieved data.
    cachePages(GetIndexToUnusedPage(rowIndex)) = _
        New DataPage(table, rowIndex)

    Return RetrieveElement(rowIndex, columnIndex)

End Function

' Returns the index of the cached page most distant from the given index
' and therefore least likely to be reused.
Private Function GetIndexToUnusedPage(ByVal rowIndex As Integer) _
    As Integer

    If rowIndex > cachePages(0).HighestIndex AndAlso _
        rowIndex > cachePages(1).HighestIndex Then

        Dim offsetFromPage0 As Integer = _
            rowIndex - cachePages(0).HighestIndex
        Dim offsetFromPage1 As Integer = _
            rowIndex - cachePages(1).HighestIndex
        If offsetFromPage0 < offsetFromPage1 Then
            Return 1
        End If
        Return 0
    Else
        Dim offsetFromPage0 As Integer = _
            cachePages(0).LowestIndex - rowIndex
        Dim offsetFromPage1 As Integer = _
            cachePages(1).LowestIndex - rowIndex
        If offsetFromPage0 < offsetFromPage1 Then
            Return 1
        End If
        Return 0
    End If
End Function

```

```

    If offsetFromPage0 < offsetFromPage1 Then
        Return 1
    End If
    Return 0
End If

End Function

' Returns a value indicating whether the given row index is contained
' in the given DataPage.
Private Function IsRowCachedInPage( _
    ByVal pageNumber As Integer, ByVal rowIndex As Integer) As Boolean

    Return rowIndex <= cachePages(pageNumber).HighestIndex AndAlso _
        rowIndex >= cachePages(pageNumber).LowestIndex

End Function

End Class

```

Additional Considerations

The previous code examples are provided as a demonstration of just-in-time data loading. You will need to modify the code for your own needs to achieve maximum efficiency. At minimum, you will need to choose an appropriate value for the number of rows per page of data in the cache. This value is passed into the `Cache` constructor. The number of rows per page should be no less than the number of rows that can be displayed simultaneously in your [DataGridView](#) control.

For best results, you will need to conduct performance testing and usability testing to determine the requirements of your system and your users. Several factors that you will need to take into consideration include the amount of memory in the client machines running your application, the available bandwidth of the network connection used, and the latency of the server used. The bandwidth and latency should be determined at times of peak usage.

To improve the scrolling performance of your application, you can increase the amount of data stored locally. To improve startup time, however, you must avoid loading too much data initially. You may want to modify the `Cache` class to increase the number of data pages it can store. Using more data pages can improve scrolling efficiency, but you will need to determine the ideal number of rows in a data page, depending on the available bandwidth and the server latency. With smaller pages, the server will be accessed more frequently, but will take less time to return the requested data. If latency is more of an issue than bandwidth, you may want to use larger data pages.

See Also

- [DataGridView](#)
- [VirtualMode](#)
- [Performance Tuning in the Windows Forms DataGridView Control](#)
- [Best Practices for Scaling the Windows Forms DataGridView Control](#)
- [Virtual Mode in the Windows Forms DataGridView Control](#)
- [Walkthrough: Implementing Virtual Mode in the Windows Forms DataGridView Control](#)
- [How to: Implement Virtual Mode with Just-In-Time Data Loading in the Windows Forms DataGridView Control](#)

How to: Implement Virtual Mode with Just-In-Time Data Loading in the Windows Forms DataGridView Control

5/4/2018 • 12 min to read • [Edit Online](#)

The following code example shows how to use virtual mode in the [DataGridView](#) control with a data cache that loads data from a server only when it is needed. This example is described in detail in [Implementing Virtual Mode with Just-In-Time Data Loading in the Windows Forms DataGridView Control](#).

Example

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Windows.Forms;

public class VirtualJustInTimeDemo : System.Windows.Forms.Form
{
    private DataGridView dataGridView1 = new DataGridView();
    private Cache memoryCache;

    // Specify a connection string. Replace the given value with a
    // valid connection string for a Northwind SQL Server sample
    // database accessible to your system.
    private string connectionString =
        "Initial Catalog=NorthWind;Data Source=localhost;" +
        "Integrated Security=SSPI;Persist Security Info=False";
    private string table = "Orders";

    protected override void OnLoad(EventArgs e)
    {
        // Initialize the form.
        this.AutoSize = true;
        this.Controls.Add(this.dataGridView1);
        this.Text = "DataGridView virtual-mode just-in-time demo";

        // Complete the initialization of the DataGridView.
        this.dataGridView1.Size = new Size(800, 250);
        this.dataGridView1.Dock = DockStyle.Fill;
        this.dataGridView1.VirtualMode = true;
        this.dataGridView1.ReadOnly = true;
        this.dataGridView1.AllowUserToAddRows = false;
        this.dataGridView1.AllowUserToOrderColumns = false;
        this.dataGridView1.SelectionMode =
            DataGridViewSelectionMode.FullRowSelect;
        this.dataGridView1.CellValueNeeded += new
            DataGridViewCellValueEventHandler(dataGridView1_CellValueNeeded);

        // Create a DataRetriever and use it to create a Cache object
        // and to initialize the DataGridView columns and rows.
        try
        {
            DataRetriever retriever =
                new DataRetriever(connectionString, table);
            memoryCache = new Cache(retriever, 16);
            foreach (DataColumn column in retriever.Columns)
            {

```

```

        dataGridView1.Columns.Add(
            column.ColumnName, column.ColumnName);
    }
    this.dataGridView1.RowCount = retriever.RowCount;
}
catch (SqlException)
{
    MessageBox.Show("Connection could not be established. " +
        "Verify that the connection string is valid.");
    Application.Exit();
}

// Adjust the column widths based on the displayed values.
this.dataGridView1.AutoResizeColumns(
    DataGridViewAutoSizeColumnsMode.DisplayedCells);

base.OnLoad(e);
}

private void dataGridView1_CellValueNeeded(object sender,
    DataGridViewCellValueEventArgs e)
{
    e.Value = memoryCache.RetrieveElement(e.RowIndex, e.ColumnIndex);
}

[STAThreadAttribute()]
public static void Main()
{
    Application.Run(new VirtualJustInTimeDemo());
}

}

public interface IDataPageRetriever
{
    DataTable SupplyPageOfData(int lowerPageBoundary, int rowsPerPage);
}

public class DataRetriever : IDataPageRetriever
{
    private string tableName;
    private SqlCommand command;

    public DataRetriever(string connectionString, string tableName)
    {
        SqlConnection connection = new SqlConnection(connectionString);
        connection.Open();
        command = connection.CreateCommand();
        this.tableName = tableName;
    }

    private int rowCountValue = -1;

    public int RowCount
    {
        get
        {
            // Return the existing value if it has already been determined.
            if (rowCountValue != -1)
            {
                return rowCountValue;
            }

            // Retrieve the row count from the database.
            command.CommandText = "SELECT COUNT(*) FROM " + tableName;
            rowCountValue = (int)command.ExecuteScalar();
            return rowCountValue;
        }
    }
}

```

```
'
```

```
    private DataColumnCollection columnsValue;
```

```
    public DataColumnCollection Columns
    {
        get
        {
            // Return the existing value if it has already been determined.
            if (columnsValue != null)
            {
                return columnsValue;
            }

            // Retrieve the column information from the database.
            command.CommandText = "SELECT * FROM " + tableName;
            SqlDataAdapter adapter = new SqlDataAdapter();
            adapter.SelectCommand = command;
            DataTable table = new DataTable();
            table.Locale = System.Globalization.CultureInfo.InvariantCulture;
            adapter.FillSchema(table, SchemaType.Source);
            columnsValue = table.Columns;
            return columnsValue;
        }
    }

    private string commaSeparatedListOfColumnNamesValue = null;

    private string CommaSeparatedListOfColumnNames
    {
        get
        {
            // Return the existing value if it has already been determined.
            if (commaSeparatedListOfColumnNamesValue != null)
            {
                return commaSeparatedListOfColumnNamesValue;
            }

            // Store a list of column names for use in the
            // SupplyPageOfData method.
            System.Text.StringBuilder commaSeparatedColumnNames =
                new System.Text.StringBuilder();
            bool firstColumn = true;
            foreach (DataColumn column in Columns)
            {
                if (!firstColumn)
                {
                    commaSeparatedColumnNames.Append(", ");
                }
                commaSeparatedColumnNames.Append(column.ColumnName);
                firstColumn = false;
            }

            commaSeparatedListOfColumnNamesValue =
                commaSeparatedColumnNames.ToString();
            return commaSeparatedListOfColumnNamesValue;
        }
    }

    // Declare variables to be reused by the SupplyPageOfData method.
    private string columnToSortBy;
    private SqlDataAdapter adapter = new SqlDataAdapter();

    public DataTable SupplyPageOfData(int lowerPageBoundary, int rowsPerPage)
    {
        // Store the name of the ID column. This column must contain unique
        // values so the SQL below will work properly.
        if (columnToSortBy == null)
        {
            columnToSortBy = this.Columns[0].ColumnName;
        }
    }
}
```

```

        columnToSortBy = this.Columns[columnToSortBy].ColumnName,
    }

    if (!this.Columns[columnToSortBy].Unique)
    {
        throw new InvalidOperationException(String.Format(
            "Column {0} must contain unique values.", columnToSortBy));
    }

    // Retrieve the specified number of rows from the database, starting
    // with the row specified by the lowerPageBoundary parameter.
    command.CommandText = "Select Top " + rowsPerPage + " " +
        CommaSeparatedListOfColumnNames + " From " + tableName +
        " WHERE " + columnToSortBy + " NOT IN (SELECT TOP " +
        lowerPageBoundary + " " + columnToSortBy + " From " +
        tableName + " Order By " + columnToSortBy +
        ") Order By " + columnToSortBy;
    adapter.SelectCommand = command;

    DataTable table = new DataTable();
    table.Locale = System.Globalization.CultureInfo.InvariantCulture;
    adapter.Fill(table);
    return table;
}

public class Cache
{
    private static int RowsPerPage;

    // Represents one page of data.
    public struct DataPage
    {
        public DataTable table;
        private int lowestIndexValue;
        private int highestIndexValue;

        public DataPage(DataTable table, int rowIndex)
        {
            this.table = table;
            lowestIndexValue = MapToLowerBoundary(rowIndex);
            highestIndexValue = MapToUpperBoundary(rowIndex);
            System.Diagnostics.Debug.Assert(lowestIndexValue >= 0);
            System.Diagnostics.Debug.Assert(highestIndexValue >= 0);
        }

        public int LowestIndex
        {
            get
            {
                return lowestIndexValue;
            }
        }

        public int HighestIndex
        {
            get
            {
                return highestIndexValue;
            }
        }

        public static int MapToLowerBoundary(int rowIndex)
        {
            // Return the lowest index of a page containing the given index.
            return (rowIndex / RowsPerPage) * RowsPerPage;
        }
    }
}

```

```

private static int MapToUpperBoundary(int rowIndex)
{
    // Return the highest index of a page containing the given index.
    return MapToLowerBoundary(rowIndex) + RowsPerPage - 1;
}

private DataPage[] cachePages;
private IDataPageRetriever dataSupply;

public Cache(IDataPageRetriever dataSupplier, int rowsPerPage)
{
    dataSupply = dataSupplier;
    Cache.RowsPerPage = rowsPerPage;
    LoadFirstTwoPages();
}

// Sets the value of the element parameter if the value is in the cache.
private bool IfPageCached_ThenSetElement(int rowIndex,
    int columnIndex, ref string element)
{
    if (IsRowCachedInPage(0, rowIndex))
    {
        element = cachePages[0].table
            .Rows[rowIndex % RowsPerPage][columnIndex].ToString();
        return true;
    }
    else if (IsRowCachedInPage(1, rowIndex))
    {
        element = cachePages[1].table
            .Rows[rowIndex % RowsPerPage][columnIndex].ToString();
        return true;
    }

    return false;
}

public string RetrieveElement(int rowIndex, int columnIndex)
{
    string element = null;

    if (IfPageCached_ThenSetElement(rowIndex, columnIndex, ref element))
    {
        return element;
    }
    else
    {
        return RetrieveData_CacheIt_ThenReturnElement(
            rowIndex, columnIndex);
    }
}

private void LoadFirstTwoPages()
{
    cachePages = new DataPage[]{
        new DataPage(dataSupply.SupplyPageOfData(
            DataPage.MapToLowerBoundary(0), RowsPerPage), 0),
        new DataPage(dataSupply.SupplyPageOfData(
            DataPage.MapToLowerBoundary(RowsPerPage),
            RowsPerPage), RowsPerPage)};
}

private string RetrieveData_CacheIt_ThenReturnElement(
    int rowIndex, int columnIndex)
{
    // Retrieve a page worth of data containing the requested value.
    DataTable table = dataSupply.SupplyPageOfData(
        DataPage.MapToLowerBoundary(rowIndex), RowsPerPage);
}

```

```

        // Replace the cached page furthest from the requested cell
        // with a new page containing the newly retrieved data.
        cachePages[GetIndexToUnusedPage(rowIndex)] = new DataPage(table, rowIndex);

        return RetrieveElement(rowIndex, columnIndex);
    }

    // Returns the index of the cached page most distant from the given index
    // and therefore least likely to be reused.
    private int GetIndexToUnusedPage(int rowIndex)
    {
        if (rowIndex > cachePages[0].HighestIndex &&
            rowIndex > cachePages[1].HighestIndex)
        {
            int offsetFromPage0 = rowIndex - cachePages[0].HighestIndex;
            int offsetFromPage1 = rowIndex - cachePages[1].HighestIndex;
            if (offsetFromPage0 < offsetFromPage1)
            {
                return 1;
            }
            return 0;
        }
        else
        {
            int offsetFromPage0 = cachePages[0].LowestIndex - rowIndex;
            int offsetFromPage1 = cachePages[1].LowestIndex - rowIndex;
            if (offsetFromPage0 < offsetFromPage1)
            {
                return 1;
            }
            return 0;
        }
    }

    // Returns a value indicating whether the given row index is contained
    // in the given DataPage.
    private bool IsRowCachedInPage(int pageNumber, int rowIndex)
    {
        return rowIndex <= cachePages[pageNumber].HighestIndex &&
               rowIndex >= cachePages[pageNumber].LowestIndex;
    }
}

```

```

Imports System.Data
Imports System.Data.SqlClient
Imports System.Drawing
Imports System.Windows.Forms

Public Class VirtualJustInTimeDemo
    Inherits System.Windows.Forms.Form

    Private WithEvents dataGridView1 As New DataGridView()
    Private memoryCache As Cache

    ' Specify a connection string. Replace the given value with a
    ' valid connection string for a Northwind SQL Server sample
    ' database accessible to your system.
    Private connectionString As String =
        "Initial Catalog=NorthWind;Data Source=localhost;" & _
        "Integrated Security=SSPI;Persist Security Info=False"
    Private table As String = "Orders"

    Private Sub VirtualJustInTimeDemo_Load( _
        ByVal sender As Object, ByVal e As EventArgs) _
        Handles Me.Load

```

```

' Initialize the form.
With Me
    .AutoSize = True
    .Controls.Add(Me.dataGridView1)
    .Text = "DataGridView virtual-mode just-in-time demo"
End With

' Complete the initialization of the DataGridView.
With Me.dataGridView1
    .Size = New Size(800, 250)
    .Dock = DockStyle.Fill
    .VirtualMode = True
    .ReadOnly = True
    .AllowUserToAddRows = False
    .AllowUserToOrderColumns = False
    .SelectionMode = DataGridViewSelectionMode.FullRowSelect
End With

' Create a DataRetriever and use it to create a Cache object
' and to initialize the DataGridView columns and rows.
Try
    Dim retriever As New DataRetriever(connectionString, table)
    memoryCache = New Cache(retriever, 16)
    For Each column As DataColumn In retriever.Columns
        dataGridView1.Columns.Add(
            column.ColumnName, column.ColumnName)
    Next
    Me.dataGridView1.RowCount = retriever.RowCount
Catch ex As SqlException
    MessageBox.Show("Connection could not be established. " & _
        "Verify that the connection string is valid.")
    Application.Exit()
End Try

' Adjust the column widths based on the displayed values.
Me.dataGridView1.AutoResizeColumns( _
    DataGridViewAutoSizeColumnsMode.DisplayedCells)

End Sub

Private Sub dataGridView1_CellValueNeeded( _
    ByVal sender As Object, ByVal e As DataGridViewCellValueEventArgs) _
Handles dataGridView1.CellValueNeeded

    e.Value = memoryCache.RetrieveElement(e.RowIndex, e.ColumnIndex)

End Sub

<STAThreadAttribute()> _
Public Shared Sub Main()
    Application.Run(New VirtualJustInTimeDemo())
End Sub

End Class

Public Interface IDataPageRetriever

Function SupplyPageOfData( _
    ByVal lowerPageBoundary As Integer, ByVal rowsPerPage As Integer) _
    As DataTable

End Interface

Public Class DataRetriever
    Implements IDataPageRetriever

    Private tableName As String
    Private command As SqlCommand

```

```

Public Sub New( _
    ByVal connectionString As String, ByVal tableName As String)

    Dim connection As New SqlConnection(connectionString)
    connection.Open()
    command = connection.CreateCommand()
    Me.tableName = tableName

End Sub

Private rowCountValue As Integer = -1

Public ReadOnly Property RowCount() As Integer
    Get
        ' Return the existing value if it has already been determined.
        If Not rowCountValue = -1 Then
            Return rowCountValue
        End If

        ' Retrieve the row count from the database.
        command.CommandText = "SELECT COUNT(*) FROM " & tableName
        rowCountValue = CInt(command.ExecuteScalar())
        Return rowCountValue
    End Get
End Property

Private columnsValue As DataColumnCollection

Public ReadOnly Property Columns() As DataColumnCollection
    Get
        ' Return the existing value if it has already been determined.
        If columnsValue IsNot Nothing Then
            Return columnsValue
        End If

        ' Retrieve the column information from the database.
        command.CommandText = "SELECT * FROM " & tableName
        Dim adapter As New SqlDataAdapter()
        adapter.SelectCommand = command
        Dim table As New DataTable()
        table.Locale = System.Globalization.CultureInfo.InvariantCulture
        adapter.FillSchema(table, SchemaType.Source)
        columnsValue = table.Columns
        Return columnsValue
    End Get
End Property

Private commaSeparatedListOfColumnNamesValue As String = Nothing

Public ReadOnly Property CommaSeparatedListOfColumnNames() As String
    Get
        ' Return the existing value if it has already been determined.
        If commaSeparatedListOfColumnNamesValue IsNot Nothing Then
            Return commaSeparatedListOfColumnNamesValue
        End If

        ' Store a list of column names for use in the
        ' SupplyPageOfData method.
        Dim commaSeparatedColumnNames As New System.Text.StringBuilder()
        Dim firstColumn As Boolean = True
        For Each column As DataColumn In Columns
            If Not firstColumn Then
                commaSeparatedColumnNames.Append(", ")
            End If
            commaSeparatedColumnNames.Append(column.ColumnName)
            firstColumn = False
        Next
        Return commaSeparatedColumnNames.ToString()
    End Get
End Property

```

```

        commaSeparatedListOfColumnNamesValue = _
            commaSeparatedColumnNames.ToString()
        Return commaSeparatedListOfColumnNamesValue
    End Get
End Property

' Declare variables to be reused by the SupplyPageOfData method.
Private columnToSortBy As String
Private adapter As New SqlDataAdapter()

Public Function SupplyPageOfData( _
    ByVal lowerPageBoundary As Integer, ByVal rowsPerPage As Integer) _
As DataTable Implements IDataPageRetriever.SupplyPageOfData

    ' Store the name of the ID column. This column must contain unique
    ' values so the SQL below will work properly.
    If columnToSortBy Is Nothing Then
        columnToSortBy = Me.Columns(0).ColumnName
    End If

    If Not Me.Columns(columnToSortBy).Unique Then
        Throw New InvalidOperationException(String.Format( _
            "Column {0} must contain unique values.", columnToSortBy))
    End If

    ' Retrieve the specified number of rows from the database, starting
    ' with the row specified by the lowerPageBoundary parameter.
    command.CommandText = _
        "Select Top " & rowsPerPage & " " & _
        CommaSeparatedListOfColumnNames & " From " & tableName & _
        " WHERE " & columnToSortBy & " NOT IN (SELECT TOP " & _
        lowerPageBoundary & " " & columnToSortBy & " From " & _
        tableName & " Order By " & columnToSortBy & _
        ") Order By " & columnToSortBy
    adapter.SelectCommand = command

    Dim table As New DataTable()
    table.Locale = System.Globalization.CultureInfo.InvariantCulture
    adapter.Fill(table)
    Return table

End Function

End Class

Public Class Cache

    Private Shared RowsPerPage As Integer

    ' Represents one page of data.
    Public Structure DataPage

        Public table As DataTable
        Private lowestIndexValue As Integer
        Private highestIndexValue As Integer

        Public Sub New(ByVal table As DataTable, ByVal rowIndex As Integer)

            Me.table = table
            lowestIndexValue = MapToLowerBoundary(rowIndex)
            highestIndexValue = MapToUpperBoundary(rowIndex)
            System.Diagnostics.Debug.Assert(lowestIndexValue >= 0)
            System.Diagnostics.Debug.Assert(highestIndexValue >= 0)

        End Sub

        Public ReadOnly Property LowestIndex() As Integer
            Get
                Return lowestIndexValue
            End Get
        End Property

        Public ReadOnly Property HighestIndex() As Integer
            Get
                Return highestIndexValue
            End Get
        End Property

    End Structure

End Class

```

```

        End Get
    End Property

    Public ReadOnly Property HighestIndex() As Integer
        Get
            Return highestIndexValue
        End Get
    End Property

    Public Shared Function MapToLowerBoundary( _
        ByVal rowIndex As Integer) As Integer

        ' Return the lowest index of a page containing the given index.
        Return (rowIndex \ RowsPerPage) * RowsPerPage

    End Function

    Private Shared Function MapToUpperBoundary( _
        ByVal rowIndex As Integer) As Integer

        ' Return the highest index of a page containing the given index.
        Return MapToLowerBoundary(rowIndex) + RowsPerPage - 1

    End Function

End Structure

Private cachePages As DataPage()
Private dataSupply As IDataPageRetriever

Public Sub New(ByVal dataSupplier As IDataPageRetriever, _
    ByVal rowsPerPage As Integer)

    dataSupply = dataSupplier
    Cache.RowsPerPage = rowsPerPage
    LoadFirstTwoPages()

End Sub

' Sets the value of the element parameter if the value is in the cache.
Private Function IfPageCached_ThenSetElement(ByVal rowIndex As Integer, _
    ByVal columnIndex As Integer, ByRef element As String) As Boolean

    If IsRowCachedInPage(0, rowIndex) Then
        element = cachePages(0).table.Rows(rowIndex Mod RowsPerPage) _
            .Item(columnIndex).ToString()
        Return True
    ElseIf IsRowCachedInPage(1, rowIndex) Then
        element = cachePages(1).table.Rows(rowIndex Mod RowsPerPage) _
            .Item(columnIndex).ToString()
        Return True
    End If

    Return False
End Function

Public Function RetrieveElement(ByVal rowIndex As Integer, _
    ByVal columnIndex As Integer) As String

    Dim element As String = Nothing
    If IfPageCached_ThenSetElement(rowIndex, columnIndex, element) Then
        Return element
    Else
        Return RetrieveData_CacheIt_ThenReturnElement( _
            rowIndex, columnIndex)
    End If
End Function

```

```

Private Sub LoadFirstTwoPages()

    cachePages = New DataPage() { _
        New DataPage(dataSupply.SupplyPageOfData( _
            DataPage.MapToLowerBoundary(0), RowsPerPage), 0), _
        New DataPage(dataSupply.SupplyPageOfData( _
            DataPage.MapToLowerBoundary(RowsPerPage), _ 
            RowsPerPage), RowsPerPage) _
    }

End Sub

Private Function RetrieveData_CacheIt_ThenReturnElement( _
    ByVal rowIndex As Integer, ByVal columnIndex As Integer) As String

    ' Retrieve a page worth of data containing the requested value.
    Dim table As DataTable = dataSupply.SupplyPageOfData( _
        DataPage.MapToLowerBoundary(rowIndex), RowsPerPage)

    ' Replace the cached page furthest from the requested cell
    ' with a new page containing the newly retrieved data.
    cachePages(GetIndexToUnusedPage(rowIndex)) = _
        New DataPage(table, rowIndex)

    Return RetrieveElement(rowIndex, columnIndex)

End Function

' Returns the index of the cached page most distant from the given index
' and therefore least likely to be reused.
Private Function GetIndexToUnusedPage(ByVal rowIndex As Integer) _
    As Integer

    If rowIndex > cachePages(0).HighestIndex AndAlso _
        rowIndex > cachePages(1).HighestIndex Then

        Dim offsetFromPage0 As Integer = _
            rowIndex - cachePages(0).HighestIndex
        Dim offsetFromPage1 As Integer = _
            rowIndex - cachePages(1).HighestIndex
        If offsetFromPage0 < offsetFromPage1 Then
            Return 1
        End If
        Return 0
    Else
        Dim offsetFromPage0 As Integer = _
            cachePages(0).LowestIndex - rowIndex
        Dim offsetFromPage1 As Integer = _
            cachePages(1).LowestIndex - rowIndex
        If offsetFromPage0 < offsetFromPage1 Then
            Return 1
        End If
        Return 0
    End If

End Function

' Returns a value indicating whether the given row index is contained
' in the given DataPage.
Private Function IsRowCachedInPage( _
    ByVal pageNumber As Integer, ByVal rowIndex As Integer) As Boolean

    Return rowIndex <= cachePages(pageNumber).HighestIndex AndAlso _
        rowIndex >= cachePages(pageNumber).LowestIndex

End Function

End Class

```

Compiling the Code

This example requires:

- References to the System, System.Data, System.Xml, and System.Windows.Forms assemblies.
- Access to a server with the Northwind SQL Server sample database installed.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

.NET Framework Security

Storing sensitive information, such as a password, within the connection string can affect the security of your application. Using Windows Authentication (also known as integrated security) is a more secure way to control access to a database. For more information, see [Protecting Connection Information](#).

See Also

[DataGridView](#)

[VirtualMode](#)

[CellValueNeeded](#)

[Implementing Virtual Mode with Just-In-Time Data Loading in the Windows Forms DataGridView Control](#)

[Performance Tuning in the Windows Forms DataGridView Control](#)

[Virtual Mode in the Windows Forms DataGridView Control](#)

Default keyboard and mouse handling in the Windows Forms DataGridView control

5/4/2018 • 8 min to read • [Edit Online](#)

The following tables describe how users can interact with the [DataGridView](#) control through a keyboard and a mouse.

NOTE

To customize keyboard behavior, you can handle standard keyboard events such as [KeyDown](#). In edit mode, however, the hosted editing control receives the keyboard input and the keyboard events do not occur for the [DataGridView](#) control. To handle editing control events, attach your handlers to the editing control in an [EditingControlShowing](#) event handler. Alternatively, you can customize keyboard behavior in a [DataGridView](#) subclass by overriding the [ProcessDialogKey](#) and [ProcessDataGridViewKey](#) methods.

Default keyboard handling

Basic navigation and entry keys

KEY OR KEY COMBINATION	DESCRIPTION
DOWN ARROW	Moves the focus to the cell directly below the current cell. If the focus is in the last row, does nothing.
LEFT ARROW	Moves the focus to the previous cell in the row. If the focus is in the first cell in the row, does nothing.
RIGHT ARROW	Moves the focus to the next cell in the row. If the focus is in the last cell in the row, does nothing.
UP ARROW	Moves the focus to the cell directly above the current cell. If the focus is in the first row, does nothing.
HOME	Moves the focus to the first cell in the current row.
END	Moves the focus to the last cell in the current row.
PAGE DOWN	Scrolls the control downward by the number of rows that are fully displayed. Moves the focus to the last fully displayed row without changing columns.
PAGE UP	Scrolls the control upward by the number of rows that are fully displayed. Moves focus to the first displayed row without changing columns.

KEY OR KEY COMBINATION	DESCRIPTION
TAB	<p>If the StandardTab property value is <code>false</code>, moves the focus to the next cell in the current row. If the focus is already in the last cell of the row, moves the focus to the first cell in the next row. If the focus is in the last cell in the control, moves the focus to the next control in the tab order of the parent container.</p> <p>If the StandardTab property value is <code>true</code>, moves the focus to the next control in the tab order of the parent container.</p>
SHIFT+TAB	<p>If the StandardTab property value is <code>false</code>, moves the focus to the previous cell in the current row. If the focus is already in the first cell of the row, moves the focus to the last cell in the previous row. If the focus is in the first cell in the control, moves the focus to the previous control in the tab order of the parent container.</p> <p>If the StandardTab property value is <code>true</code>, moves the focus to the previous control in the tab order of the parent container.</p>
CTRL+TAB	<p>If the StandardTab property value is <code>false</code>, moves the focus to the next control in the tab order of the parent container.</p> <p>If the StandardTab property value is <code>true</code>, moves the focus to the next cell in the current row. If the focus is already in the last cell of the row, moves the focus to the first cell in the next row. If the focus is in the last cell in the control, moves the focus to the next control in the tab order of the parent container.</p>
CTRL+SHIFT+TAB	<p>If the StandardTab property value is <code>false</code>, moves the focus to the previous control in the tab order of the parent container.</p> <p>If the StandardTab property value is <code>true</code>, moves the focus to the previous cell in the current row. If the focus is already in the first cell of the row, moves the focus to the last cell in the previous row. If the focus is in the first cell in the control, moves the focus to the previous control in the tab order of the parent container.</p>
CTRL+ARROW	Moves the focus to the farthest cell in the direction of the arrow.
CTRL+HOME	Moves the focus to the first cell in the control.
CTRL+END	Moves the focus to the last cell in the control.
CTRL+PAGE DOWN/UP	Same as PAGE DOWN or PAGE UP.
F2	Puts the current cell into cell edit mode if the EditMode property value is <code>EditOnF2</code> or <code>EditOnKeystrokeOrF2</code> .

KEY OR KEY COMBINATION	DESCRIPTION
F3	Sorts the current column if the DataGridViewColumn.SortMode property value is Automatic . It's the same as clicking the current column header. Available since .NET Framework 4.7.2. To enable this feature, applications must target .NET Framework 4.7.2 or later versions or explicitly opt into accessibility improvements using AppContext switches.
F4	If the current cell is a DataGridViewComboBoxCell , puts the cell into edit mode and displays the drop-down list.
ALT+UP/DOWN ARROW	If the current cell is a DataGridViewComboBoxCell , puts the cell into edit mode and displays the drop-down list.
SPACE	If the current cell is a DataGridViewButtonCell , DataGridViewLinkCell , or DataGridViewCheckBoxCell , raises the CellClick and CellContentClick events. If the current cell is a DataGridViewButtonCell , also presses the button. If the current cell is a DataGridViewCheckBoxCell , also changes the check state.
ENTER	Commits any changes to the current cell and row and moves the focus to the cell directly below the current cell. If the focus is in the last row, commits any changes without moving the focus.
ESC	If the control is in edit mode, cancels the edit. If the control is not in edit mode, reverts any changes that have been made to the current row if the control is bound to a data source that supports editing or virtual mode has been implemented with row-level commit scope.
BACKSPACE	Deletes the character before the insertion point when editing a cell.
DELETE	Deletes the character after the insertion point when editing a cell.
CTRL+ENTER	Commits any changes to the current cell without moving the focus. Also commits any changes to the current row if the control is bound to a data source that supports editing or virtual mode has been implemented with row-level commit scope.
CTRL+0	Enters a DBNull.Value value into the current cell if the cell can be edited. By default, the display value for a DBNull cell value is the value of the NullValue property of the DataGridViewCellStyle in effect for the current cell.

Selection keys

If the [MultiSelect](#) property is set to `false` and the [SelectionMode](#) property is set to [CellSelect](#), changing the current cell by using the navigation keys changes the selection to the new cell. The SHIFT, CTRL, and ALT keys do not affect this behavior.

If the [SelectionMode](#) is set to [RowHeaderSelect](#) or [ColumnHeaderSelect](#), the same behavior occurs but with the following additions.

KEY OR KEY COMBINATION	DESCRIPTION
SHIFT+SPACEBAR	Selects the full row or column (the same as clicking the row or column header).
navigation key (arrow key, PAGE UP/DOWN, HOME, END)	If a full row or column is selected, changing the current cell to a new row or column moves the selection to the full new row or column (depending on the selection mode).

If `MultiSelect` is set to `false` and `SelectionMode` is set to `FullRowSelect` or `FullColumnSelect`, changing the current cell to a new row or column by using the keyboard moves the selection to the full new row or column. The SHIFT, CTRL, and ALT keys do not affect this behavior.

If `MultiSelect` is set to `true`, the navigation behavior does not change, but navigating with the keyboard while pressing SHIFT (including CTRL+SHIFT) will modify a multi-cell selection. Before navigation begins, the control marks the current cell as an anchor cell. When you navigate while pressing SHIFT, the selection includes all cells between the anchor cell and the current cell. Other cells in the control will remain selected if they were already selected, but they may become unselected if the keyboard navigation temporarily puts them between the anchor cell and the current cell.

If `MultiSelect` is set to `true` and `SelectionMode` is set to `FullRowSelect` or `FullColumnSelect`, the behavior of the anchor cell and current cell is the same, but only full rows or columns become selected or unselected.

Default mouse handling

Basic mouse handling

NOTE

Clicking a cell with the left mouse button always changes the current cell. Clicking a cell with the right mouse button opens a shortcut menu, when one is available.

MOUSE ACTION	DESCRIPTION
Left mouse button down	Makes the clicked cell the current cell, and raises the <code>DataGridView.CellMouseDown</code> event.
Left mouse button up	Raises the <code>DataGridView.CellMouseUp</code> event
Left mouse button click	Raises the <code>DataGridView.CellClick</code> and <code>DataGridView.CellMouseClick</code> events
Left mouse button down, and drag on a column header cell	If the <code>DataGridView.AllowUserToOrderColumns</code> property is <code>true</code> , moves the column so that it can be dropped into a new position.

Mouse selection

No selection behavior is associated with the middle mouse button or the mouse wheel.

If the `MultiSelect` property is set to `false` and the `SelectionMode` property is set to `CellSelect`, the following behavior occurs.

MOUSE ACTION	DESCRIPTION
Click left mouse button	Selects only the current cell if the user clicks a cell. No selection behavior if the user clicks a row or column header.
Click right mouse button	Displays a shortcut menu if one is available.

The same behavior occurs when the [SelectionMode](#) is set to [RowHeaderSelect](#) or [ColumnHeaderSelect](#), except that, depending on the selection mode, clicking a row or column header will select the full row or column and set the current cell to the first cell in the row or column.

If [SelectionMode](#) is set to [FullRowSelect](#) or [FullColumnSelect](#), clicking any cell in a row or column will select the full row or column.

If [MultiSelect](#) is set to `true`, clicking a cell while pressing CTRL or SHIFT will modify a multi-cell selection.

When you click a cell while pressing CTRL, the cell will change its selection state while all other cells retain their current selection state.

When you click a cell or a series of cells while pressing SHIFT, the selection includes all cells between the current cell and an anchor cell located at the position of the current cell before the first click. When you click and drag the pointer across multiple cells, the anchor cell is the cell clicked at the beginning of the drag operation. Subsequent clicks while pressing SHIFT change the current cell, but not the anchor cell. Other cells in the control will remain selected if they were already selected, but they may become unselected if mouse navigation temporarily puts them between the anchor cell and the current cell.

If [MultiSelect](#) is set to `true` and [SelectionMode](#) is set to [RowHeaderSelect](#) or [ColumnHeaderSelect](#), clicking a row or column header (depending on the selection mode) while pressing SHIFT will modify an existing selection of full rows or columns if such a selection exists. Otherwise, it will clear the selection and start a new selection of full rows or columns. Clicking a row or column header while pressing CTRL, however, will add or remove the clicked row or column from the current selection without otherwise modifying the current selection.

If [MultiSelect](#) is set to `true` and [SelectionMode](#) is set to [FullRowSelect](#) or [FullColumnSelect](#), clicking a cell while pressing SHIFT or CTRL behaves the same way except that only full rows and columns are affected.

See also

[DataGridView](#)

[DataGridView Control](#)

Differences Between the Windows Forms DataGridView and DataGrid Controls

5/4/2018 • 3 min to read • [Edit Online](#)

The [DataGridView](#) control is a new control that replaces the [DataGrid](#) control. The [DataGridView](#) control provides numerous basic and advanced features that are missing in the [DataGrid](#) control. Additionally, the architecture of the [DataGridView](#) control makes it much easier to extend and customize than the [DataGrid](#) control.

The following table describes a few of the primary features available in the [DataGridView](#) control that are missing from the [DataGrid](#) control.

DataGridView Control Feature	Description
Multiple column types	The DataGridView control provides more built-in column types than the DataGrid control. These column types meet the needs of most common scenarios, but are also easier to extend or replace than the column types in the DataGrid control. For more information, see Column Types in the Windows Forms DataGridView Control .
Multiple ways to display data	The DataGrid control is limited to displaying data from an external data source. The DataGridView control, however, can display unbound data stored in the control, data from a bound data source, or bound and unbound data together. You can also implement virtual mode in the DataGridView control to provide custom data management. For more information, see Data Display Modes in the Windows Forms DataGridView Control .
Multiple ways to customize the display of data	The DataGridView control provides many properties and events that enable you to specify how data is formatted and displayed. For example, you can change the appearance of cells, rows, and columns depending on the data they contain, or you can replace data of one data type with equivalent data of another type. For more information, see Data Formatting in the Windows Forms DataGridView Control .
Multiple options for changing cell, row, column, and header appearance and behavior	The DataGridView control enables you to work with individual grid components in numerous ways. For example, you can freeze rows and columns to prevent them from scrolling; hide rows, columns, and headers; change the way row, column, and header sizes are adjusted; change the way users make selections; and provide ToolTips and shortcut menus for individual cells, rows, and columns.

The [DataGrid](#) control is retained for backward compatibility and for special needs. For nearly all purposes, you should use the [DataGridView](#) control. The only feature that is available in the [DataGrid](#) control that is not available in the [DataGridView](#) control is the hierarchical display of information from two related tables in a single control. You must use two [DataGridView](#) controls to display information from two tables that are in a master/detail relationship.

Upgrading to the DataGridView Control

If you have existing applications that use the [DataGrid](#) control in a simple data-bound scenario without customizations, you can simply replace the old control with the new control. Both controls use the standard Windows Forms data-binding architecture, so the [DataGridView](#) control will display your bound data with no additional configuration needed. You might want to consider taking advantage of data-binding improvements, however, by binding your data to a [BindingSource](#) component, which you can then bind to the [DataGridView](#) control. For more information, see [BindingSource Component](#).

Because the [DataGridView](#) control has an entirely new architecture, there is no straightforward conversion path that will enable you to use [DataGrid](#) customizations with the [DataGridView](#) control. Many [DataGrid](#) customizations are unnecessary with the [DataGridView](#) control, however, because of the built-in features available in the new control. If you have created custom column types for the [DataGrid](#) control that you want to use with the [DataGridView](#) control, you will have to implement them again using the new architecture. For more information, see [Customizing the Windows Forms DataGridView Control](#).

See Also

- [DataGridView](#)
- [DataGrid](#)
- [BindingSource](#)
- [DataGridView Control](#)
- [DataGrid Control](#)
- [BindingSource Component](#)
- [Column Types in the Windows Forms DataGridView Control](#)
- [Cell Styles in the Windows Forms DataGridView Control](#)
- [Data Display Modes in the Windows Forms DataGridView Control](#)
- [Data Formatting in the Windows Forms DataGridView Control](#)
- [Sizing Options in the Windows Forms DataGridView Control](#)
- [Column Sort Modes in the Windows Forms DataGridView Control](#)
- [Selection Modes in the Windows Forms DataGridView Control](#)
- [Customizing the Windows Forms DataGridView Control](#)

DateTimePicker Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `DateTimePicker` control allows the user to select a single item from a list of dates or times. When used to represent a date, it appears in two parts: a drop-down list with a date represented in text, and a grid that appears when you click on the down-arrow next to the list.

In This Section

[DateTimePicker Control Overview](#)

Introduces the general concepts of the `DateTimePicker` control, which allows users to select a single item from a list of dates or times.

[How to: Display a Date in a Custom Format with the Windows Forms DateTimePicker Control](#)

Explains how to use format strings to display dates in a preferred format.

[How to: Set and Return Dates with the Windows Forms DateTimePicker Control](#)

Provides steps to set the date in the control and to access the date the user has selected.

[How to: Display Time with the DateTimePicker Control](#)

Shows steps to for a `DateTimePicker` to display times only.

Reference

[DateTimePicker](#)

Describes this class and has links to all its members.

[MonthCalendar](#)

Describes this class and has links to all its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

[MonthCalendar Control](#)

Presents an intuitive graphical interface for users to view and set date information.

DateTimePicker Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [DateTimePicker](#) control allows the user to select a single item from a list of dates or times. When used to represent a date, it appears in two parts: a drop-down list with a date represented in text, and a grid that appears when you click on the down-arrow next to the list. The grid looks like the [MonthCalendar](#) control, which can be used for selecting multiple dates. For more information on the [MonthCalendar](#) control, see [MonthCalendar Control Overview](#).

Key Properties

If you wish the [DateTimePicker](#) to appear as a control for picking or editing times instead of dates, set the [ShowUpDown](#) property to `true` and the [Format](#) property to [Time](#). For more information see [How to: Display Time with the DateTimePicker Control](#).

When the [ShowCheckBox](#) property is set to `true`, a check box is displayed next to the selected date in the control. When the check box is checked, the selected date-time value can be updated. When the check box is empty, the value appears unavailable.

The control's [MaxDate](#) and [MinDate](#) properties determine the range of dates and times. The [Value](#) property contains the current date and time the control is set to. For details, see [How to: Set and Return Dates with the Windows Forms DateTimePicker Control](#). The values can be displayed in four formats, which are set by the [Format](#) property: [Long](#), [Short](#), [Time](#), or [Custom](#). If a custom format is selected, you must set the [CustomFormat](#) property to an appropriate string. For details, see [How to: Display a Date in a Custom Format with the Windows Forms DateTimePicker Control](#).

See Also

[How to: Display a Date in a Custom Format with the Windows Forms DateTimePicker Control](#)

[How to: Set and Return Dates with the Windows Forms DateTimePicker Control](#)

How to: Display a Date in a Custom Format with the Windows Forms DateTimePicker Control

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [DateTimePicker](#) control gives you flexibility in formatting the display of dates and times in the control. The [Format](#) property allows you to select from predefined formats, listed in the [DateTimePickerFormat](#). If none of these is adequate for your purposes, you can create your own format style using format characters listed in [CustomFormat](#).

To display a custom format

1. Set the [Format](#) property to `DateTimePickerFormat.Custom`.
2. Set the [CustomFormat](#) property to a format string.

```
DateTimePicker1.Format = DateTimePickerFormat.Custom  
' Display the date as "Mon 27 Feb 2012".  
DateTimePicker1.CustomFormat = "ddd dd MMM yyyy"
```

```
dateTimePicker1.Format = DateTimePickerFormat.Custom;  
// Display the date as "Mon 27 Feb 2012".  
dateTimePicker1.CustomFormat = "ddd dd MMM yyyy";
```

```
dateTimePicker1->Format = DateTimePickerFormat::Custom;  
// Display the date as "Mon 27 Feb 2012".  
dateTimePicker1->CustomFormat = "ddd dd MMM yyyy";
```

To add text to the formatted value

1. Use single quotation marks to enclose any character that is not a format character like "M" or a delimiter like ":". For example, the format string below displays the current date with the format "Today is: 05:30:31 Friday March 02, 2012" in the English (United States) culture.

```
DateTimePicker1.CustomFormat = "'Today is:' hh:mm:ss dddd MMMM dd, yyyy"
```

```
dateTimePicker1.CustomFormat = "'Today is:' hh:mm:ss dddd MMMM dd, yyyy";
```

```
dateTimePicker1->CustomFormat =  
"'Today is:' hh:mm:ss dddd MMMM dd, yyyy";
```

Depending on the culture setting, any characters not enclosed in single quotation marks may be changed. For example, the format string above displays the current date with the format "Today is: 05:30:31 Friday March 02, 2012" in the English (United States) culture. Note that the first colon is enclosed in single quotation marks, because it is not intended to be a delimiting character as it is in "hh:mm:ss". In another culture, the format might appear as "Today is: 05.30.31 Friday March 02, 2012".

See Also

[DateTimePicker Control](#)

[How to: Set and Return Dates with the Windows Forms DateTimePicker Control](#)

How to: Set and Return Dates with the Windows Forms DateTimePicker Control

5/4/2018 • 1 min to read • [Edit Online](#)

The currently selected date or time in the Windows Forms [DateTimePicker](#) control is determined by the [Value](#) property. You can set the [Value](#) property before the control is displayed (for example, at design time or in the form's [Load](#) event) to determine which date will be initially selected in the control. By default, the control's [Value](#) is set to the current date. If you change the control's [Value](#) in code, the control is automatically updated on the form to reflect the new setting.

The [Value](#) property returns a [DateTime](#) structure as its value. There are several properties of the [DateTime](#) structure that return specific information about the displayed date. These properties can only be used to return a value; do not use them to set a value.

- For date values, the [Month](#), [Day](#), and [Year](#) properties return integer values for those time units of the selected date. The [DayOfWeek](#) property returns a value indicating the selected day of the week (possible values are listed in the [DayOfWeek](#) enumeration).
- For time values, the [Hour](#), [Minute](#), [Second](#), and [Millisecond](#) properties return integer values for those time units. To configure the control to display times, see [How to: Display Time with the DateTimePicker Control](#).

To set the date and time value of the control

- Set the [Value](#) property to a date or time value.

```
DateTimePicker1.Value = New DateTime(2001, 10, 20)
```

```
dateTimePicker1.Value = new DateTime(2001, 10, 20);
```

```
dateTimePicker1->Value = DateTime(2001, 10, 20);
```

To return the date and time value

- Call the [Text](#) property to return the entire value as formatted in the control, or call the appropriate method of the [Value](#) property to return a part of the value. Use [ToString](#) to convert the information into a string that can be displayed to the user.

```
MessageBox.Show("The selected value is ", DateTimePicker1.Text)
MessageBox.Show("The day of the week is ",
    DateTimePicker1.Value.DayOfWeek.ToString())
MessageBox.Show("Millisecond is: ",
    DateTimePicker1.Value.Millisecond.ToString())
```

```
MessageBox.Show ("The selected value is " +
    dateTimePicker1.Text);
MessageBox.Show ("The day of the week is " +
    dateTimePicker1.Value.DayOfWeek.ToString());
MessageBox.Show("Millisecond is: " +
    dateTimePicker1.Value.Millisecond.ToString());
```

```
MessageBox::Show (String::Concat("The selected value is ",  
    dateTimePicker1->Text));  
MessageBox::Show (String::Concat("The day of the week is ",  
    dateTimePicker1->Value.DayOfWeek.ToString()));  
MessageBox::Show(String::Concat("Millisecond is: ",  
    dateTimePicker1->Value.Millisecond.ToString()));
```

See Also

[DateTimePicker Control](#)

[How to: Display a Date in a Custom Format with the Windows Forms DateTimePicker Control](#)

How to: Display Time with the DateTimePicker Control

5/4/2018 • 1 min to read • [Edit Online](#)

If you want your application to enable users to select a date and time, and to display that date and time in the specified format, use the [DateTimePicker](#) control. The following procedure shows how to use the [DateTimePicker](#) control to display the time.

To display the time with the DateTimePicker control

1. Set the [Format](#) property to [Time](#)

```
timePicker.Format = DateTimePickerFormat.Time;
```

```
timePicker.Format = DateTimePickerFormat.Time
```

2. Set the [ShowUpDown](#) property for the [DateTimePicker](#) to [true](#).

```
timePicker.ShowUpDown = true;
```

```
timePicker.ShowUpDown = True
```

Example

The following code sample shows how to create a [DateTimePicker](#) that enables users to choose a time only.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace TimePickerApplication
{
    public class Form1 : Form
    {
        public Form1()
        {
            InitializeTimePicker();
        }
        private DateTimePicker timePicker;

        private void InitializeTimePicker()
        {
            timePicker = new DateTimePicker();
            timePicker.Format = DateTimePickerFormat.Time;
            timePicker.ShowUpDown = true;
            timePicker.Location = new Point(10, 10);
            timePicker.Width = 100;
            Controls.Add(timePicker);
        }
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.Run(new Form1());
        }
    }
}
```

```

Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Drawing
Imports System.Text
Imports System.Windows.Forms

Public Class Form1
    Inherits Form

    Public Sub New()
        InitializeTimePicker()

    End Sub
    Private timePicker As DateTimePicker

    Private Sub InitializeTimePicker()
        timePicker = New DateTimePicker()
        timePicker.Format = DateTimePickerFormat.Time
        timePicker.ShowUpDown = True
        timePicker.Location = New Point(10, 10)
        timePicker.Width = 100
        Controls.Add(timePicker)
    End Sub

    <STAThread()> _
    Shared Sub Main()
        Application.EnableVisualStyles()
        Application.Run(New Form1())
    End Sub
End Class

```

Compiling the Code

This example requires:

- References to the System, System.Data, System.Drawing and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[DateTimePicker Control](#)

Dialog-Box Controls and Components (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The following Windows Forms controls and components present standard dialog boxes. Follow the links for more information about the functions available in each dialog box.

Reference

[ColorDialog](#)

Provides reference information about the [ColorDialog](#) class and its members.

[FontDialog](#)

Provides reference information about the [FontDialog](#) class and its members.

[OpenFileDialog](#)

Provides reference information about the [OpenFileDialog](#) class and its members.

[PageSetupDialog](#)

Provides reference information about the [PageSetupDialog](#) class and its members.

[PrintDialog](#)

Provides reference information about the [PrintDialog](#) class and its members.

[PrintPreviewDialog](#)

Provides reference information about the [PrintPreviewDialog](#) class and its members.

[SaveFileDialog](#)

Provides reference information about the [SaveFileDialog](#) class and its members.

Related Sections

[Dialog Boxes in Windows Forms](#)

Describes how to create a dialog box for a Windows Form.

[ColorDialog Component Overview](#)

Enables the user to select a color from a palette in a pre-configured dialog box and to add custom colors to that palette.

[FolderBrowserDialog Component Overview \(Windows Forms\)](#)

Enables users to browse and select folders.

[FontDialog Component Overview](#)

Exposees the fonts that are currently installed on the system.

[OpenFileDialog Component Overview](#)

Allows users to open files via a pre-configured dialog box.

[PageSetupDialog Component Overview](#)

Sets page details for printing via a pre-configured dialog box.

[PrintDialog Component Overview](#)

Selects a printer, chooses the pages to print, and determines other print-related settings.

[PrintPreviewDialog Control Overview](#)

Displays a document as it will appear when it is printed.

[SaveFileDialog Component Overview](#)

Selects files to save and where to save them.

Also see [Dialog Boxes in Windows Forms](#).

DomainUpDown Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [DomainUpDown](#) control looks like a combination of a text box and a pair of buttons for moving up or down through a list. The control displays and sets a text string from a list of choices. The user can select the string by clicking up and down buttons to move through a list, by pressing the UP and DOWN ARROW keys, or by typing a string that matches an item in the list. One possible use for this control is for selecting items from an alphabetically sorted list of names. (To sort the list, set the [Sorted](#) property to `true`.) The function of this control is very similar to the list box or combo box, but it takes up very little space.

The key properties of the control are [Items](#), [ReadOnly](#), and [Wrap](#). The [Items](#) property contains the list of objects whose text values are displayed in the control. If [ReadOnly](#) is set to `false`, the control automatically completes text that the user types and matches it to a value in the list. If [Wrap](#) is set to `true`, scrolling past the last item will take you to the first item in the list and vice versa. The key methods of the control are [UpButton](#) and [DownButton](#).

This control displays only text strings. If you want a control that displays numeric values, use the [NumericUpDown](#) control. For more information, see [NumericUpDown Control](#).

In This Section

[DomainUpDown Control Overview](#)

Introduces the general concepts of the [DomainUpDown](#) control, which allows users to browse through and select from a list of text strings.

[How to: Add Items to Windows Forms DomainUpDown Controls Programmatically](#)

Describes how to specify the text strings the [DomainUpDown](#) control should display.

[How to: Remove Items from Windows Forms DomainUpDown Controls](#)

Describes how to delete items from the [DomainUpDown](#) control in code.

Reference

[DomainUpDown](#)

Describes this class and has links to all its members.

[NumericUpDown](#)

Describes this class and has links to all its members..

Related Sections

[Controls You Can Use On Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

DomainUpDown Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [DomainUpDown](#) control is essentially a combination of a text box and a pair of buttons for moving up or down through a list. The control displays and sets a text string from a list of choices. The user can select the string by clicking up and down buttons to move through a list, by pressing the UP and DOWN ARROW keys, or by typing a string that matches an item in the list. One possible use for this control is for selecting items from an alphabetically sorted list of names.

NOTE

To sort the list, set the [Sorted](#) property to `true`.

The function of this control is very similar to the list box or combo box, but it takes up very little space.

Key Properties

The key properties of the control are [Items](#), [ReadOnly](#), and [Wrap](#). The [Items](#) property contains the list of objects whose text values are displayed in the control. If [ReadOnly](#) is set to `false`, the control automatically completes text that the user types and matches it to a value in the list. If [Wrap](#) is set to `true`, scrolling past the last item will take you to the first item in the list and vice versa. The key methods of the control are [UpButton](#) and [DownButton](#).

This control displays only text strings. If you want a control that displays numeric values, use the [NumericUpDown](#) control. For more information, see [NumericUpDown Control Overview](#).

See Also

[DomainUpDown](#)

[DomainUpDown Control](#)

How to: Add Items to Windows Forms DomainUpDown Controls Programmatically

5/4/2018 • 1 min to read • [Edit Online](#)

You can add items to the Windows Forms [DomainUpDown](#) control in code. Call the [Add](#) or [Insert](#) method of the [DomainUpDown.DomainUpDownItemCollection](#) class to add items to the control's [Items](#) property. The [Add](#) method adds an item to the end of a collection, while the [Insert](#) method adds an item at a specified position.

To add a new item

1. Use the [Add](#) method to add an item to the end of the list of items.

```
DomainUpDown1.Items.Add("noodles")
```

```
domainUpDown1.Items.Add("noodles");
```

```
domainUpDown1->Items->Add("noodles");
```

-or-

2. Use the [Insert](#) method to insert an item into the list at a specified position.

```
' Inserts an item at the third position in the list
DomainUpDown1.Items.Insert(2, "rice")
```

```
// Inserts an item at the third position in the list
domainUpDown1.Items.Insert(2, "rice");
```

```
// Inserts an item at the third position in the list
domainUpDown1->Items->Insert(2, "rice");
```

See Also

[DomainUpDown](#)

[DomainUpDown.DomainUpDownItemCollection.Add](#)

[ArrayList.Insert](#)

[DomainUpDown Control](#)

[DomainUpDown Control Overview](#)

How to: Remove Items from Windows Forms DomainUpDown Controls

5/4/2018 • 1 min to read • [Edit Online](#)

You can remove items from the Windows Forms [DomainUpDown](#) control by calling the [Remove](#) or [RemoveAt](#) method of the [DomainUpDown.DomainUpDownItemCollection](#) class. The [Remove](#) method removes a specific item, while the [RemoveAt](#) method removes an item by its position.

To remove an item

- Use the [Remove](#) method of the [DomainUpDown.DomainUpDownItemCollection](#) class to remove an item by name.

```
DomainUpDown1.Items.Remove("noodles")
```

```
domainUpDown1.Items.Remove("noodles");
```

```
domainUpDown1->Items->Remove("noodles");
```

-or-

- Use the [RemoveAt](#) method to remove an item by its position.

```
' Removes the first item in the list.  
DomainUpDown1.Items.RemoveAt(0)
```

```
// Removes the first item in the list.  
domainUpDown1.Items.RemoveAt(0);
```

```
// Removes the first item in the list.  
domainUpDown1->Items->RemoveAt(0);
```

See Also

[DomainUpDown](#)

[DomainUpDown.DomainUpDownItemCollection.Remove](#)

[DomainUpDown.DomainUpDownItemCollection.RemoveAt](#)

[DomainUpDown Control](#)

[DomainUpDown Control Overview](#)

ErrorProvider Component (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `ErrorProvider` component is used to show the user in a non-intrusive way that something is wrong. It is typically used in conjunction with validating user input on a form, or displaying errors within a dataset.

In This Section

[ErrorProvider Component Overview](#)

Explains what this component is and its key features and properties.

[How to: Display Error Icons for Form Validation with the Windows Forms ErrorProvider Component](#)

Gives directions for validating user input with an error provider component.

[How to: View Errors Within a DataSet with the Windows Forms ErrorProvider Component](#)

Gives directions for using an error provider component to display data errors.

Reference

[ErrorProvider](#)

Describes this class and has links to all its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

ErrorProvider Component Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [ErrorProvider](#) component is used to validate user input on a form or control. It is typically used in conjunction with validating user input on a form, or displaying errors within a dataset. An error provider is a better alternative than displaying an error message in a message box, because once a message box is dismissed, the error message is no longer visible. The [ErrorProvider](#) component displays an error icon (感叹号) next to the relevant control, such as a text box; when the user positions the mouse pointer over the error icon, a ToolTip appears, showing the error message string.

Key Properties

The [ErrorProvider](#) component's key properties are [DataSource](#), [ContainerControl](#), and [Icon](#). When using [ErrorProvider](#) component with data-bound controls, the [ContainerControl](#) property must be set to the appropriate container (usually the Windows Form) in order for the component to display an error icon on the form. When the component is added in the designer, the [ContainerControl](#) property is set to the containing form; if you add the control in code, you must set it yourself.

The [Icon](#) property can be set to a custom error icon instead of the default. When the [DataSource](#) property is set, the [ErrorProvider](#) component can display error messages for a dataset. The key method of the [ErrorProvider](#) component is the [SetError](#) method, which specifies the error message string and where the error icon should appear.

NOTE

The [ErrorProvider](#) component does not provide built-in support for accessibility clients. To make your application accessible when using this component, you must provide an additional, accessible feedback mechanism.

See Also

[ErrorProvider](#)

[How to: View Errors Within a DataSet with the Windows Forms ErrorProvider Component](#)

[How to: Display Error Icons for Form Validation with the Windows Forms ErrorProvider Component](#)

How to: Display Error Icons for Form Validation with the Windows Forms ErrorProvider Component

5/4/2018 • 1 min to read • [Edit Online](#)

You can use a Windows Forms [ErrorProvider](#) component to display an error icon when the user enters invalid data. You must have at least two controls on the form in order to tab between them and thereby invoke the validation code.

To display an error icon when a control's value is invalid

1. Add two controls — for example, text boxes — to a Windows Form.
2. Add an [ErrorProvider](#) component to the form.
3. Select the first control and add code to its [Validating](#) event handler. In order for this code to run properly, the procedure must be connected to the event. For more information, see [How to: Create Event Handlers at Run Time for Windows Forms](#).

The following code tests the validity of the data the user has entered; if the data is invalid, the [SetError](#) method is called. The first argument of the [SetError](#) method specifies which control to display the icon next to. The second argument is the error text to display.

```
Private Sub TextBox1_Validating(ByVal Sender As Object, _
    ByVal e As System.ComponentModel.CancelEventArgs) Handles _
    TextBox1.Validating
    If Not IsNumeric(TextBox1.Text) Then
        ErrorProvider1.SetError(TextBox1, "Not a numeric value.")
    Else
        ' Clear the error.
        ErrorProvider1.SetError(TextBox1, "")
    End If
End Sub
```

```
protected void textBox1_Validating (object sender,
    System.ComponentModel.CancelEventArgs e)
{
    try
    {
        int x = Int32.Parse(textBox1.Text);
        errorProvider1.SetError(textBox1, "");
    }
    catch (Exception ex)
    {
        errorProvider1.SetError(textBox1, "Not an integer value.");
    }
}
```

```
private:  
    System::Void textBox1_Validate(System::Object ^ sender,  
        System::ComponentModel::CancelEventArgs ^ e)  
{  
    try  
    {  
        int x = Int32::Parse(textBox1->Text);  
        errorProvider1->SetError(textBox1, "");  
    }  
    catch (System::Exception ^ ex)  
    {  
        errorProvider1->SetError(textBox1, "Not an integer value.");  
    }  
}
```

(Visual C#, Visual C++) Place the following code in the form's constructor to register the event handler.

```
this.textBox1.Validating += new  
System.ComponentModel.CancelEventHandler(this.textBox1_Validating);
```

```
this->textBox1->Validating += gcnew  
System::ComponentModel::CancelEventHandler  
(this, &Form1::textBox1_Validating);
```

4. Run the project. Type invalid (in this example, non-numeric) data into the first control, and then tab to the second. When the error icon is displayed, point at it with the mouse pointer to see the error text.

See Also

[SetError](#)

[ErrorProvider Component Overview](#)

[How to: View Errors Within a DataSet with the Windows Forms ErrorProvider Component](#)

How to: View Errors Within a DataSet with the Windows Forms ErrorProvider Component

5/4/2018 • 1 min to read • [Edit Online](#)

You can use the Windows Forms [ErrorProvider](#) component to view column errors within a dataset or other data source. For an [ErrorProvider](#) component to display data errors on a form, it does not have to be directly associated with a control. Once it is bound to a data source, it can display an error icon next to any control that is bound to the same data source.

NOTE

If you change the error provider's [DataSource](#) and [DataMember](#) properties at run time, you should use the [BindToDataAndErrors](#) method to avoid conflicts.

To display data errors

1. Bind the component to a specific column within a data table.

```
' Assumes existence of DataSet1, DataTable1
TextBox1.DataBindings.Add("Text", DataSet1, "Customers.Name")
ErrorProvider1.DataSource = DataSet1
ErrorProvider1.DataMember = "Customers"
```

```
// Assumes existence of DataSet1, DataTable1
textBox1.DataBindings.Add("Text", DataSet1, "Customers.Name");
errorProvider1.DataSource = DataSet1;
errorProvider1.DataMember = "Customers";
```

2. Set the [ContainerControl](#) property to the form.

```
ErrorProvider1.ContainerControl = Me
```

```
errorProvider1.ContainerControl = this;
```

3. Set the position of the current record to a row that contains a column error.

```
DataTable1.Rows(5).SetColumnError("Name", "Bad data in this row.")
Me.BindingContext(DataTable1).Position = 5
```

```
DataTable1.Rows[5].SetColumnError("Name", "Bad data in this row.");
this.BindingContext [DataTable1].Position = 5;
```

See Also

[ErrorProvider Component Overview](#)

[How to: Display Error Icons for Form Validation with the Windows Forms ErrorProvider Component](#)

FileDialog Class

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [FileDialog](#) class is the common base class for the [OpenFileDialog](#) and [SaveFileDialog](#) components. You can make changes to the [FileDialog](#) class that affect the appearance and behavior of these dialog boxes, depending on the version of Windows the application is running on.

In This Section

[How To: Opt Out of File Dialog Box Automatic Upgrade](#)

Describes how to opt out of a style automatic upgrade to a file dialog box.

[How To: Add a Custom Place to a File Dialog Box](#)

Describes how to add a custom file location to a file dialog box.

[Known Folder GUIDs for File Dialog Custom Places](#)

List the folder names and their associated GUIDs.

Reference

[OpenFileDialog](#)

[SaveFileDialog](#)

Related Sections

[OpenFileDialog Component](#)

[SaveFileDialog Component](#)

How To: Opt Out of File Dialog Box Automatic Upgrade

5/4/2018 • 1 min to read • [Edit Online](#)

When the [OpenFileDialog](#) and [SaveFileDialog](#) classes are used in an application, their appearance and behavior depend on the version of Windows the application is running on. When an application that was created on the .NET Framework 2.0 or earlier is displayed on Windows Vista, [OpenFileDialog](#) and [SaveFileDialog](#) are automatically displayed with the Windows Vista appearance and behavior. Starting in the .NET Framework 3.0, you can opt out of the automatic upgrade to display the [OpenFileDialog](#) and [SaveFileDialog](#) with a Windows XP-style appearance and behavior.

To opt out of file dialog box automatic upgrade

1. Set the [AutoUpgradeEnabled](#) property of [OpenFileDialog](#) or [SaveFileDialog](#) to `false` before you display the dialog box.

See Also

[FileDialog](#)

How To: Add a Custom Place to a File Dialog Box

5/4/2018 • 1 min to read • [Edit Online](#)

The default open and save dialog boxes on Windows Vista have an area on the left side of the dialog box titled **Favorite Links**. This area is called custom places. The [OpenFileDialog](#) and [SaveFileDialog](#) classes allow you to add folders to the [CustomPlaces](#) collection.

NOTE

In order for a custom place to appear in the [OpenFileDialog](#) or [SaveFileDialog](#), the [AutoUpgradeEnabled](#) property must be set to `true` (the default).

To add a custom place to a file dialog box

- Add a path, a Known Folder GUID, or a [FileDialogCustomPlace](#) object to the [CustomPlaces](#) collection of the dialog box.

The following code example shows how to add a path:

```
OpenFileDialog1.CustomPlaces.Add("C:\\MyCustomPlace")
```

```
openFileDialog1.CustomPlaces.Add("C:\\\\MyCustomPlace");
```

See Also

[FileDialog](#)

[FileDialogCustomPlacesCollection.Add](#)

[Known Folder GUIDs for File Dialog Custom Places](#)

Known Folder GUIDs for File Dialog Custom Places

5/4/2018 • 1 min to read • [Edit Online](#)

You use a [Guid](#) to specify a Windows Vista Known Folder when you add folders to a [CustomPlaces](#) collection. Known Folder GUIDs are not case sensitive and are defined in the KnownFolders.h file in the Windows SDK.

NOTE

In some cases, a Known Folder added to the [FileDialogCustomPlacesCollection](#) will not be shown in the **Favorite Links** area. For example, if the specified Known Folder is not present on the computer that is running the application, the Known Folder is not shown.

List of GUIDs

The following table lists Windows Vista Known Folders and their associated [Guid](#).

AddNewPrograms

DE61D971-5EBC-4F02-A3A9-6C82895E5C04

AdminTools

724EF170-A42D-4FEF-9F26-B60E846FBA4F

AppDataLow

A520A1A4-1780-4FF6-BD18-167343C5AF16

AppUpdates

A305CE99-F527-492B-8B1A-7E76FA98D6E4

CDBurning

9E52AB10-F80D-49DF-ACB8-4330F5687855

ChangeRemovePrograms

DF7266AC-9274-4867-8D55-3BD661DE872D

CommonAdminTools

D0384E7D-BAC3-4797-8F14-CBA229B392B5

CommonOEMLinks

C1BAE2D0-10DF-4334-BEDD-7AA20B227A9D

CommonPrograms

0139D44E-6AFE-49F2-8690-3DAFCAE6FFB8

CommonStartMenu

A4115719-D62E-491D-AA7C-E74B8BE3B067

CommonStartup

82A5EA35-D9CD-47C5-9629-E15D2F714E6E

CommonTemplates

B94237E7-57AC-4347-9151-B08C6C32D1F7

Computer

0AC0837C-BBF8-452A-850D-79D08E667CA7

Conflict
4BFEFB45-347D-4006-A5BE-AC0CB0567192

Connections
6F0CD92B-2E97-45D1-88FF-B0D186B8DEDD

Contacts
56784854-C6CB-462B-8169-88E350ACB882

ControlPanel
82A74AEB-AEB4-465C-A014-D097EE346D63

Cookies
2B0F765D-C0E9-4171-908E-08A611B84FF6

Desktop
B4BFCC3A-DB2C-424C-B029-7FE99A87C641

Documents
FDD39AD0-238F-46AF-ADB4-6C85480369C7

Downloads
374DE290-123F-4565-9164-39C4925E467B

Favorites
1777F761-68AD-4D8A-87BD-30B759FA33DD

Fonts
FD228CB7-AE11-4AE3-864C-16F3910AB8FE

Games
CAC52C1A-B53D-4EDC-92D7-6B2E8AC19434

GameTasks
054FAE61-4DD8-4787-80B6-090220C4B700

History
D9DC8A3B-B784-432E-A781-5A1130A75963

Internet
4D9F7874-4E0C-4904-967B-40B0D20C3E4B

InternetCache
352481E8-33BE-4251-BA85-6007CAEDCF9D

Links
BFB9D5E0-C6A9-404C-B2B2-AE6DB6AF4968

LocalAppData
F1B32785-6FBA-4FCF-9D55-7B8E7F157091

LocalizedResourcesDir
2A00375E-224C-49DE-B8D1-440DF7EF3DDC

Music
4BD8D571-6D19-48D3-BE97-422220080E43

NetHood
C5ABB53-E17F-4121-8900-86626FC2C973

Network

D20BEEC4-5CA8-4905-AE3B-BF251EA09B53

OriginalImages

2C36C0AA-5812-4B87-BFD0-4CD0DFB19B39

PhotoAlbums

69D2CF90-FC33-4FB7-9A0C-EBB0F0FCB43C

Pictures

33E28130-4E1E-4676-835A-98395C3BC3BB

Playlists

DE92C1C7-837F-4F69-A3BB-86E631204A23

Printers

76FC4E2D-D6AD-4519-A663-37BD56068185

PrintHood

9274BD8D-CFD1-41C3-B35E-B13F55A758F4

Profile

5E6C858F-0E22-4760-9AFE-EA3317B67173

ProgramData

62AB5D82-FDC1-4DC3-A9DD-070D1D495D97

ProgramFiles

905E63B6-C1BF-494E-B29C-65B732D3D21A

ProgramFilesCommon

F7F1ED05-9F6D-47A2-AAAE-29D317C6F066

ProgramFilesCommonX64

6365D5A7-0F0D-45E5-87F6-0DA56B6A4F7D

ProgramFilesCommonX86

DE974D24-D9C6-4D3E-BF91-F4455120B917

ProgramFilesX64

6D809377-6AF0-444B-8957-A3773F02200E

ProgramFilesX86

7C5A40EF-A0FB-4BFC-874A-C0F2E0B9FA8E

Programs

A77F5D77-2E2B-44C3-A6A2-ABA601054A51

Public

DFDF76A2-C82A-4D63-906A-5644AC457385

PublicDesktop

C4AA340D-F20F-4863-AFF-F87EF2E6BA25

PublicDocuments

ED4824AF-DCE4-45A8-81E2-FC7965083634

PublicDownloads

3D644C9B-1FB8-4F30-9B45-F670235F79C0

PublicGameTasks

DEBF2536-E1A8-4C59-B6A2-414586476AEA

PublicMusic
3214FAB5-9757-4298-BB61-92A9DEAA44FF

PublicPictures
B6EBFB86-6907-413C-9AF7-4FC2ABF07CC5

PublicVideos
2400183A-6185-49FB-A2D8-4A392A602BA3

QuickLaunch
52A4F021-7B75-48A9-9F6B-4B87A210BC8F

Recent
AE50C081-EBD2-438A-8655-8A092E34987A

RecordedTV
BD85E001-112E-431E-983B-7B15AC09FFF1

RecycleBin
B7534046-3ECB-4C18-BE4E-64CD4CB7D6AC

ResourceDir
8AD10C31-2ADB-4296-A8F7-E4701232C972

RoamingAppData
3EB685DB-65F9-4CF6-A03A-E3EF65729F3D

SampleMusic
B250C668-F57D-4EE1-A63C-290EE7D1AA1F

SamplePictures
C4900540-2379-4C75-844B-64E6FAF8716B

SamplePlaylists
15CA69B3-30EE-49C1-ACE1-6B5EC372AFB5

SampleVideos
859EAD94-2E85-48AD-A71A-0969CB56A6CD

SavedGames
4C5C32FF-BB9D-43B0-B5B4-2D72E54EAAA4

SavedSearches
7D1D3A04-DEBB-4115-95CF-2F29DA2920DA

SEARCH_CSC
EE32E446-31CA-4ABA-814F-A5EBD2FD6D5E

SEARCH_MAPI
98EC0E18-2098-4D44-8644-66979315A281

SearchHome
190337D1-B8CA-4121-A639-6D472D16972A

SendTo
8983036C-27C0-404B-8F08-102D10DCFD74

SidebarDefaultParts
7B396E54-9EC5-4300-BE0A-2482EBAE1A26

SidebarParts

A75D362E-50FC-4FB7-AC2C-A8BEAA314493

StartMenu

625B53C3-AB48-4EC1-BA1F-A1EF4146FC19

Startup

B97D20BB-F46A-4C97-BA10-5E3608430854

SyncManager

43668BF8-C14E-49B2-97C9-747784D784B7

SyncResults

289A9A43-BE44-4057-A41B-587A76D7E7F9

SyncSetup

0F214138-B1D3-4A90-BBA9-27CBC0C5389A

System

1AC14E77-02E7-4E5D-B744-2EB1AE5198B7

SystemX86

D65231B0-B2F1-4857-A4CE-A8E7C6EA7D27

Templates

A63293E8-664E-48DB-A079-DF759E0509F7

TreeProperties

5B3749AD-B49F-49C1-83EB-15370FBD4882

UserProfiles

0762D272-C50A-4BB0-A382-697DCD729B80

UsersFiles

F3CE0F7C-4901-4ACC-8648-D5D44B04EF8F

Videos

18989B1D-99B5-455B-841C-AB7C74E4DDFC

Windows

F38BF404-1D43-42F2-9305-67DE0B28FC23

See Also

[FileDialogCustomPlace](#)

[How To: Add a Custom Place to a File Dialog Box](#)

FlowLayoutPanel Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The `FlowLayoutPanel` control arranges its contents in a horizontal or vertical flow direction. Its contents can be wrapped from one row to the next, or from one column to the next. Alternately, its contents can be clipped instead of wrapped.

The topics in this section describe the concepts and techniques that allow you to build `FlowLayoutPanel` features into your applications.

In This Section

[FlowLayoutPanel Control Overview](#)

Introduces the general concepts of the `FlowLayoutPanel` control, which allows you to create a layout which flows horizontally or vertically.

[How to: Anchor and Dock Child Controls in a FlowLayoutPanel Control](#)

Explains how to use the `Anchor` and `Dock` properties to anchor and dock child controls in a `FlowLayoutPanel` control.

Also see [Walkthrough: Arranging Controls on Windows Forms Using a FlowLayoutPanel](#).

Reference

[FlowLayoutPanel](#)

Provides reference documentation for the `FlowLayoutPanel` control.

FlowLayoutPanel Control Overview

5/4/2018 • 1 min to read • [Edit Online](#)

The [FlowLayoutPanel](#) control arranges its contents in a horizontal or vertical flow direction. You can wrap the control's contents from one row to the next, or from one column to the next. Alternately, you can clip instead of wrap its contents.

You can specify the flow direction by setting the value of the [FlowDirection](#) property. The [FlowLayoutPanel](#) control correctly reverses its flow direction in Right-to-Left (RTL) layouts. You can also specify whether the [FlowLayoutPanel](#) control's contents are wrapped or clipped by setting the value of the [WrapContents](#) property.

The [FlowLayoutPanel](#) control automatically sizes to its contents when you set the [AutoSize](#) property to `true`. It also provides a [FlowBreak](#) property to its child controls. Setting the value of the [FlowBreak](#) property to `true` causes the [FlowLayoutPanel](#) control to stop laying out controls in the current flow direction and wrap to the next row or column.

Any Windows Forms control can be a child of the [FlowLayoutPanel](#) control, including other instances of [FlowLayoutPanel](#). With this capability, you can construct sophisticated layouts that adapt to your form's dimensions at run time.

Also see [Walkthrough: Arranging Controls on Windows Forms Using a FlowLayoutPanel](#).

See Also

[FlowDirection](#)

[TableLayoutPanel](#)

[FlowLayoutPanel Control](#)

How to: Anchor and Dock Child Controls in a FlowLayoutPanel Control

5/4/2018 • 7 min to read • [Edit Online](#)

The [FlowLayoutPanel](#) control supports the [Anchor](#) and [Dock](#) properties in its child controls.

To anchor and dock child controls in a FlowLayoutPanel control

1. Create a [FlowLayoutPanel](#) control on your form.
2. Set the [Width](#) of the [FlowLayoutPanel](#) control to **300**, and set its [FlowDirection](#) to [TopDown](#).
3. Create two [Button](#) controls, and place them in the [FlowLayoutPanel](#) control.
4. Set the [Width](#) of the first button to **200**.
5. Set the [Dock](#) property of the second button to [Fill](#).

NOTE

The second button assumes the same width as the first button. It does not stretch across the width of the [FlowLayoutPanel](#) control.

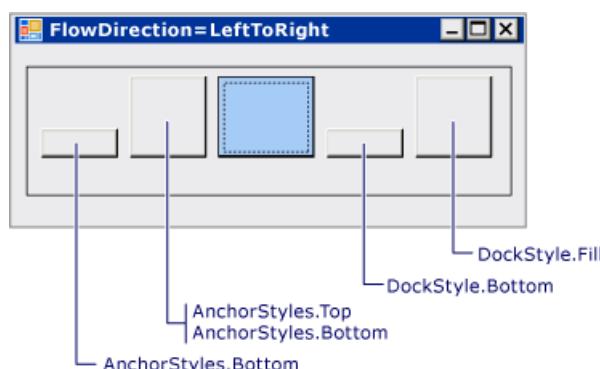
6. Set the [Dock](#) property of the second button to [None](#). This causes the button to assume its original width.
7. Set the [Anchor](#) property of the second button to [Left, Right](#).

IMPORTANT

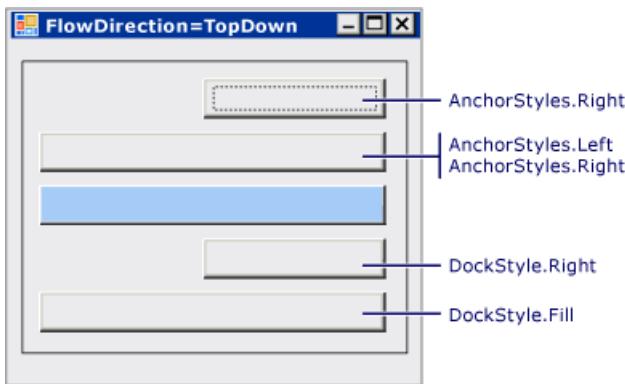
The second button assumes the same width as the first button. It does not stretch across the width of the [FlowLayoutPanel](#) control. This is the general rule for anchoring and docking in the [FlowLayoutPanel](#) control: for vertical flow directions, the [FlowLayoutPanel](#) control calculates the width of an implied column from the widest child control in the column. All other controls in this column with [Anchor](#) or [Dock](#) properties are aligned or stretched to fit this implied column. The behavior works in a similar way for horizontal flow directions. The [FlowLayoutPanel](#) control calculates the height of an implied row from the tallest child control in the row, and all docked or anchored child controls in this row are aligned or sized to fit the implied row.

Example

The following illustration shows four buttons that are anchored and docked relative to the blue button in a [FlowLayoutPanel](#). The [FlowDirection](#) is [LeftToRight](#).



The following illustration shows four buttons that are anchored and docked relative to the blue button in a `FlowLayoutPanel`. The `FlowDirection` is `TopDown`.



The following code example demonstrates various `Anchor` property values for a `Button` control in a `FlowLayoutPanel` control.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Windows.Forms;

public class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

    private FlowLayoutPanel flowLayoutPanel3;
    private Label label2;
    private Button button11;
    private Button button12;
    private Button button13;
    private Button button14;
    private Button button15;
    private FlowLayoutPanel flowLayoutPanel1;
    private Label label1;
    private Button button1;
    private Button button2;
    private Button button3;
    private Button button4;
    private Button button5;

    private System.ComponentModel.IContainer components = null;

    protected override void Dispose(bool disposing)
    {
        if (disposing && (components != null))
        {
            components.Dispose();
        }
        base.Dispose(disposing);
    }

    private void InitializeComponent()
    {
        this.flowLayoutPanel3 = new System.Windows.Forms.FlowLayoutPanel();
        this.label2 = new System.Windows.Forms.Label();
        this.button11 = new System.Windows.Forms.Button();
        this.button12 = new System.Windows.Forms.Button();
        this.button13 = new System.Windows.Forms.Button();
        this.button14 = new System.Windows.Forms.Button();
    }
}
```

```
    this.button1 = new System.Windows.Forms.Button();
    this.flowLayoutPanel1 = new System.Windows.Forms.FlowLayoutPanel();
    this.label1 = new System.Windows.Forms.Label();
    this.button2 = new System.Windows.Forms.Button();
    this.button3 = new System.Windows.Forms.Button();
    this.button4 = new System.Windows.Forms.Button();
    this.button5 = new System.Windows.Forms.Button();
    this.flowLayoutPanel3.SuspendLayout();
    this.flowLayoutPanel1.SuspendLayout();
    this.SuspendLayout();
    //
    // flowLayoutPanel3
    //
    this.flowLayoutPanel3.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Bottom) | System.Windows.Forms.AnchorStyles.Left) | System.Windows.Forms.AnchorStyles.Right));
    this.flowLayoutPanel3.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle;
    this.flowLayoutPanel3.Controls.Add(this.label2);
    this.flowLayoutPanel3.Controls.Add(this.button11);
    this.flowLayoutPanel3.Controls.Add(this.button12);
    this.flowLayoutPanel3.Controls.Add(this.button13);
    this.flowLayoutPanel3.Controls.Add(this.button14);
    this.flowLayoutPanel3.Controls.Add(this.button15);
    this.flowLayoutPanel3.Location = new System.Drawing.Point(12, 12);
    this.flowLayoutPanel3.Name = "flowLayoutPanel3";
    this.flowLayoutPanel3.Size = new System.Drawing.Size(631, 100);
    this.flowLayoutPanel3.TabIndex = 2;
    //
    // label2
    //
    this.label2.Anchor = System.Windows.Forms.AnchorStyles.None;
    this.label2.AutoSize = true;
    this.label2.Location = new System.Drawing.Point(3, 28);
    this.label2.Name = "label2";
    this.label2.Size = new System.Drawing.Size(138, 14);
    this.label2.TabIndex = 10;
    this.label2.Text = "FlowDirection=LeftToRight";
    //
    // button11
    //
    this.button11.Anchor = System.Windows.Forms.AnchorStyles.Bottom;
    this.button11.AutoSize = true;
    this.button11.Location = new System.Drawing.Point(147, 44);
    this.button11.Name = "button11";
    this.button11.Size = new System.Drawing.Size(86, 23);
    this.button11.TabIndex = 5;
    this.button11.Text = "Anchor=Bottom";
    //
    // button12
    //
    this.button12.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Bottom)));
    this.button12.AutoSize = true;
    this.button12.Location = new System.Drawing.Point(239, 3);
    this.button12.Name = "button12";
    this.button12.Size = new System.Drawing.Size(111, 64);
    this.button12.TabIndex = 6;
    this.button12.Text = "Anchor=Top, Bottom";
    //
    // button13
    //
    this.button13.Anchor = System.Windows.Forms.AnchorStyles.None;
    this.button13.BackColor = System.Drawing.SystemColors.GradientActiveCaption;
    this.button13.Location = new System.Drawing.Point(356, 3);
    this.button13.Name = "button13";
    this.button13.Size = new System.Drawing.Size(75, 64);
    this.button13.TabIndex = 7.
```

```
this.button14.TabIndex = 7;
//
// button14
//
this.button14.Dock = System.Windows.Forms.DockStyle.Bottom;
this.button14.Location = new System.Drawing.Point(437, 44);
this.button14.Name = "button14";
this.button14.TabIndex = 8;
this.button14.Text = "Dock=Bottom";
//
// button15
//
this.button15.Dock = System.Windows.Forms.DockStyle.Fill;
this.button15.Location = new System.Drawing.Point(518, 3);
this.button15.Name = "button15";
this.button15.Size = new System.Drawing.Size(75, 64);
this.button15.TabIndex = 9;
this.button15.Text = "Dock=Fill";
//
// flowLayoutPanel1
//
this.flowLayoutPanel1.Anchor = ((System.Windows.Forms.AnchorStyles)
((System.Windows.Forms.AnchorStyles.Bottom | System.Windows.Forms.AnchorStyles.Left)
| System.Windows.Forms.AnchorStyles.Right)));
this.flowLayoutPanel1BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle;
this.flowLayoutPanel1.Controls.Add(this.label1);
this.flowLayoutPanel1.Controls.Add(this.button1);
this.flowLayoutPanel1.Controls.Add(this.button2);
this.flowLayoutPanel1.Controls.Add(this.button3);
this.flowLayoutPanel1.Controls.Add(this.button4);
this.flowLayoutPanel1.Controls.Add(this.button5);
this.flowLayoutPanel1.FlowDirection = System.Windows.Forms.FlowDirection.TopDown;
this.flowLayoutPanel1.Location = new System.Drawing.Point(12, 118);
this.flowLayoutPanel1.Name = "flowLayoutPanel1";
this.flowLayoutPanel1.Size = new System.Drawing.Size(200, 209);
this.flowLayoutPanel1.TabIndex = 3;
//
// label1
//
this.label1.AutoSize = true;
this.label1.Location = new System.Drawing.Point(3, 3);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(128, 14);
this.label1.TabIndex = 11;
this.label1.Text = "FlowDirection=TopDown";
//
// button1
//
this.button1.Anchor = System.Windows.Forms.AnchorStyles.Right;
this.button1.Location = new System.Drawing.Point(74, 23);
this.button1.Name = "button1";
this.button1.TabIndex = 5;
this.button1.Text = "Anchor=Right";
//
// button2
//
this.button2.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Left |
System.Windows.Forms.AnchorStyles.Right)));
this.button2.Location = new System.Drawing.Point(3, 52);
this.button2.Name = "button2";
this.button2.Size = new System.Drawing.Size(146, 23);
this.button2.TabIndex = 6;
this.button2.Text = "Anchor=Left, Right";
//
// button3
//
this.button3.BackColor = System.Drawing.SystemColors.GradientActiveCaption;
this.button3.Location = new System.Drawing.Point(3, 81);
this.button3.Name = "button3";
```

```

this.button3.Size = new System.Drawing.Size(146, 23);
this.button3.TabIndex = 7;
//
// button4
//
this.button4.Dock = System.Windows.Forms.DockStyle.Right;
this.button4.Location = new System.Drawing.Point(74, 110);
this.button4.Name = "button4";
this.button4.TabIndex = 8;
this.button4.Text = "Dock=Right";
//
// button5
//
this.button5.Dock = System.Windows.Forms.DockStyle.Fill;
this.button5.Location = new System.Drawing.Point(3, 139);
this.button5.Name = "button5";
this.button5.Size = new System.Drawing.Size(146, 23);
this.button5.TabIndex = 9;
this.button5.Text = "Dock=Fill";
//
// Form1
//
this.ClientSize = new System.Drawing.Size(658, 341);
this.Controls.Add(this.flowLayoutPanel1);
this.Controls.Add(this.flowLayoutPanel3);
this.Name = "Form1";
this.Text = "Form1";
this.flowLayoutPanel3.ResumeLayout(false);
this.flowLayoutPanel3.PerformLayout();
this.flowLayoutPanel1.ResumeLayout(false);
this.flowLayoutPanel1.PerformLayout();
this.ResumeLayout(false);
}

[STAThread]
static void Main()
{
    Application.EnableVisualStyles();
    Application.Run(new Form1());
}
}

```

```

Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Drawing
Imports System.Windows.Forms

Public Class Form1
    Inherits Form

    Public Sub New()
        InitializeComponent()
    End Sub 'New
    Private flowLayoutPanel3 As FlowLayoutPanel
    Private label2 As Label
    Private button11 As Button
    Private button12 As Button
    Private button13 As Button
    Private button14 As Button
    Private button15 As Button
    Private flowLayoutPanel1 As FlowLayoutPanel
    Private label1 As Label

```

```

Private button1 As Button
Private button2 As Button
Private button3 As Button
Private button4 As Button
Private button5 As Button

Private components As System.ComponentModel.IContainer = Nothing

Protected Overrides Sub Dispose(disposing As Boolean)
    If disposing AndAlso (components IsNot Nothing) Then
        components.Dispose()
    End If
    MyBase.Dispose(disposing)
End Sub 'Dispose

Private Sub InitializeComponent()
    Me.flowLayoutPanel3 = New System.Windows.Forms.FlowLayoutPanel()
    Me.label2 = New System.Windows.Forms.Label()
    Me.button11 = New System.Windows.Forms.Button()
    Me.button12 = New System.Windows.Forms.Button()
    Me.button13 = New System.Windows.Forms.Button()
    Me.button14 = New System.Windows.Forms.Button()
    Me.button15 = New System.Windows.Forms.Button()
    Me.flowLayoutPanel11 = New System.Windows.Forms.FlowLayoutPanel()
    Me.label11 = New System.Windows.Forms.Label()
    Me.button1 = New System.Windows.Forms.Button()
    Me.button2 = New System.Windows.Forms.Button()
    Me.button3 = New System.Windows.Forms.Button()
    Me.button4 = New System.Windows.Forms.Button()
    Me.button5 = New System.Windows.Forms.Button()
    Me.flowLayoutPanel3.SuspendLayout()
    Me.flowLayoutPanel11.SuspendLayout()
    Me.SuspendLayout()

    ' flowLayoutPanel3
    ' 

    Me.flowLayoutPanel3.Anchor = CType(System.Windows.Forms.AnchorStyles.Top Or
System.Windows.Forms.AnchorStyles.Bottom Or System.Windows.Forms.AnchorStyles.Left Or
System.Windows.Forms.AnchorStyles.Right, System.Windows.Forms.AnchorStyles)
    Me.flowLayoutPanel3.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle
    Me.flowLayoutPanel3.Controls.Add(Me.label2)
    Me.flowLayoutPanel3.Controls.Add(Me.button11)
    Me.flowLayoutPanel3.Controls.Add(Me.button12)
    Me.flowLayoutPanel3.Controls.Add(Me.button13)
    Me.flowLayoutPanel3.Controls.Add(Me.button14)
    Me.flowLayoutPanel3.Controls.Add(Me.button15)
    Me.flowLayoutPanel3.Location = New System.Drawing.Point(12, 12)
    Me.flowLayoutPanel3.Name = "flowLayoutPanel3"
    Me.flowLayoutPanel3.Size = New System.Drawing.Size(631, 100)
    Me.flowLayoutPanel3.TabIndex = 2

    ' label2
    ' 

    Me.label2.Anchor = System.Windows.Forms.AnchorStyles.None
    Me.label2.AutoSize = True
    Me.label2.Location = New System.Drawing.Point(3, 28)
    Me.label2.Name = "label2"
    Me.label2.Size = New System.Drawing.Size(138, 14)
    Me.label2.TabIndex = 10
    Me.label2.Text = "FlowDirection=LeftToRight"

    ' button11
    ' 

    Me.button11.Anchor = System.Windows.Forms.AnchorStyles.Bottom
    Me.button11.AutoSize = True
    Me.button11.Location = New System.Drawing.Point(147, 44)
    Me.button11.Name = "button11"

```

```
Me.button11.Size = New System.Drawing.Size(86, 23)
Me.button11.TabIndex = 5
Me.button11.Text = "Anchor=Bottom"
'

' button12
'

Me.button12.Anchor = CType(System.Windows.Forms.AnchorStyles.Top Or
System.Windows.Forms.AnchorStyles.Bottom, System.Windows.Forms.AnchorStyles)
Me.button12.AutoSize = True
Me.button12.Location = New System.Drawing.Point(239, 3)
Me.button12.Name = "button12"
Me.button12.Size = New System.Drawing.Size(111, 64)
Me.button12.TabIndex = 6
Me.button12.Text = "Anchor=Top, Bottom"
'

' button13
'

Me.button13.Anchor = System.Windows.Forms.AnchorStyles.None
Me.button13.BackColor = System.Drawing.SystemColors.GradientActiveCaption
Me.button13.Location = New System.Drawing.Point(356, 3)
Me.button13.Name = "button13"
Me.button13.Size = New System.Drawing.Size(75, 64)
Me.button13.TabIndex = 7
'

' button14
'

Me.button14.Dock = System.Windows.Forms.DockStyle.Bottom
Me.button14.Location = New System.Drawing.Point(437, 44)
Me.button14.Name = "button14"
Me.button14.TabIndex = 8
Me.button14.Text = "Dock=Bottom"
'

' button15
'

Me.button15.Dock = System.Windows.Forms.DockStyle.Fill
Me.button15.Location = New System.Drawing.Point(518, 3)
Me.button15.Name = "button15"
Me.button15.Size = New System.Drawing.Size(75, 64)
Me.button15.TabIndex = 9
Me.button15.Text = "Dock=Fill"
'

' flowLayoutPanel1
'

Me.flowLayoutPanel1.Anchor = CType(System.Windows.Forms.AnchorStyles.Bottom Or
System.Windows.Forms.AnchorStyles.Left Or System.Windows.Forms.AnchorStyles.Right,
System.Windows.Forms.AnchorStyles)
Me.flowLayoutPanel1.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle
Me.flowLayoutPanel1.Controls.Add(Me.label1)
Me.flowLayoutPanel1.Controls.Add(Me.button1)
Me.flowLayoutPanel1.Controls.Add(Me.button2)
Me.flowLayoutPanel1.Controls.Add(Me.button3)
Me.flowLayoutPanel1.Controls.Add(Me.button4)
Me.flowLayoutPanel1.Controls.Add(Me.button5)
Me.flowLayoutPanel1.FlowDirection = System.Windows.Forms.FlowDirection.TopDown
Me.flowLayoutPanel1.Location = New System.Drawing.Point(12, 118)
Me.flowLayoutPanel1.Name = "flowLayoutPanel1"
Me.flowLayoutPanel1.Size = New System.Drawing.Size(200, 209)
Me.flowLayoutPanel1.TabIndex = 3
'

' label1
'

Me.label1.AutoSize = True
Me.label1.Location = New System.Drawing.Point(3, 3)
Me.label1.Name = "label1"
Me.label1.Size = New System.Drawing.Size(128, 14)
Me.label1.TabIndex = 11
Me.label1.Text = "FlowDirection=TopDown"
'

' button1
```

```

        '
Me.button1.Anchor = System.Windows.Forms.AnchorStyles.Right
Me.button1.Location = New System.Drawing.Point(74, 23)
Me.button1.Name = "button1"
Me.button1.TabIndex = 5
Me.button1.Text = "Anchor=Right"
'
' button2
'
Me.button2.Anchor = CType(System.Windows.Forms.AnchorStyles.Left Or
System.Windows.Forms.AnchorStyles.Right, System.Windows.Forms.AnchorStyles)
Me.button2.Location = New System.Drawing.Point(3, 52)
Me.button2.Name = "button2"
Me.button2.Size = New System.Drawing.Size(146, 23)
Me.button2.TabIndex = 6
Me.button2.Text = "Anchor=Left, Right"
'
' button3
'
Me.button3.BackColor = System.Drawing.SystemColors.GradientActiveCaption
Me.button3.Location = New System.Drawing.Point(3, 81)
Me.button3.Name = "button3"
Me.button3.Size = New System.Drawing.Size(146, 23)
Me.button3.TabIndex = 7
'
' button4
'
Me.button4.Dock = System.Windows.Forms.DockStyle.Right
Me.button4.Location = New System.Drawing.Point(74, 110)
Me.button4.Name = "button4"
Me.button4.TabIndex = 8
Me.button4.Text = "Dock=Right"
'
' button5
'
Me.button5.Dock = System.Windows.Forms.DockStyle.Fill
Me.button5.Location = New System.Drawing.Point(3, 139)
Me.button5.Name = "button5"
Me.button5.Size = New System.Drawing.Size(146, 23)
Me.button5.TabIndex = 9
Me.button5.Text = "Dock=Fill"
'
' Form1
'
Me.ClientSize = New System.Drawing.Size(658, 341)
Me.Controls.Add(flowLayoutPanel1)
Me.Controls.Add(flowLayoutPanel3)
Me.Name = "Form1"
Me.Text = "Form1"
Me.flowLayoutPanel3.ResumeLayout(False)
Me.flowLayoutPanel3.PerformLayout()
Me.flowLayoutPanel1.ResumeLayout(False)
Me.flowLayoutPanel1.PerformLayout()
Me.ResumeLayout(False)
End Sub 'InitializeComponent

<STAThread()> _
Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())
End Sub 'Main
End Class 'Form1

```

Compiling the Code

This example requires:

- References to the System, System.Data, System.Drawing and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[FlowLayoutPanel](#)

[FlowLayoutPanel Control Overview](#)

Walkthrough: Arranging Controls on Windows Forms Using a FlowLayoutPanel

5/4/2018 • 8 min to read • [Edit Online](#)

Some applications require a form with a layout that arranges itself appropriately as the form is resized or as the contents change in size. When you need a dynamic layout and you do not want to handle [Layout](#) events explicitly in your code, consider using a layout panel.

The [FlowLayoutPanel](#) control and the [TableLayoutPanel](#) control provide intuitive ways to arrange controls on your form. Both provide an automatic, configurable ability to control the relative positions of child controls contained within them, and both give you dynamic layout features at run time, so they can resize and reposition child controls as the dimensions of the parent form change. Layout panels can be nested within layout panels, to enable the realization of sophisticated user interfaces.

The [TableLayoutPanel](#) arranges its contents in a grid, providing functionality similar to the HTML <table> element. Its cells are arranged in rows and columns, and these can have different sizes. For more information, see [Walkthrough: Arranging Controls on Windows Forms Using a TableLayoutPanel](#).

The [FlowLayoutPanel](#) arranges its contents in a specific flow direction: horizontal or vertical. Its contents can be wrapped from one row to the next, or from one column to the next. Alternately, its contents can be clipped instead of wrapped. Tasks illustrated in this walkthrough include:

- Creating a Windows Forms project
- Arranging Controls Horizontally and Vertically
- Changing Flow Direction
- Inserting Flow Breaks
- Arranging Controls Using Padding and Margins
- Inserting Controls by Double-clicking Them in the Toolbox
- Inserting a Control by Drawing Its Outline
- Inserting Controls Using the Caret
- Reassigning Existing Controls to a Different Parent

When you are finished, you will have an understanding of the role played by these important layout features.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

Creating the Project

The first step is to create the project and set up the form.

To create the project

1. Create a Windows-based application project called "FlowLayoutPanelExample". For more information, see

How to: Create a Windows Application Project.

2. Select the form in the **Forms Designer**.

Arranging Controls Horizontally and Vertically

The [FlowLayoutPanel](#) control allows you to place controls along rows or columns without requiring you to precisely specify the position of each individual control.

The [FlowLayoutPanel](#) control can resize or reflow its child controls as the dimensions of the parent form change.

To arrange controls horizontally and vertically using a FlowLayoutPanel

1. Drag a [FlowLayoutPanel](#) control from the **Toolbox** onto your form.
2. Drag a [Button](#) control from the **Toolbox** into the [FlowLayoutPanel](#). Note that it is automatically moved to the upper-left corner of the [FlowLayoutPanel](#) control.
3. Drag another [Button](#) control from the **Toolbox** into the [FlowLayoutPanel](#). Note that the [Button](#) control is automatically moved to a position next to the first [Button](#) control. If your [FlowLayoutPanel](#) is too narrow to fit the two controls on the same row, the new [Button](#) control is automatically moved to the next row.
4. Drag several more [Button](#) controls from the **Toolbox** into the [FlowLayoutPanel](#). Continue placing [Button](#) controls until one wraps to the next row.
5. Change the value of the [FlowLayoutPanel](#) control's [WrapContents](#) property to `false`. Note that the child controls no longer flow to the next row. Instead, they are moved to the first row and clipped.
6. Change the value of the [FlowLayoutPanel](#) control's [WrapContents](#) property to `true`. Note that the child controls again wrap to the next row.
7. Decrease the width of the [FlowLayoutPanel](#) control until all the [Button](#) controls are moved into the first column.
8. Increase the width of the [FlowLayoutPanel](#) control until all the [Button](#) controls are moved into the first row. You may need to resize your form to accommodate the greater width.

Changing Flow Direction

The [FlowDirection](#) property allows you to change the direction in which controls are arranged. You can arrange the child controls from left to right, from right to left, from top to bottom, or from bottom to top.

To change the flow direction in a FlowLayoutPanel

1. Change the value of the [FlowLayoutPanel](#) control's [FlowDirection](#) property to [TopDown](#). Note that the child controls are rearranged into one or more columns, depending on the height of the control.
2. Resize the [FlowLayoutPanel](#) so its height is shorter than the column of [Button](#) controls. Note that the [FlowLayoutPanel](#) rearranges the child controls to flow into the next column. Continue decreasing the height and note that the child controls flow into consecutive columns. Change the value of the [FlowLayoutPanel](#) control's [FlowDirection](#) property to [RightToLeft](#). Note that the positions of the child controls are reversed. Observe the layout when you change the value of the [FlowDirection](#) property to [BottomUp](#).

Inserting Flow Breaks

The [FlowLayoutPanel](#) control provides a [FlowBreak](#) property to its child controls. Setting the value of the [FlowBreak](#) property to `true` causes the [FlowLayoutPanel](#) control to stop laying out controls in the current flow direction and wrap to the next row or column.

To insert flow breaks

1. Change the value of the [FlowLayoutPanel](#) control's [FlowDirection](#) property to [TopDown](#).
2. Select one of the [Button](#) controls in the middle of the leftmost column.
3. Set the value of the [Button](#) control's [FlowBreak](#) property to [true](#). Note that the column is broken and the controls following the selected [Button](#) control flow into the next column. Set the value of the [Button](#) control's [FlowBreak](#) property to [false](#) to return to the original behavior.

Positioning Controls Using Docking and Anchoring

Docking and anchoring behaviors of child controls differ from the behaviors in other container controls. Both docking and anchoring are relative to the largest control in the flow direction.

To position controls using docking and anchoring

1. Increase the size of the [FlowLayoutPanel](#) until the [Button](#) controls are all arranged in a column.
2. Select the top [Button](#) control. Increase its width so that it is about twice as wide as the other [Button](#) controls.
3. Select the second [Button](#) control. Change the value of its [Anchor](#) property to [Right](#). Note that it is moved so that its right border is aligned with the first [Button](#) control's right border.
4. Change the value of its [Anchor](#) property to [Right](#) and [Left](#). Note that it is sized to the same width as the first [Button](#) control.
5. Select the third [Button](#) control. Change the value of its [Dock](#) property to [Fill](#). Note that it is sized to the same width as the first [Button](#) control.

Arranging Controls Using Padding and Margins

You can also arrange controls in your [FlowLayoutPanel](#) control by changing the [Padding](#) and [Margin](#) properties.

The [Padding](#) property allows you to control the placement of controls within a [FlowLayoutPanel](#) control's cell. It specifies the spacing between the child controls and the [FlowLayoutPanel](#) control's border.

The [Margin](#) property allows you to control the spacing between controls.

To arrange controls using the Padding and Margin properties

1. Change the value of the [FlowLayoutPanel](#) control's [Dock](#) property to [Fill](#). If your form is large enough, the [Button](#) controls will be moved into the first column of the [FlowLayoutPanel](#) control.
2. Change the value of the [FlowLayoutPanel](#) control's [Padding](#) property by expanding the [Padding](#) entry in the [Properties](#) window and setting the [All](#) property to **20**. For more information, see [Walkthrough: Laying Out Windows Forms Controls with Padding, Margins, and the AutoSize Property](#). Note that the child controls are moved toward the center of the [FlowLayoutPanel](#) control. The increased value for the [Padding](#) property pushes the child controls away from the [FlowLayoutPanel](#) control's borders.
3. Select all of the [Button](#) controls in the [FlowLayoutPanel](#) and set the value of the [Margin](#) property to **20**. Note that the spacing between the [Button](#) controls increases, so they are moved further apart. You may need to resize the [FlowLayoutPanel](#) control to be larger to see all of the child controls.

Inserting Controls by Double-clicking Them in the Toolbox

You can populate your [FlowLayoutPanel](#) control by double-clicking controls in the [Toolbox](#).

To insert controls by double-clicking in the Toolbox

1. Double-click the [Button](#) control icon in the [Toolbox](#). Note that a new [Button](#) control appears in the [FlowLayoutPanel](#) control.

2. Double-click several more controls in the **Toolbox**. Note that the new controls appear successively in the **FlowLayoutPanel** control.

Inserting a Control by Drawing Its Outline

You can insert a control into a **FlowLayoutPanel** control and specify its size by drawing its outline in a cell.

To insert a Control by drawing its outline

1. In the **Toolbox**, click the **Button** control icon. Do not drag it onto the form.
2. Move the mouse pointer over the **FlowLayoutPanel** control. Note that the pointer changes to a crosshair with the **Button** control icon attached.
3. Click and hold the mouse button.
4. Drag the mouse pointer to draw the outline of the **Button** control. When you are satisfied with the size, release the mouse button. Note that the **Button** control is created in the next open location of the **FlowLayoutPanel** control.

Inserting Controls Using the Insertion Bar

You can insert controls at a specific position in a **FlowLayoutPanel** control. When you drag a control into the **FlowLayoutPanel** control's client area, an insertion bar appears to indicate where the control will be inserted.

To insert a control using the caret

1. Drag a **Button** control from the **Toolbox** into the **FlowLayoutPanel** control and point to the space between two **Button** controls. Note that an insertion bar is drawn, indicating where the **Button** will be placed when it is dropped into the **FlowLayoutPanel** control. Before you drop the new **Button** control into the **FlowLayoutPanel** control, move the mouse pointer around to observe how the insertion bar moves.
2. Drop the new **Button** control into the **FlowLayoutPanel** control. Note that the new **Button** control is not aligned with the others, because its **Margin** property has a different value.

Reassigning Existing Controls to a Different Parent

You can assign controls that exist on your form to a new **FlowLayoutPanel** control.

To reparent existing controls

1. Drag three **Button** controls from the **Toolbox** onto the form. Position them near to each other, but leave them unaligned.
2. In the **Toolbox**, click the **FlowLayoutPanel** control icon. Do not drag it onto the form.
3. Move the mouse pointer close to the three **Button** controls. Note that the pointer changes to a crosshair with the **FlowLayoutPanel** control icon attached.
4. Click and hold the mouse button.
5. Drag the mouse pointer to draw the outline of the **FlowLayoutPanel** control. Draw the outline around the three **Button** controls.
6. Release the mouse button. Note that the three **Button** controls are inserted into the **FlowLayoutPanel** control.

Next Steps

You can achieve a complex layout using a combination of layout panels and controls. Suggestions for more exploration include:

- Resize one of the [Button](#) controls to a larger size and note the effect on the layout.
- Layout panels can contain other layout panels. Experiment with dropping a [TableLayoutPanel](#) control into the existing control.
- Dock the [FlowLayoutPanel](#) control to the parent form. Resize the form and note the effect on the layout.
- Set the [Visible](#) property of one of the controls to `false` and note how the [FlowLayoutPanel](#) reflows in response.

See Also

[FlowLayoutPanel](#)

[TableLayoutPanel](#)

[Walkthrough: Arranging Controls on Windows Forms Using a TableLayoutPanel](#)

[Walkthrough: Arranging Controls on Windows Forms Using Snaplines](#)

[Microsoft Windows User Experience, Official Guidelines for User Interface Developers and Designers. Redmond, WA: Microsoft Press, 1999. \(ISBN: 0-7356-0566-1\)](#)

[AutoSize Property Overview](#)

[How to: Dock Controls on Windows Forms](#)

[How to: Anchor Controls on Windows Forms](#)

[Walkthrough: Laying Out Windows Forms Controls with Padding, Margins, and the AutoSize Property](#)

FolderBrowserDialog Component (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `FolderBrowserDialog` component displays an interface with which users can browse and select a folder or create a new one. It is a complement to the [OpenFileDialog Component](#) component, which is used for browsing and selecting files.

In This Section

[FolderBrowserDialog Component Overview \(Windows Forms\)](#)

Explains what this control is and its key features and properties.

[How to: Choose Folders with the Windows Forms FolderBrowserDialog Component](#)

Explains how to programmatically extract the selected folder from the dialog box, as well as work with some of the other, optional properties of the component.

Reference

[FolderBrowserDialog](#)

Describes this class and has links to all its members.

Related Sections

[Dialog Boxes in Windows Forms](#)

Provides a list of tasks for dialog boxes, which often display tabs.

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

FolderBrowserDialog Component Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [FolderBrowserDialog](#) component is a modal dialog box that is used for browsing and selecting folders. New folders can also be created from within the [FolderBrowserDialog](#) component.

NOTE

To select files, instead of folders, use the [OpenFileDialog](#) component.

The [FolderBrowserDialog](#) component is displayed at run time using the [ShowDialog](#) method. Set the [RootFolder](#) property to determine the top-most folder and any subfolders that will appear within the tree view of the dialog box. Once the dialog box has been shown, you can use the [SelectedPath](#) property to get the path of the folder that was selected.

When it is added to a form, the [FolderBrowserDialog](#) component appears in the tray at the bottom of the Windows Forms Designer.

See Also

[FolderBrowserDialog](#)

[How to: Choose Folders with the Windows Forms FolderBrowserDialog Component](#)

[FolderBrowserDialog Component](#)

How to: Choose Folders with the Windows Forms FolderBrowserDialog Component

5/4/2018 • 1 min to read • [Edit Online](#)

Often, within Windows applications you create, you will have to prompt users to select a folder, most frequently to save a set of files. The Windows Forms [FolderBrowserDialog](#) component allows you to easily accomplish this task.

To choose folders with the [FolderBrowserDialog](#) component

1. In a procedure, check the [FolderBrowserDialog](#) component's [DialogResult](#) property to see how the dialog box was closed and get the value of the [FolderBrowserDialog](#) component's [SelectedPath](#) property.
2. If you need to set the top-most folder that will appear within the tree view of the dialog box, set the [RootFolder](#) property, which takes a member of the [Environment.SpecialFolder](#) enumeration.
3. Additionally, you can set the [Description](#) property, which specifies the text string that appears at the top of the folder-browser tree view.

In the example below, the [FolderBrowserDialog](#) component is used to select a folder, similar to when you create a project in Visual Studio and are prompted to select a folder to save it in. In this example, the folder name is then displayed in a [TextBox](#) control on the form. It is a good idea to place the location in an editable area, such as a [TextBox](#) control, so that users may edit their selection in case of error or other issues. This example assumes a form with a [FolderBrowserDialog](#) component and a [TextBox](#) control.

```
Public Sub ChooseFolder()
    If FolderBrowserDialog1.ShowDialog() = DialogResult.OK Then
        TextBox1.Text = FolderBrowserDialog1.SelectedPath
    End If
End Sub
```

```
public void ChooseFolder()
{
    if (folderBrowserDialog1.ShowDialog() == DialogResult.OK)
    {
        textBox1.Text = folderBrowserDialog1.SelectedPath;
    }
}
```

```
public:
void ChooseFolder()
{
    if (folderBrowserDialog1->ShowDialog() == DialogResult::OK)
    {
        textBox1->Text = folderBrowserDialog1->SelectedPath;
    }
}
```

IMPORTANT

To use this class, your assembly requires a privilege level granted by the [PathDiscovery](#) property, which is part of the [FileIOPermissionAccess](#) enumeration. If you are running in a partial-trust context, the process might throw an exception because of insufficient privileges. For more information, see [Code Access Security Basics](#).

For information on how to save files, see [How to: Save Files Using the SaveFileDialog Component](#).

See Also

[FolderBrowserDialog](#)

[FolderBrowserDialog Component Overview \(Windows Forms\)](#)

[FolderBrowserDialog Component](#)

FontDialog Component (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [FontDialog](#) component is a pre-configured dialog box. It is the same **Font** dialog box exposed by the Windows operating system. The component inherits from the [CommonDialog](#) class.

In This Section

[FontDialog Component Overview](#)

Introduces the general concepts of the [FontDialog](#) component, which you use to display a pre-configured dialog box. Users can use the dialog box to manipulate fonts and their settings.

[How to: Show a Font List with the FontDialog Component](#)

Explains how to choose a font at run time through an instance of the [FontDialog](#) component.

Reference

[FontDialog](#)

Provides reference information about the [FontDialog](#) class and its members.

Related Sections

[Dialog-Box Controls and Components](#)

Describes a set of controls and components that enable users to perform standard interactions with the application or system.

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information about their use.

FontDialog Component Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [FontDialog](#) component is a pre-configured dialog box, which is the standard Windows **Font** dialog box used to expose the fonts that are currently installed on the system. Use it within your Windows-based application as a simple solution for font selection in lieu of configuring your own dialog box.

By default, the dialog box shows list boxes for Font, Font style, and Size; check boxes for effects like Strikeout and Underline; a drop-down list for Script; and a sample of how the font will appear. (Script refers to different character scripts that are available for a given font, for example, Hebrew or Japanese.) To display the font dialog box, call the [ShowDialog](#) method.

Key Properties

The component has a number of properties that configure its appearance. The properties that set the dialog-box selections are [Font](#) and [Color](#). The [Font](#) property sets the font, style, size, script, and effects; for example,

```
Arial, 10pt, style=Italic, Strikeout.
```

See Also

[FontDialog](#)

[FontDialog Component](#)

How to: Show a Font List with the FontDialog Component

5/4/2018 • 1 min to read • [Edit Online](#)

The [FontDialog](#) component allows users to select a font, as well as change its display aspects, such as its weight and size.

The font selected in the dialog box is returned in the [Font](#) property. Thus, taking advantage of the font selected by the user is as easy as reading a property.

To select font properties using the FontDialog Component

1. Display the dialog box using the [ShowDialog](#) method.
2. Use the [DialogResult](#) property to determine how the dialog box was closed.
3. Use the [Font](#) property to set the desired font.

In the example below, the [Button](#) control's [Click](#) event handler opens a [FontDialog](#) component. When a font is chosen and the user clicks **OK**, the [Font](#) property of a [TextBox](#) control that is on the form is set to the chosen font. The example assumes your form has a [Button](#) control, a [TextBox](#) control, and a [FontDialog](#) component.

```
Private Sub Button1_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles Button1.Click  
    If FontDialog1.ShowDialog() = DialogResult.OK Then  
        TextBox1.Font = FontDialog1.Font  
    End If  
End Sub
```

```
private void button1_Click(object sender, System.EventArgs e)  
{  
    if(fontDialog1.ShowDialog() == DialogResult.OK)  
    {  
        textBox1.Font = fontDialog1.Font;  
    }  
}
```

```
private:  
    void button1_Click(System::Object ^ sender,  
        System::EventArgs ^ e)  
    {  
        if(fontDialog1->ShowDialog() == DialogResult::OK)  
        {  
            textBox1->Font = fontDialog1->Font;  
        }  
    }  
}
```

(Visual C# and Visual C++) Place the following code in the form's constructor to register the event handler.

```
this.button1.Click += new System.EventHandler(this.button1_Click);
```

```
button1->Click += gcnew System::EventHandler(this, &Form1::button1_Click);
```

See Also

[FontDialog](#)

[FontDialog Component](#)

GroupBox Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

Windows Forms [GroupBox](#) controls are used to provide an identifiable grouping for other controls. Typically, you use group boxes to subdivide a form by function. For example, you may have an order form that specifies mailing options such as which overnight carrier to use. Grouping all options in a group box gives the user a logical visual cue. The [GroupBox](#) control is similar to the [Panel](#) control; however, only the [GroupBox](#) control displays a caption, and only the [Panel](#) control can have scroll bars.

In This Section

[GroupBox Control Overview](#)

Explains what this control is and its key features and properties.

[How to: Group Controls with the Windows Forms GroupBox Control](#)

Describes how to use this control to group controls.

Reference

[GroupBox](#)

Describes this class and has links to all its members.

[Panel](#)

Describes this class and has links to all its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

GroupBox Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

Windows Forms [GroupBox](#) controls are used to provide an identifiable grouping for other controls. Typically, you use group boxes to subdivide a form by function. For example, you may have an order form that specifies mailing options such as which overnight carrier to use. Grouping all options in a group box gives the user a logical visual cue, and at design time all the controls can be moved easily — when you move the single [GroupBox](#) control, all its contained controls move, too.

The group box's caption is defined by the [Text](#) property. For more information, see [How to: Set the Text Displayed by a Windows Forms Control](#).

GroupBox and Panel

The [GroupBox](#) control is similar to the [Panel](#) control; however, only the [GroupBox](#) control displays a caption, and only the [Panel](#) control can have scroll bars.

See Also

[GroupBox Control](#)

How to: Group Controls with the Windows Forms GroupBox Control

5/4/2018 • 1 min to read • [Edit Online](#)

Windows Forms [GroupBox](#) controls are used to group other controls. There are three reasons to group controls:

- To create a visual grouping of related form elements for a clear user interface.
- To create programmatic grouping (of radio buttons, for example).
- For moving the controls as a unit at design time.

To create a group of controls

1. Draw a [GroupBox](#) control on a form.
2. Add other controls to the group box, drawing each inside the group box.

If you have existing controls that you want to enclose in a group box, you can select all the controls, cut them to the Clipboard, select the [GroupBox](#) control, and then paste them into the group box. You can also drag them into the group box.

3. Set the [Text](#) property of the group box to an appropriate caption.

See Also

[GroupBox](#)

[GroupBox Control](#)

HelpProvider Component (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `HelpProvider` component is used to associate an HTML Help 1.x Help file (either a .chm file, produced with the HTML Help Workshop, or an .htm file) with your Windows-based application.

In This Section

[HelpProvider Component Overview](#)

Introduces the general concepts of the `HelpProvider` component, which lets you associate an HTML Help file with a Windows-based application.

See [Help Systems in Windows Forms Applications](#).

Reference

[HelpProvider](#)

Describes this class and has links to all its members.

[Help](#)

Describes this class and has links to all its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

Also see [Help Systems in Windows Forms Applications](#).

HelpProvider Component Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [HelpProvider](#) component is used to associate an HTML Help 1.x Help file (either a .chm file, produced with the HTML Help Workshop, or an .htm file) with your Windows application. You can provide help in a variety of ways:

- Provide context-sensitive Help for controls on Windows Forms.
- Provide context-sensitive Help on a particular dialog box or specific controls on a dialog box.
- Open a Help file to specific areas, such as the main page of a Table of Contents, the Index, or a search function.

Using the Help Provider

Adding a [HelpProvider](#) component to your Windows Form allows the other controls on the form to expose the Help properties of the [HelpProvider](#) component. This enables you to provide help for the controls on your Windows Form. You can associate a Help file with the [HelpProvider](#) component using the [HelpNamespace](#) property. You specify the type of Help provided by calling [SetHelpNavigator](#) and providing a value from the [HelpNavigator](#) enumeration for the specified control. You provide the keyword or topic for Help by calling the [SetHelpKeyword](#) method.

Optionally, to associate a specific Help string with another control, use the [SetHelpString](#) method. The string that you associate with a control using this method is displayed in a pop-up window when the user presses the F1 key while the control has focus.

If [HelpNamespace](#) has not been set, you must use [SetHelpString](#) to provide the Help text. If you have set both [HelpNamespace](#) and the Help string, Help based on [HelpNamespace](#) will take precedence.

NOTE

You may encounter problems using the relative path when specifying the path to the Help file in the [ShowHelp](#) method or [HelpNamespace](#) property of the [HelpProvider](#) control. As such, be sure to use the absolute file path to specify the Help file.

See Also

[Help Systems in Windows Forms Applications](#)

HScrollBar and VScrollBar Controls (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

Windows Forms scroll bar controls are used to provide easy navigation through a long list of items or a large amount of information by scrolling either horizontally or vertically within an application or control. Scroll bars are a common element of the Windows interface.

In This Section

[HScrollBar and VScrollBar Controls Overview](#)

Introduces the general concepts of the [HScrollBar](#) and [VScrollBar](#) controls, which allow users to scroll horizontally and vertically through large amounts of information.

Reference

[HScrollBar](#)

Describes this class and has links to all its members.

[VScrollBar](#)

Describes this class and has links to all its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

HScrollBar and VScrollBar Controls Overview (Windows Forms)

5/4/2018 • 2 min to read • [Edit Online](#)

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

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

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

Value Property

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

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

LargeChange and SmallChange Properties

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

When the user presses one of the arrow keys or clicks one of the scroll bar buttons, the [Value](#) property changes according to the value set in the [SmallChange](#) property.

See Also

[HScrollBar](#)

[VScrollBar](#)

[Additions to Windows Forms for the .NET Framework 2.0](#)

[Controls to Use on Windows Forms](#)

ImageList Component (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `ImageList` component is used to store images, which can then be displayed by controls. An image list allows you to write code for a single, consistent catalog of images.

In This Section

[ImageList Component Overview](#)

Explains what this component is and its key features and properties

[How to: Add or Remove Images with the Windows Forms ImageList Component](#)

Gives directions for adding and removing images from an image list.

Also see [How to: Add or Remove ImageList Images with the Designer](#).

Reference

[ImageList](#)

Describes this class and has links to all its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

ImageList Component Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [ImageList](#) component is used to store images, which can then be displayed by controls. An image list allows you to write code for a single, consistent catalog of images. For example, you can rotate images displayed by a [Button](#) control simply by changing the button's [ImageIndex](#) or [ImageKey](#) property. You can also associate the same image list with multiple controls. For example, if you are using both a [ListView](#) control and a [TreeView](#) control to display the same list of files, changing a file's icon in the image list will cause the new icon to appear in both views.

Using ImageList with Controls

You can use an image list with any control that has an `ImageList` property — or in the case of the [ListView](#) control, [SmallImageList](#) and [LargeImageList](#) properties. The controls that can be associated with an image list include: the [ListView](#), [TreeView](#), [ToolBar](#), [TabControl](#), [Button](#), [CheckBox](#), [RadioButton](#), and [Label](#) controls. To associate the image list with a control, set the control's `ImageList` property to the name of the [ImageList](#) component.

Key Properties

The key property of the [ImageList](#) component is [Images](#), which contains the pictures to be used by the associated control. Each individual image can be accessed by its index value or by its key. The [ColorDepth](#) property determines the number of colors that the images are rendered with. The images will all be displayed at the same size, set by the [ImageSize](#) property. Images that are larger will be scaled to fit.

If you are using Visual Studio 2005, you have access to a large library of standard images that you can use in your applications.

See Also

[ImageList](#)

[How to: Add or Remove Images with the Windows Forms ImageList Component](#)

How to: Add or Remove Images with the Windows Forms ImageList Component

5/4/2018 • 2 min to read • [Edit Online](#)

The Windows Forms [ImageList](#) component is typically populated with images before it is associated with a control. However, you can add and remove images after associating the image list with a control.

NOTE

When you remove images, verify that the [ImageIndex](#) property of any associated controls is still valid.

To add images programmatically

- Use the [Add](#) method of the image list's [Images](#) property.

In the following code example, the path set for the location of the image is the **My Documents** folder. This location is used because you can assume that most computers that are running the Windows operating system will include this folder. Choosing this location also lets users who have minimal system access levels more safely run the application. The following code example requires that you have a form with an [ImageList](#) control already added.

```
Public Sub LoadImage()
    Dim myImage As System.Drawing.Image = _
        Image.FromFile _
        (System.Environment.GetFolderPath _
        (System.Environment.SpecialFolder.Personal) _
        & "\Image.gif")
    ImageList1.Images.Add(myImage)
End Sub
```

```
public void addImage()
{
    // Be sure that you use an appropriate escape sequence (such as the
    // @) when specifying the location of the file.
    System.Drawing.Image myImage =
        Image.FromFile
        (System.Environment.GetFolderPath
        (System.Environment.SpecialFolder.Personal)
        + @"\Image.gif");
    imageList1.Images.Add(myImage);
}
```

```

public:
    void addImage()
    {
        // Replace the bold image in the following sample
        // with your own icon.
        // Be sure that you use an appropriate escape sequence (such as
        // \\) when specifying the location of the file.
        System::Drawing::Image ^ myImage =
            Image::FromFile(String::Concat(
                System::Environment::GetFolderPath(
                    System::Environment::SpecialFolder::Personal),
                "\\Image.gif"));
        imageList1->Images->Add(myImage);
    }

```

To add images with a key value.

- Use one of the [Add](#) methods of the image list's [Images](#) property that takes a key value.

In the following code example, the path set for the location of the image is the **My Documents** folder. This location is used because you can assume that most computers that are running the Windows operating system will include this folder. Choosing this location also lets users who have minimal system access levels more safely run the application. The following code example requires that you have a form with an [ImageList](#) control already added.

```

Public Sub LoadImage()
    Dim myImage As System.Drawing.Image = _
        Image.FromFile(_
            (System.Environment.GetFolderPath(_
                (System.Environment.SpecialFolder.Personal) __
                & "\Image.gif"))
        ImageList1.Images.Add("myPhoto", myImage)
End Sub

```

```

public void addImage()
{
    // Be sure that you use an appropriate escape sequence (such as the
    // @) when specifying the location of the file.
    System.Drawing.Image myImage =
        Image.FromFile
        (System.Environment.GetFolderPath
        (System.Environment.SpecialFolder.Personal)
        + @"\Image.gif");
    imageList1.Images.Add("myPhoto", myImage);
}

```

1.

To remove all images programmatically

- Use the [Remove](#) method to remove a single image

,or-

Use the [Clear](#) method to clear all images in the image list.

```

' Removes the first image in the image list
ImageList1.Images.Remove(myImage)
' Clears all images in the image list
ImageList1.Images.Clear()

```

```
// Removes the first image in the image list.  
imageList1.Images.Remove(myImage);  
// Clears all images in the image list.  
imageList1.Images.Clear();
```

To remove images by key

- Use the [RemoveByKey](#) method to remove a single image by its key.

```
' Removes the image named "myPhoto" from the list.  
ImageList1.Images.RemoveByKey("myPhoto")
```

```
// Removes the image named "myPhoto" from the list.  
imageList1.Images.RemoveByKey("myPhoto");
```

See Also

[ImageList Component](#)

[ImageList Component Overview](#)

[Images, Bitmaps, and Metafiles](#)

How to: Add or Remove ImageList Images with the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

You can add images to an [ImageList](#) component several different ways. You can add images very quickly by using the smart tag associated with the [ImageList](#), or if you are setting several other properties on the [ImageList](#), you may find it more convenient to add images with the Properties window. You can also add images by using code. For more information about how to add images with code, see [How to: Add or Remove Images with the Windows Forms ImageList Component](#). Typically you populate the [ImageList](#) component with images before it is associated with a control, but this is not required.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To add or remove images by using the Properties window

1. Select the [ImageList](#) component, or add one to the form.
2. In the Properties window, click the ellipsis button (...) next to the [Images](#) property.
3. In the **Image Collection Editor**, click **Add** or **Remove** to add or remove images from the list.

To add or remove images using the smart tag

1. Select the [ImageList](#) component, or add one to the form.
2. Click the smart tag glyph (□)
3. In the **ImageList Tasks** dialog box, select **Choose Images**.
4. In the **Images Collection Editor** click **Add** or **Remove** to add or remove images from the list.

See Also

[Images, Bitmaps, and Metafiles](#)

[Walkthrough: Performing Common Tasks Using Smart Tags on Windows Forms Controls](#)

[ImageList Component](#)

Label Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

IMPORTANT

The [ToolStripLabel](#) control replaces and adds functionality to the [Label](#) control. You can use the [ToolStripLabel](#) with other new controls such as the [ToolStripDropDown](#). However, the [Label](#) control is retained for both backward compatibility and future use, if you choose.

Windows Forms [Label](#) controls are used to display text or images that cannot be edited by the user. They are used to identify objects on a form—to provide a description of what a certain control will do if clicked, for example, or to display information in response to a run-time event or process in your application. Because the [Label](#) control cannot receive focus, it can also be used to create access keys for other controls.

In This Section

[Label Control Overview](#)

Explains what this control is and its key features and properties.

[How to: Create Access Keys with Windows Forms Label Controls](#)

Describes how to use a label to define an access key for another control.

[How to: Size a Windows Forms Label Control to Fit Its Contents](#)

Explains adjusting the size of a label control for its caption.

Reference

[Label](#)

Describes this class and has links to all its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

Label Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

Windows Forms [Label](#) controls are used to display text or images that cannot be edited by the user. They are used to identify objects on a form — to provide a description of what a certain control will do if clicked, for example, or to display information in response to a run-time event or process in your application. For example, you can use labels to add descriptive captions to text boxes, list boxes, combo boxes, and so on. You can also write code that changes the text displayed by a label in response to events at run time. For example, if your application takes a few minutes to process a change, you can display a processing-status message in a label.

Working with the Label Control

Because the [Label](#) control cannot receive the focus, it can also be used to create access keys for other controls. An access key allows a user to select the other control by pressing the ALT key with the access key. For more information, see [Creating Access Keys for Windows Forms Controls](#) and [How to: Create Access Keys with Windows Forms Label Controls](#).

The caption displayed in the label is contained in the [Text](#) property. The [TextAlign](#) property allows you to set the alignment of the text within the label. For more information, see [How to: Set the Text Displayed by a Windows Forms Control](#).

See Also

[Label](#)

[How to: Size a Windows Forms Label Control to Fit Its Contents](#)

[How to: Create Access Keys with Windows Forms Label Controls](#)

How to: Create Access Keys with Windows Forms Label Controls

5/4/2018 • 1 min to read • [Edit Online](#)

Windows Forms [Label](#) controls can be used to define access keys for other controls. When you define an access key in a label control, the user can press the ALT key plus the character you designate to move the focus to the control that follows it in the tab order. Because labels cannot receive focus, focus automatically moves to the next control in the tab order. Use this technique to assign access keys to text boxes, combo boxes, list boxes, and data grids.

To assign an access key to a control with a label

1. Draw the label first, and then draw the other control.

-or-

Draw the controls in any order and set the [TabIndex](#) property of the label to one less than the other control.

2. Set the label's [UseMnemonic](#) property to `true`.
3. Use an ampersand (&) in the label's [Text](#) property to assign the access key for the label. For more information, see [Creating Access Keys for Windows Forms Controls](#).

NOTE

You may want to display ampersands in a label control, rather than use them to create access keys. This may occur if you bind a label control to a field in a recordset where the data includes ampersands. To display ampersands in a label control, set the [UseMnemonic](#) property to `false`. If you wish to display ampersands and also have an access key, set the [UseMnemonic](#) property to `true` and indicate the access key with one ampersand (&) and the ampersand to display with two ampersands.

```
Label1.UseMnemonic = True  
Label1.Text = "&Print"  
Label2.UseMnemonic = True  
Label2.Text = "&Copy && Paste"
```

```
label1.UseMnemonic = true;  
label1.Text = "&Print";  
label2.UseMnemonic = true;  
label2.Text = "&Copy && Paste";
```

```
label1->UseMnemonic = true;  
label1->Text = "&Print";  
label2->UseMnemonic = true;  
label2->Text = "&Copy && Paste";
```

See Also

[How to: Size a Windows Forms Label Control to Fit Its Contents](#)
[Label Control Overview](#)

Label Control

How to: Size a Windows Forms Label Control to Fit Its Contents

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [Label](#) control can be single-line or multi-line, and it can be either fixed in size or can automatically resize itself to accommodate its caption. The [AutoSize](#) property helps you size the controls to fit larger or smaller captions, which is particularly useful if the caption will change at run time.

To make a label control resize dynamically to fit its contents

1. Set its [AutoSize](#) property to `true`.

If [AutoSize](#) is set to `false`, the words specified in the [Text](#) property will wrap to the next line if possible, but the control will not grow.

See Also

- [How to: Create Access Keys with Windows Forms Label Controls](#)
- [Label Control Overview](#)
- [Label Control](#)

LinkLabel Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `LinkLabel` control enables you to add Web-style links to Windows Forms applications. You can use the `LinkLabel` control for everything that you can use the `Label` control for; you also can set part of the text as a link to an object or Web page.

In This Section

[LinkLabel Control Overview](#)

Explains what this control is and its key features and properties.

[How to: Change the Appearance of the Windows Forms LinkLabel Control](#)

Lists ways to specify the color and underlining of Web-style links in link labels.

[How to: Link to an Object or Web Page with the Windows Forms LinkLabel Control](#)

Provides how-to instructions for opening a form or Web page when a link is clicked.

[How to: Display a Web Page from a Windows Forms LinkLabel Control \(Visual Basic\)](#)

Shows how to display a Web page in the default browser when a user clicks a Windows Forms `LinkLabel` control.

Reference

[LinkLabel class](#)

Describes this class and has links to all its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information about their use.

LinkLabel Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [LinkLabel](#) control allows you to add Web-style links to Windows Forms applications. You can use the [LinkLabel](#) control for everything that you can use the [Label](#) control for; you also can set part of the text as a link to a file, folder, or Web page.

What You Can Do with the LinkLabel Control

In addition to all the properties, methods, and events of the [Label](#) control, the [LinkLabel](#) control has properties for hyperlinks and link colors. The [LinkArea](#) property sets the area of the text that activates the link. The [LinkColor](#), [VisitedLinkColor](#), and [ActiveLinkColor](#) properties set the colors of the link. The [LinkClicked](#) event determines what happens when the link text is selected.

The simplest use of the [LinkLabel](#) control is to display a single link using the [LinkArea](#) property, but you can also display multiple hyperlinks using the [Links](#) property. The [Links](#) property enables you to access a collection of links. You can also specify data in the [LinkData](#) property of each individual [LinkLabel.Link](#) object. The value of the [LinkData](#) property can be used to store the location of a file to display or the address of a Web site.

See Also

[LinkLabel](#)

[Label Control Overview](#)

[How to: Link to an Object or Web Page with the Windows Forms LinkLabel Control](#)

[How to: Change the Appearance of the Windows Forms LinkLabel Control](#)

How to: Change the Appearance of the Windows Forms LinkLabel Control

5/4/2018 • 2 min to read • [Edit Online](#)

You can change the text displayed by the [LinkLabel](#) control to suit a variety of purposes. For example, it is common practice to indicate to the user that text can be clicked by setting the text to appear in a specific color with an underline. After the user clicks the text, the color changes to a different color. To control this behavior, you can set five different properties: the [LinkBehavior](#), [LinkArea](#), [LinkColor](#), [VisitedLinkColor](#), and [LinkVisited](#) properties.

To change the appearance of a LinkLabel control

1. Set the [LinkColor](#) and [VisitedLinkColor](#) properties to the colors you want.

This can be done either programmatically or at design time in the **Properties** window.

```
' You can set the color using decimal values for red, green, and blue  
LinkLabel1.LinkColor = Color.FromArgb(0, 0, 255)  
' Or you can set the color using defined constants  
LinkLabel1.VisitedLinkColor = Color.Purple
```

```
// You can set the color using decimal values for red, green, and blue  
linkLabel1.LinkColor = Color.FromArgb(0, 0, 255);  
// Or you can set the color using defined constants  
linkLabel1.VisitedLinkColor = Color.Purple;
```

```
// You can set the color using decimal values for red, green, and blue  
linkLabel1->LinkColor = Color::FromArgb(0, 0, 255);  
// Or you can set the color using defined constants  
linkLabel1->VisitedLinkColor = Color::Purple;
```

2. Set the [Text](#) property to an appropriate caption.

This can be done either programmatically or at design time in the **Properties** window.

```
LinkLabel1.Text = "Click here to see more."
```

```
linkLabel1.Text = "Click here to see more.;"
```

```
linkLabel1->Text = "Click here to see more.;"
```

3. Set the [LinkArea](#) property to determine which part of the caption will be indicated as a link.

The [LinkArea](#) value is represented with a [LinkArea](#) containing two numbers, the starting character position and the number of characters. This can be done either programmatically or at design time in the **Properties** window.

```
LinkLabel1.LinkArea = new LinkArea(6,4)
```

```
linkLabel1.LinkArea = new LinkArea(6,4);
```

```
linkLabel1->LinkArea = LinkArea(6,4);
```

4. Set the [LinkBehavior](#) property to [AlwaysUnderline](#), [HoverUnderline](#), or [NeverUnderline](#).

If it is set to [HoverUnderline](#), the part of the caption determined by [LinkArea](#) will only be underlined when the pointer rests on it.

5. In the [LinkClicked](#) event handler, set the [LinkVisited](#) property to `true`.

When a link has been visited, it is common practice to change its appearance in some way, usually by color. The text will change to the color specified by the [VisitedLinkColor](#) property.

```
Protected Sub LinkLabel1_LinkClicked (ByVal sender As Object, _  
    ByVal e As EventArgs) Handles LinkLabel1.LinkClicked  
    ' Change the color of the link text  
    ' by setting LinkVisited to True.  
    LinkLabel1.LinkVisited = True  
    ' Then do whatever other action is appropriate  
End Sub
```

```
protected void LinkLabel1_LinkClicked(object sender, System.EventArgs e)  
{  
    // Change the color of the link text by setting LinkVisited  
    // to True.  
    linkLabel1.LinkVisited = true;  
    // Then do whatever other action is appropriate  
}
```

```
private:  
    System::Void linkLabel1_LinkClicked(System::Object ^ sender,  
        System::Windows::Forms::LinkLabelLinkClickedEventArgs ^ e)  
{  
    // Change the color of the link text by setting LinkVisited  
    // to True.  
    linkLabel1->LinkVisited = true;  
    // Then do whatever other action is appropriate  
}
```

See Also

[LinkArea](#)

[LinkColor](#)

[VisitedLinkColor](#)

[LinkVisited](#)

[LinkLabel Control Overview](#)

[How to: Link to an Object or Web Page with the Windows Forms LinkLabel Control](#)

[LinkLabel Control](#)

How to: Link to an Object or Web Page with the Windows Forms LinkLabel Control

5/4/2018 • 3 min to read • [Edit Online](#)

The Windows Forms [LinkLabel](#) control allows you to create Web-style links on your form. When the link is clicked, you can change its color to indicate the link has been visited. For more information on changing the color, see [How to: Change the Appearance of the Windows Forms LinkLabel Control](#).

Linking to Another Form

To link to another form with a LinkLabel control

1. Set the [Text](#) property to an appropriate caption.
2. Set the [LinkArea](#) property to determine which part of the caption will be indicated as a link. How it is indicated depends on the appearance-related properties of the link label. The [LinkArea](#) value is represented by a [LinkArea](#) object containing two numbers, the starting character position and the number of characters. The [LinkArea](#) property can be set in the Properties window or in code in a manner similar to the following:

```
' In this code example, the link area has been set to begin
' at the first character and extend for eight characters.
' You may need to modify this based on the text entered in Step 1.
LinkLabel1.LinkArea = New LinkArea(0, 8)
```

```
// In this code example, the link area has been set to begin
// at the first character and extend for eight characters.
// You may need to modify this based on the text entered in Step 1.
linkLabel1.LinkArea = new LinkArea(0,8);
```

```
// In this code example, the link area has been set to begin
// at the first character and extend for eight characters.
// You may need to modify this based on the text entered in Step 1.
linkLabel1->LinkArea = LinkArea(0,8);
```

3. In the [LinkClicked](#) event handler, invoke the [Show](#) method to open another form in the project, and set the [LinkVisited](#) property to `true`.

NOTE

An instance of the [LinkLabelLinkClickedEventArgs](#) class carries a reference to the [LinkLabel](#) control that was clicked, so there is no need to cast the `sender` object.

```
Protected Sub LinkLabel1_LinkClicked(ByVal Sender As System.Object, _
    ByVal e As System.Windows.Forms.LinkLabelLinkClickedEventArgs) _
Handles LinkLabel1.LinkClicked
    ' Show another form.
    Dim f2 As New Form()
    f2.Show
    LinkLabel1.LinkVisited = True
End Sub
```

```

protected void.linkLabel1_LinkClicked(object sender, System.
Windows.Forms.LinkLabelLinkClickedEventArgs e)
{
    // Show another form.
    Form f2 = new Form();
    f2.Show();
    linkLabel1.LinkVisited = true;
}

```

```

private:
void linkLabel1_LinkClicked(System::Object ^ sender,
    System::Windows::Forms::LinkLabelLinkClickedEventArgs ^ e)
{
    // Show another form.
    Form ^ f2 = new Form();
    f2->Show();
    linkLabel1->LinkVisited = true;
}

```

Linking to a Web Page

The [LinkLabel](#) control can also be used to display a Web page with the default browser.

To start Internet Explorer and link to a Web page with a LinkLabel control

1. Set the [Text](#) property to an appropriate caption.
2. Set the [LinkArea](#) property to determine which part of the caption will be indicated as a link.
3. In the [LinkClicked](#) event handler, in the midst of an exception-handling block, call a second procedure that sets the [LinkVisited](#) property to `true` and uses the [Start](#) method to start the default browser with a URL.

To use the [Start](#) method you need to add a reference to the [System.Diagnostics](#) namespace.

IMPORTANT

If the code below is run in a partial-trust environment (such as on a shared drive), the JIT compiler fails when the `VisitLink` method is called. The `System.Diagnostics.Process.Start` statement causes a link demand that fails. By catching the exception when the `VisitLink` method is called, the code below ensures that if the JIT compiler fails, the error is handled gracefully.

```

Private Sub LinkLabel1_LinkClicked(ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.LinkLabelLinkClickedEventArgs) _
Handles LinkLabel1.LinkClicked
Try
    VisitLink()
Catch ex As Exception
    ' The error message
    MessageBox.Show("Unable to open link that was clicked.")
End Try
End Sub

Sub VisitLink()
    ' Change the color of the link text by setting LinkVisited
    ' to True.
    LinkLabel1.LinkVisited = True
    ' Call the Process.Start method to open the default browser
    ' with a URL:
    System.Diagnostics.Process.Start("http://www.microsoft.com")
End Sub

```

```

private void linkLabel1_LinkClicked(object sender, System.Windows.Forms.LinkLabelLinkClickedEventArgs e)
{
    try
    {
        VisitLink();
    }
    catch (Exception ex )
    {
        MessageBox.Show("Unable to open link that was clicked.");
    }
}

private void VisitLink()
{
    // Change the color of the link text by setting LinkVisited
    // to true.
    linkLabel1.LinkVisited = true;
    //Call the Process.Start method to open the default browser
    //with a URL:
    System.Diagnostics.Process.Start("http://www.microsoft.com");
}

```

```

private:
    void linkLabel1_LinkClicked(System::Object ^  sender,
        System::Windows::Forms::LinkLabelLinkClickedEventArgs ^  e)
    {
        try
        {
            VisitLink();
        }
        catch (Exception ^ ex)
        {
            MessageBox::Show("Unable to open link that was clicked.");
        }
    }
private:
    void VisitLink()
    {
        // Change the color of the link text by setting LinkVisited
        // to true.
        linkLabel1->LinkVisited = true;
        // Call the Process.Start method to open the default browser
        // with a URL:
        System::Diagnostics::Process::Start("http://www.microsoft.com");
    }

```

See Also

[Process.Start](#)

[LinkLabel Control Overview](#)

[How to: Change the Appearance of the Windows Forms LinkLabel Control](#)

[LinkLabel Control](#)

How to: Display a Web Page from a Windows Forms LinkLabel Control (Visual Basic)

5/4/2018 • 1 min to read • [Edit Online](#)

This example displays a Web page in the default browser when a user clicks a Windows Forms [LinkLabel](#) control.

Example

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e _
As System.EventArgs) Handles MyBase.Load
    LinkLabel1.Text = "Click here to get more info."
    LinkLabel1.Links.Add(6, 4, "www.microsoft.com")
End Sub

Private Sub LinkLabel1_LinkClicked(ByVal sender As System.Object, ByVal _
e As System.Windows.Forms.LinkLabelLinkClickedEventArgs) Handles _
LinkLabel1.LinkClicked
    System.Diagnostics.Process.Start(e.Link.LinkData.ToString())
End Sub
```

Compiling the Code

This example requires:

- A Windows Form named `Form1`.
- A [LinkLabel](#) control named `LinkLabel1`.
- An active Internet connection.

.NET Framework Security

The call to the [Start](#) method requires full trust. For more information, see [SecurityException](#).

See Also

[LinkLabel](#)

[LinkLabel Control](#)

ListBox Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

A Windows Forms `ListBox` control displays a list of items from which the user can select one or more.

In This Section

[ListBox Control Overview](#)

Explains what this control is and its key features and properties.

Reference

[ListBox class](#)

Describes this class and has links to all its members.

Related Sections

[Windows Forms Controls Used to List Options](#)

Provides a list of things you can do with list boxes, combo boxes, and checked list boxes.

ListBox Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

A Windows Forms [ListBox](#) control displays a list from which the user can select one or more items. If the total number of items exceeds the number that can be displayed, a scroll bar is automatically added to the [ListBox](#) control. When the [MultiColumn](#) property is set to `true`, the list box displays items in multiple columns and a horizontal scroll bar appears. When the [MultiColumn](#) property is set to `false`, the list box displays items in a single column and a vertical scroll bar appears. When [ScrollAlwaysVisible](#) is set to `true`, the scroll bar appears regardless of the number of items. The [SelectionMode](#) property determines how many list items can be selected at a time.

Ways to Change the ListBox Control

The [SelectedIndex](#) property returns an integer value that corresponds to the first selected item in the list box. You can programmatically change the selected item by changing the [SelectedIndex](#) value in code; the corresponding item in the list will appear highlighted on the Windows Form. If no item is selected, the [SelectedIndex](#) value is -1. If the first item in the list is selected, the [SelectedIndex](#) value is 0. When multiple items are selected, the [SelectedIndex](#) value reflects the selected item that appears first in the list. The [SelectedItem](#) property is similar to [SelectedIndex](#), but returns the item itself, usually a string value. The [Count](#) property reflects the number of items in the list, and the value of the [Count](#) property is always one more than the largest possible [SelectedIndex](#) value because [SelectedIndex](#) is zero-based.

To add or delete items in a [ListBox](#) control, use the [Add](#), [Insert](#), [Clear](#) or [Remove](#) method. Alternatively, you can add items to the list by using the [Items](#) property at design time.

See Also

[ListBox](#)

[How to: Add and Remove Items from a Windows Forms ComboBox, ListBox, or CheckedListBox Control](#)

[How to: Sort the Contents of a Windows Forms ComboBox, ListBox, or CheckedListBox Control](#)

[How to: Bind a Windows Forms ComboBox or ListBox Control to Data](#)

[ComboBox Control Overview](#)

[CheckedListBox Control Overview](#)

[Windows Forms Controls Used to List Options](#)

[How to: Create a Lookup Table for a Windows Forms ComboBox, ListBox, or CheckedListBox Control](#)

ListView Control (Windows Forms)

5/4/2018 • 2 min to read • [Edit Online](#)

The Windows Forms `ListView` control displays a list of items with icons. You can use a list view to create a user interface like the right pane of Windows Explorer.

In This Section

[ListView Control Overview](#)

Describes this control and its key features and properties.

[How to: Add and Remove Items with the Windows Forms ListView Control](#)

Describes how to add or remove items from a list view.

[How to: Add Columns to the Windows Forms ListView Control](#)

Describes how to create columns in order to display information about each list item.

[How to: Display Icons for the Windows Forms ListView Control](#)

Describes how to associate a list view with an appropriate image list for displaying large or small icons.

[How to: Display Subitems in Columns with the Windows Forms ListView Control](#)

Describes how to display information about each list item in columns.

[How to: Select an Item in the Windows Forms ListView Control](#)

Describes how to programmatically select an item.

[How to: Group Items in a Windows Forms ListView Control](#)

Describes how to create groups for categorized display and how to assign items to each group.

This feature is available only for Windows XP Home Edition, Windows XP Professional, Windows Server 2003.

[How to: Enable Tile View in a Windows Forms ListView Control](#)

Describes how to display items as tiles, each of which is comprised of a large icon and multiple lines of text.

This feature is available only for Windows XP Home Edition, Windows XP Professional, Windows Server 2003.

[How to: Display an Insertion Mark in a Windows Forms ListView Control](#)

Describes how to implement user-feedback for drag-and-drop operations in which an insertion mark indicates the drop location for each mouse-pointer position.

This feature is available only for Windows XP Home Edition, Windows XP Professional, Windows Server 2003.

[How to: Add Search Capabilities to a ListView Control](#)

Describes how to programmatically find an item using either text search or screen coordinates.

- [How to: Enable Tile View in a Windows Forms ListView Control Using the Designer](#)
- [How to: Add and Remove Items with the Windows Forms ListView Control Using the Designer](#)
- [How to: Add Columns to the Windows Forms ListView Control Using the Designer](#)
- [How to: Group Items in a Windows Forms ListView Control Using the Designer](#)
- [Walkthrough: Creating an Explorer Style Interface with the ListView and TreeView Controls Using the Designer](#)

Reference

[ListView class](#)

Describes this class and has links to all its members.

Related Sections

[How to: Add Custom Information to a TreeView or ListView Control \(Windows Forms\)](#)

Describes how to inherit from an item in a list view or a node in a tree view in order to add any fields, methods, or constructors you need.

[ImageList Component](#)

Explains what an image list is and its key features and properties.

[How to: Create a Multipane User Interface with Windows Forms](#)

Gives instructions for laying out a Windows Form with multiple panes.

[Windows XP Features and Windows Forms Controls](#)

Explains how to take advantage of Windows XP-specific features that apply to the [ListView](#) control.

See Also

[Controls to Use on Windows Forms](#)

ListView Control Overview (Windows Forms)

5/4/2018 • 2 min to read • [Edit Online](#)

The Windows Forms [ListView](#) control displays a list of items with icons. You can use a list view to create a user interface like the right pane of Windows Explorer. The control has four view modes: LargeIcon, SmallIcon, List, and Details.

What You Can Do with the ListView Control

NOTE

An additional view mode, Tile, is only available on Windows XP and the Windows Server 2003 operating system. For more information, see [How to: Enable Tile View in a Windows Forms ListView Control](#).

The LargeIcon mode displays large icons next to the item text; the items appear in multiple columns if the control is large enough. The SmallIcon mode is the same except that it displays small icons. The List mode displays small icons but is always in a single column. The Details mode displays items in multiple columns. For details, see [How to: Add Columns to the Windows Forms ListView Control](#). The view mode is determined by the [View](#) property. All of the view modes can display images from image lists. For details, see [How to: Display Icons for the Windows Forms ListView Control](#).

The following table lists some of the [ListView](#) members and the views they are valid in.

LISTVIEW MEMBER	VIEW
Alignment property	SmallIcon or LargeIcon
AutoArrange property	SmallIcon or LargeIcon
AutoResizeColumn method	Details
Columns property	Details or Tile
DrawSubItem event	Details
FindItemWithText method	Details, List, or Tile
FindNearestItem method	SmallIcon or LargeIcon
GetItemAt method	Details or Tile
Groups property	All views except List
HeaderStyle property	Details.
InsertionMark property	LargeIcon, SmallIcon, or Tile

The key property of the [ListView](#) control is [Items](#), which contains the items displayed by the control. The [SelectedItems](#) property contains a collection of the items currently selected in the control. The user can select

multiple items, for example to drag and drop several items at a time to another control, if the [MultiSelect](#) property is set to `true`. The [ListView](#) control can display check boxes next to the items, if the [Checkboxes](#) property is set to `true`.

The [Activation](#) property determines what type of action the user must take to activate an item in the list: the options are [Standard](#), [OneClick](#), and [TwoClick](#). [OneClick](#) activation requires a single click to activate the item. [TwoClick](#) activation requires the user to double-click to activate the item; a single click changes the color of the item text. [Standard](#) activation requires the user to double-click to activate an item, but the item does not change appearance.

The [ListView](#) control also supports the visual styles and other features available on the Windows XP platform, including grouping, tile view, and insertion marks. For more information, see [Windows XP Features and Windows Forms Controls](#).

See Also

[ListView](#)

[ListView Control](#)

[How to: Add and Remove Items with the Windows Forms ListView Control](#)

[How to: Add Columns to the Windows Forms ListView Control](#)

[How to: Display Icons for the Windows Forms ListView Control](#)

[How to: Display Subitems in Columns with the Windows Forms ListView Control](#)

[How to: Select an Item in the Windows Forms ListView Control](#)

[How to: Group Items in a Windows Forms ListView Control](#)

[How to: Display an Insertion Mark in a Windows Forms ListView Control](#)

[How to: Add Search Capabilities to a ListView Control](#)

[How to: Add Custom Information to a TreeView or ListView Control \(Windows Forms\)](#)

[How to: Create a Multipane User Interface with Windows Forms](#)

How to: Add and Remove Items with the Windows Forms ListView Control

5/4/2018 • 1 min to read • [Edit Online](#)

The process of adding an item to a Windows Forms [ListView](#) control consists primarily of specifying the item and assigning properties to it. Adding or removing list items can be done at any time.

To add items programmatically

1. Use the [Add](#) method of the [Items](#) property.

```
// Adds a new item with ImageIndex 3
listView1.Items.Add("List item text", 3);
```

```
' Adds a new item with ImageIndex 3
ListView1.Items.Add("List item text", 3)
```

To remove items programmatically

1. Use the [RemoveAt](#) or [Clear](#) method of the [Items](#) property. The [RemoveAt](#) method removes a single item; the [Clear](#) method removes all items from the list.

```
// Removes the first item in the list.
listView1.Items.RemoveAt(0);
// Clears all the items.
listView1.Items.Clear();
```

```
' Removes the first item in the list.
ListView1.Items.RemoveAt(0)
' Clears all items:
ListView1.Items.Clear()
```

See Also

[ListView](#)

[ListView Control](#)

[ListView Control Overview](#)

How to: Add and Remove Items with the Windows Forms ListView Control Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

The process of adding an item to a Windows Forms [ListView](#) control consists primarily of specifying the item and assigning properties to it. Adding or removing list items can be done at any time.

The following procedure requires a **Windows Application** project with a form containing a [ListView](#) control. For information about setting up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To add or remove items using the designer

1. Select the [ListView](#) control.
2. In the **Properties** window, click the **Ellipsis** (...) button next to the [Items](#) property.

The **ListViewItem Collection Editor** appears.

3. To add an item, click the **Add** button. You can then set properties of the new item, such as the [Text](#) and [ImageIndex](#) properties.
4. To remove an item, select it and click the **Remove** button.

See Also

[ListView Control Overview](#)

[How to: Add Columns to the Windows Forms ListView Control](#)

[How to: Display Subitems in Columns with the Windows Forms ListView Control](#)

[How to: Display Icons for the Windows Forms ListView Control](#)

[How to: Add Custom Information to a TreeView or ListView Control \(Windows Forms\)](#)

[How to: Group Items in a Windows Forms ListView Control](#)

How to: Add Columns to the Windows Forms ListView Control

5/4/2018 • 1 min to read • [Edit Online](#)

In the Details view, the [ListView](#) control can display multiple columns for each list item. You can use the columns to display to the user several types of information about each list item. For example, a list of files could display the file name, file type, size, and date the file was last modified. For information about populating the columns after they are created, see [How to: Display Subitems in Columns with the Windows Forms ListView Control](#).

To add columns programmatically

1. Set the control's [View](#) property to [Details](#).
2. Use the [Add](#) method of the list view's [Columns](#) property.

```
// Set to details view.  
listView1.View = View.Details;  
// Add a column with width 20 and left alignment.  
listView1.Columns.Add("File type", 20, HorizontalAlignment.Left);
```

```
' Set to details view  
ListView1.View = View.Details  
' Add a column with width 20 and left alignment  
ListView1.Columns.Add("File type", 20, HorizontalAlignment.Left)
```

See Also

- [ListView](#)
- [ListView Control](#)
- [ListView Control Overview](#)

How to: Add Columns to the Windows Forms ListView Control Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [ListView](#) control can display multiple columns for each list item when in the **Details** view. You can use the columns to display several types of information about each list item. For example, a list of files could display the file name, file type, size, and date the file was last modified. For information on populating the columns once they are created, see [How to: Display Subitems in Columns with the Windows Forms ListView Control](#).

The following procedure requires a **Windows Application** project with a form containing a [ListView](#) control. For information about setting up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To add columns in the designer

1. In the **Properties** window, set the control's [View](#) property to **Details**.
2. In the **Properties** window, click the **Ellipsis** button (next to the [Columns](#) property.

The **ColumnHeader Collection Editor** appears.

3. Use the **Add** button to add new columns. You can then select the column header and set its text (the caption of the column), text alignment, and width.

See Also

[ListView Control Overview](#)

[How to: Add and Remove Items with the Windows Forms ListView Control](#)

[How to: Display Subitems in Columns with the Windows Forms ListView Control](#)

[How to: Display Icons for the Windows Forms ListView Control](#)

[How to: Add Custom Information to a TreeView or ListView Control \(Windows Forms\)](#)

How to: Add Search Capabilities to a ListView Control

5/4/2018 • 5 min to read • [Edit Online](#)

Oftentimes when working with a large list of items in a [ListView](#) control, you want to offer search capabilities to the user. The [ListView](#) control offers this capability in two different ways: text matching and location searching.

The [FindItemWithText](#) method allows you to perform a text search on a [ListView](#) in list or details view, given a search string and an optional starting and ending index. In contrast, the [FindNearestItem](#) method allows you to find an item in a [ListView](#) in icon or tile view, given a set of x- and y-coordinates and a direction to search.

To find an item using text

1. Create a [ListView](#) with the [View](#) property set to [Details](#) or [List](#), and then populate the [ListView](#) with items.
2. Call the [FindItemWithText](#) method, passing the text of the item you would like to find.
3. The following code example demonstrates how to create a basic [ListView](#), populate it with items, and use text input from the user to find an item in the list.

```

private:
    ListView^ textListView;
    TextBox^ searchBox;

private:
    void InitializeTextSearchListView()
    {
        textListView = gcnew ListView();
        searchBox = gcnew TextBox();
        searchBox->Location = Point(150, 20);
        textListView->Scrollable = true;
        textListView->Width = 100;

        // Set the View to list to use the FindItemWithText method.
        textListView->View = View::List;

        // Populate the ListViewWithItems
        textListView->Items->AddRange(gcnew array<ListViewItem^>{
            gcnew ListViewItem("Amy Alberts"),
            gcnew ListViewItem("Amy Recker"),
            gcnew ListViewItem("Erin Hagens"),
            gcnew ListViewItem("Barry Johnson"),
            gcnew ListViewItem("Jay Hamlin"),
            gcnew ListViewItem("Brian Valentine"),
            gcnew ListViewItem("Brian Welker"),
            gcnew ListViewItem("Daniel Weisman") });

        // Handle the TextChanged to get the text for our search.
        searchBox->TextChanged += gcnew EventHandler(this,
            &Form1::searchBox_TextChanged);

        // Add the controls to the form.
        this->Controls->Add(textListView);
        this->Controls->Add(searchBox);
    }

private:
    void searchBox_TextChanged(Object^ sender, EventArgs^ e)
    {
        // Call FindItemWithText with the contents of the textbox.
        ListViewItem^ foundItem =
            textListView->FindItemWithText(searchBox->Text, false, 0, true);
        if (foundItem != nullptr)
        {
            textListView->TopItem = foundItem;
        }
    }
}

```

```
private ListView textListView = new ListView();
private TextBox searchBox = new TextBox();
private void InitializeTextSearchListView()
{
    searchBox.Location = new Point(10, 60);
    textListView.Scrollable = true;
    textListView.Width = 80;
    textListView.Height = 50;

    // Set the View to list to use the FindItemWithText method.
    textListView.View = View.List;

    // Populate the ListViewWithItems
    textListView.Items.AddRange(new ListViewItem[]{
        new ListViewItem("Amy Alberts"),
        new ListViewItem("Amy Recker"),
        new ListViewItem("Erin Hagens"),
        new ListViewItem("Barry Johnson"),
        new ListViewItem("Jay Hamlin"),
        new ListViewItem("Brian Valentine"),
        new ListViewItem("Brian Welker"),
        new ListViewItem("Daniel Weisman") });

    // Handle the TextChanged to get the text for our search.
    searchBox.TextChanged += new EventHandler(searchBox_TextChanged);

    // Add the controls to the form.
    this.Controls.Add(textListView);
    this.Controls.Add(searchBox);
}

private void searchBox_TextChanged(object sender, EventArgs e)
{
    // Call FindItemWithText with the contents of the textbox.
    ListViewItem foundItem =
        textListView.FindItemWithText(searchBox.Text, false, 0, true);
    if (foundItem != null)
    {
        textListView.TopItem = foundItem;
    }
}
```

```

Private textListView As New ListView()
Private WithEvents searchBox As New TextBox()

Private Sub InitializeTextSearchListView()
    searchBox.Location = New Point(150, 20)
    textListView.Scrollable = True
    textListView.Width = 80
    textListView.Height = 50

    ' Set the View to list to use the FindItemWithText method.
    textListView.View = View.List

    ' Populate the ListView with items.
    textListView.Items.AddRange(New ListViewItem() { _
        New ListViewItem("Amy Alberts"), _
        New ListViewItem("Amy Recker"), _
        New ListViewItem("Erin Hagens"), _
        New ListViewItem("Barry Johnson"), _
        New ListViewItem("Jay Hamlin"), _
        New ListViewItem("Brian Valentine"), _
        New ListViewItem("Brian Welker"), _
        New ListViewItem("Daniel Weisman")})

    ' Add the controls to the form.
    Me.Controls.Add(textListView)
    Me.Controls.Add(searchBox)

End Sub

Private Sub searchBox_TextChanged(ByVal sender As Object, ByVal e As EventArgs) _
    Handles searchBox.TextChanged

    ' Call FindItemWithText with the contents of the textbox.
    Dim foundItem As ListViewItem = _
        textListView.FindItemWithText(searchBox.Text, False, 0, True)

    If (foundItem IsNot Nothing) Then
        textListView.TopItem = foundItem
    End If

End Sub

```

To find an item using x- and y-coordinates

1. Create a [ListView](#) with the [View](#) property set to [SmallIcon](#) or [LargeIcon](#), and then populate the [ListView](#) with items.
2. Call the [FindNearestItem](#) method, passing the desired x- and y-coordinates and the direction you would like to search.
3. The following code example demonstrates how to create a basic icon [ListView](#), populate it with items, and capture the [MouseDown](#) event to find the nearest item in the up direction.

```

ListView^ iconListView;
TextBox^ previousItemBox;

private:
    void InitializeLocationSearchListView()
    {
        previousItemBox = gcnew TextBox();
        iconListView = gcnew ListView();
        previousItemBox->Location = Point(150, 20);

        // Create an image list for the icon ListView.
        iconListView->SmallImageList = gcnew ImageList();

        // Add an image to the ListView small icon list.
        iconListView->SmallImageList->Images->Add(
            gcnew Bitmap(Control::typeid, "Edit.bmp"));

        // Set the view to small icon and add some items with the image
        // in the image list.
        iconListView->View = View::SmallIcon;
        iconListView->Items->AddRange(gcnew array<ListViewItem^>{
            gcnew ListViewItem("Amy Alberts", 0),
            gcnew ListViewItem("Amy Recker", 0),
            gcnew ListViewItem("Erin Hagens", 0),
            gcnew ListViewItem("Barry Johnson", 0),
            gcnew ListViewItem("Jay Hamlin", 0),
            gcnew ListViewItem("Brian Valentine", 0),
            gcnew ListViewItem("Brian Welker", 0),
            gcnew ListViewItem("Daniel Weisman", 0) });
        this->Controls->Add(iconListView);
        this->Controls->Add(previousItemBox);

        // Handle the MouseDown event to capture user input.
        iconListView->MouseDown += gcnew MouseEventHandler(
            this, &Form1::iconListView_MouseDown);
    }

void iconListView_MouseDown(Object^ sender, MouseEventArgs^ e)
{
    // Find the next item up from where the user clicked.
    ListViewItem^ foundItem = iconListView->FindNearestItem(
        SearchDirectionHint::Up, e->X, e->Y);

    // Display the results in a textbox..
    if (foundItem != nullptr)
    {
        previousItemBox->Text = foundItem->Text;
    }
    else
    {
        previousItemBox->Text = "No item found";
    }
}

```

```

ListView iconListView = new ListView();
TextBox previousItemBox = new TextBox();

private void InitializeLocationSearchListView()
{
    previousItemBox.Location = new Point(150, 20);

    // Create an image list for the icon ListView.
    iconListView.LargeImageList = new ImageList();
    iconListView.Height = 400;

    // Add an image to the ListView large icon list.
    iconListView.LargeImageList.Images.Add(
        new Bitmap(typeof(Control), "Edit.bmp"));

    // Set the view to large icon and add some items with the image
    // in the image list.
    iconListView.View = View.LargeIcon;
    iconListView.Items.AddRange(new ListViewItem[]{
        new ListViewItem("Amy Alberts", 0),
        new ListViewItem("Amy Recker", 0),
        new ListViewItem("Erin Hagens", 0),
        new ListViewItem("Barry Johnson", 0),
        new ListViewItem("Jay Hamlin", 0),
        new ListViewItem("Brian Valentine", 0),
        new ListViewItem("Brian Welker", 0),
        new ListViewItem("Daniel Weisman", 0) });
    this.Controls.Add(iconListView);
    this.Controls.Add(previousItemBox);

    // Handle the MouseDown event to capture user input.
    iconListView.MouseDown +=
        new MouseEventHandler(iconListView_MouseDown);
    //iconListView.MouseWheel += new MouseEventHandler(iconListView_MouseWheel);
}

void iconListView_MouseDown(object sender, MouseEventArgs e)
{
    // Find the an item above where the user clicked.
    ListViewItem foundItem =
        iconListView.FindNearestItem(SearchDirectionHint.Up, e.X, e.Y);

    // Display the results in a textbox..
    if (foundItem != null)
        previousItemBox.Text = foundItem.Text;
    else
        previousItemBox.Text = "No item found";
}

```

```

Private WithEvents iconListView As New ListView()
Private previousItemBox As New TextBox()

Private Sub InitializeLocationSearchListView()
    previousItemBox.Location = New Point(150, 20)

    ' Create an image list for the icon ListView.
    iconListView.LargeImageList = New ImageList()

    ' Add an image to the ListView large icon list.
    iconListView.LargeImageList.Images.Add(New Bitmap(GetType(Control), "Edit.bmp"))

    ' Set the view to large icon and add some items with the image
    ' in the image list.
    iconListView.View = View.SmallIcon
    iconListView.Items.AddRange(New ListViewItem() {
        New ListViewItem("Amy Alberts", 0),
        New ListViewItem("Amy Recker", 0),
        New ListViewItem("Erin Hagens", 0),
        New ListViewItem("Barry Johnson", 0),
        New ListViewItem("Jay Hamlin", 0),
        New ListViewItem("Brian Valentine", 0),
        New ListViewItem("Brian Welker", 0),
        New ListViewItem("Daniel Weisman", 0)})

    Me.Controls.Add(iconListView)
    Me.Controls.Add(previousItemBox)
End Sub

Sub iconListView_MouseDown(ByVal sender As Object, ByVal e As MouseEventArgs) _
Handles iconListView.MouseDown

    ' Find the next item up from where the user clicked.
    Dim foundItem As ListViewItem =
        iconListView.FindNearestItem(SearchDirectionHint.Up, e.X, e.Y)

    ' Display the results in a textbox.
    If (foundItem IsNot Nothing) Then
        previousItemBox.Text = foundItem.Text
    Else
        previousItemBox.Text = "No item found"
    End If

End Sub

```

See Also

[ListView](#)
[FindItemWithText](#)
[FindNearestItem](#)
[ListView Control](#)
[ListView Control Overview](#)
[How to: Add and Remove Items with the Windows Forms ListView Control](#)

How to: Display Icons for the Windows Forms ListView Control

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [ListView](#) control can display icons from three image lists. The List, Details, and SmallIcon views display images from the image list specified in the [SmallImageList](#) property. The LargeIcon view displays images from the image list specified in the [LargeImageList](#) property. A list view can also display an additional set of icons, set in the [StateImageList](#) property, next to the large or small icons. For more information about image lists, see [ImageList Component](#) and [How to: Add or Remove Images with the Windows Forms ImageList Component](#).

To display images in a list view

1. Set the appropriate property—[SmallImageList](#), [LargeImageList](#), or [StateImageList](#)—to the existing [ImageList](#) component you wish to use.

These properties can be set in the designer with the Properties window, or in code.

```
listView1.SmallImageList = imageList1;
```

```
ListView1.SmallImageList = ImageList1
```

2. Set the [ImageIndex](#) or [StateImageIndex](#) property for each list item that has an associated icon.

These properties can be set in code, or within the [ListViewItem Collection Editor](#). To open the [ListViewItem Collection Editor](#), click the ellipsis button (...) next to the [Items](#) property on the [Properties](#) window.

```
// Sets the first list item to display the 4th image.  
listView1.Items[0].ImageIndex = 3;
```

```
' Sets the first list item to display the 4th image.  
ListView1.Items(0).ImageIndex = 3
```

See Also

[ListView Control Overview](#)

[How to: Add and Remove Items with the Windows Forms ListView Control](#)

[How to: Add Columns to the Windows Forms ListView Control](#)

[How to: Add Custom Information to a TreeView or ListView Control \(Windows Forms\)](#)

[ImageList Component](#)

How to: Display Subitems in Columns with the Windows Forms ListView Control

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [ListView](#) control can display additional text, or subitems, for each item in the Details view. The first column displays the item text, for example an employee number. The second, third, and subsequent columns display the first, second, and subsequent associated subitems.

To add subitems to a list item

1. Add any columns needed. Because the first column will display the item's [Text](#) property, you need one more column than there are subitems. For more information on adding columns, see [How to: Add Columns to the Windows Forms ListView Control](#).
2. Call the [Add](#) method of the collection returned by the [SubItems](#) property of an item. The code example below sets the employee name and department for a list item.

```
// Adds two subitems to the first list item.  
listView1.Items[0].SubItems.Add("John Smith");  
listView1.Items[0].SubItems.Add("Accounting");
```

```
' Adds two subitems to the first list item  
ListView1.Items(0).SubItems.Add("John Smith")  
ListView1.Items(0).SubItems.Add("Accounting")
```

See Also

[ListView Control Overview](#)

[How to: Add and Remove Items with the Windows Forms ListView Control](#)

[How to: Add Columns to the Windows Forms ListView Control](#)

[How to: Display Icons for the Windows Forms ListView Control](#)

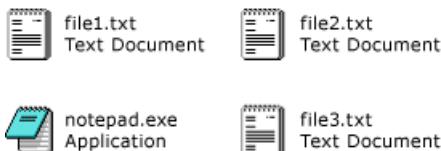
[How to: Add Custom Information to a TreeView or ListView Control \(Windows Forms\)](#)

How to: Enable Tile View in a Windows Forms ListView Control

5/4/2018 • 5 min to read • [Edit Online](#)

With the tile view feature of the [ListView](#) control, you can provide a visual balance between graphical and textual information. The textual information displayed for an item in tile view is the same as the column information defined for details view. Tile view works in combination with either the grouping or insertion mark features in the [ListView](#) control.

The tile view uses a 32 x 32 pixel icon and several lines of text, as shown in the following images.



Tile view icons and text

To enable tile view, set the [View](#) property to [Tile](#). You can adjust the size of the tiles by setting the [TileSize](#) property, and the number of text lines displayed in the tile by adjusting the [Columns](#) collection.

NOTE

The tile view is available only on Windows XP Home Edition, Windows XP Professional, Windows Server 2003 when your application calls the [Application.EnableVisualStyles](#) method. On earlier operating systems, any code related to the tile view has no effect, and the [ListView](#) control displays in the large icon view. For more information, see [ListView.View](#).

To set tile view programmatically

1. Use the [View](#) enumeration of the [ListView](#) control.

```
ListView1.View = View.Tile
```

```
listView1.View = View.Tile;
```

Example

The following complete code example demonstrates Tile view with tiles modified to show three lines of text. The tile size has been adjusted to prevent line-wrapping.

```
#using <System.dll>
#using <System.Windows.Forms.dll>
#using <System.Drawing.dll>

using namespace System;
using namespace System::Drawing;
using namespace System::Windows::Forms;

public ref class ListViewTilingExample: public Form
{
private:
    ImageList^ myImageList;
```

```

public:
    ListViewTilingExample()
    {
        // Initialize myListview.
        ListView^ myListview = gcnew ListView;
        myListview->Dock = DockStyle::Fill;
        myListview->View = View::Tile;

        // Initialize the tile size.
        myListview->TileSize = System::Drawing::Size( 400, 45 );

        // Initialize the item icons.
        myImageList = gcnew ImageList;
        System::Drawing::Icon^ myIcon = gcnew System::Drawing::Icon( "book.ico" );
        try
        {
            myImageList->Images->Add( myIcon );
        }
        finally
        {
            if ( myIcon )
                delete safe_cast<IDisposable^>(myIcon);
        }

        myImageList->ImageSize = System::Drawing::Size( 32, 32 );
        myListview->LargeImageList = myImageList;

        // Add column headers so the subitems will appear.
        array<ColumnHeader^>^temp0 = {gcnew ColumnHeader,gcnew ColumnHeader,gcnew ColumnHeader};
        myListview->Columns->AddRange( temp0 );

        // Create items and add them to myListview.
        array<String^>^temp1 = {"Programming Windows","Petzold, Charles","1998"};
        ListViewItem^ item0 = gcnew ListViewItem( temp1,0 );
        array<String^>^temp2 = {"Code: The Hidden Language of Computer Hardware and Software","Petzold, Charles","2000"};
        ListViewItem^ item1 = gcnew ListViewItem( temp2,0 );
        array<String^>^temp3 = {"Programming Windows with C#","Petzold, Charles","2001"};
        ListViewItem^ item2 = gcnew ListViewItem( temp3,0 );
        array<String^>^temp4 = {"Coding Techniques for Microsoft Visual Basic .NET","Connell, John","2001"};
        ListViewItem^ item3 = gcnew ListViewItem( temp4,0 );
        array<String^>^temp5 = {"C# for Java Developers","Jones, Allen & Freeman, Adam","2002"};
        ListViewItem^ item4 = gcnew ListViewItem( temp5,0 );
        array<String^>^temp6 = {"Microsoft .NET XML Web Services Step by Step","Jones, Allen & Freeman, Adam","2002"};
        ListViewItem^ item5 = gcnew ListViewItem( temp6,0 );
        array<ListViewItem^>^temp7 = {item0,item1,item2,item3,item4,item5};
        myListview->Items->AddRange( temp7 );

        // Initialize the form.
        this->Controls->Add( myListview );
        this->Size = System::Drawing::Size( 430, 330 );
        this->Text = "ListView Tiling Example";
    }

protected:

    // Clean up any resources being used.
    ~ListViewTilingExample()
    {
        if ( myImageList != nullptr )
        {
            delete myImageList;
        }
    }
};

[STAThread]

```

```
int main()
{
    Application::EnableVisualStyles();
    Application::Run( gcnew ListViewTilingExample );
}
```

```
using System;
using System.Drawing;
using System.Windows.Forms;

public class ListViewTilingExample : Form
{
    private ImageList myImageList;

    public ListViewTilingExample()
    {
        // Initialize myListview.
        ListView myListview = new ListView();
        myListview.Dock = DockStyle.Fill;
        myListview.View = View.Tile;

        // Initialize the tile size.
        myListview.TileSize = new Size(400, 45);

        // Initialize the item icons.
        myImageList = new ImageList();
        using (Icon myIcon = new Icon("book.ico"))
        {
            myImageList.Images.Add(myIcon);
        }
        myImageList.ImageSize = new Size(32, 32);
        myListview.LargeImageList = myImageList;

        // Add column headers so the subitems will appear.
        myListview.Columns.AddRange(new ColumnHeader[]
        {new ColumnHeader(), new ColumnHeader(), new ColumnHeader()});

        // Create items and add them to myListview.
        ListViewItem item0 = new ListViewItem( new string[]
        {"Programming Windows",
        "Petzold, Charles",
        "1998"}, 0 );
        ListViewItem item1 = new ListViewItem( new string[]
        {"Code: The Hidden Language of Computer Hardware and Software",
        "Petzold, Charles",
        "2000"}, 0 );
        ListViewItem item2 = new ListViewItem( new string[]
        {"Programming Windows with C#",
        "Petzold, Charles",
        "2001"}, 0 );
        ListViewItem item3 = new ListViewItem( new string[]
        {"Coding Techniques for Microsoft Visual Basic .NET",
        "Connell, John",
        "2001"}, 0 );
        ListViewItem item4 = new ListViewItem( new string[]
        {"C# for Java Developers",
        "Jones, Allen & Freeman, Adam",
        "2002"}, 0 );
        ListViewItem item5 = new ListViewItem( new string[]
        {"Microsoft .NET XML Web Services Step by Step",
        "Jones, Allen & Freeman, Adam",
        "2002"}, 0 );
        myListview.Items.AddRange(
        new ListViewItem[] {item0, item1, item2, item3, item4, item5});

        // Initialize the form.
        this.Controls.Add(myListview);
    }
}
```

```

        this.Size = new System.Drawing.Size(430, 330);
        this.Text = "ListView Tiling Example";
    }

    // Clean up any resources being used.
    protected override void Dispose(bool disposing)
    {
        if (disposing)
        {
            myImageList.Dispose();
        }
        base.Dispose(disposing);
    }

    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.Run(new ListViewTilingExample());
    }
}

```

```

Imports System
Imports System.Drawing
Imports System.Windows.Forms

Public Class ListViewTilingExample
    Inherits Form

    Private myImageList As ImageList

    Public Sub New()
        ' Initialize myListview.
        Dim myListview As New ListView()
        myListview.Dock = DockStyle.Fill
        myListview.View = View.Tile

        ' Initialize the tile size.
        myListview.TileSize = New Size(400, 45)

        ' Initialize the item icons.
        myImageList = New ImageList()
        Dim myIcon as Icon = new Icon("book.ico")
        Try
            myImageList.Images.Add(myIcon)
        Finally
            myIcon.Dispose()
        End Try
        myIcon.Dispose()
        myImageList.ImageSize = New Size(32, 32)
        myListview.LargeImageList = myImageList

        ' Add column headers so the subitems will appear.
        myListview.Columns.AddRange(New ColumnHeader() _
            {New ColumnHeader(), New ColumnHeader(), New ColumnHeader()})

        ' Create items and add them to myListview.
        Dim item0 As New ListViewItem( New String() _
            {"Programming Windows", _
            "Petzold, Charles", _
            "1998"}, 0 )
        Dim item1 As New ListViewItem( New String() _
            {"Code: The Hidden Language of Computer Hardware and Software", _
            "Petzold, Charles", _
            "2000"}, 0 )
        Dim item2 As New ListViewItem( New String() _
            {"", "", ""})

```

```

        {"Programming Windows with C#", _
        "Petzold, Charles", _
        "2001"}, 0 )
Dim item3 As New ListViewItem( New String() _
    {"Coding Techniques for Microsoft Visual Basic .NET", _
    "Connell, John", _
    "2001"}, 0 )
Dim item4 As New ListViewItem( New String() _
    {"C# for Java Developers", _
    "Jones, Allen / Freeman, Adam", _
    "2002"}, 0 )
Dim item5 As New ListViewItem( New String() _
    {"Microsoft .NET XML Web Services Step by Step", _
    "Jones, Allen / Freeman, Adam", _
    "2002"}, 0 )
myListView.Items.AddRange( _
    New ListViewItem() {item0, item1, item2, item3, item4, item5})

' Initialize the form.
Me.Controls.Add(myListView)
Me.Size = new System.Drawing.Size(430, 330)
Me.Text = "ListView Tiling Example"
End Sub 'New

' Clean up any resources being used.
Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    If (disposing) Then
        myImageList.Dispose()
    End If

    MyBase.Dispose(disposing)
End Sub

<STAThread()> _
Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New ListViewTilingExample())
End Sub 'Main

End Class 'ListViewTilingExample

```

Compiling the Code

This example requires:

- References to the System and System.Windows.Forms assemblies.
- An icon file named book.ico in the same directory as the executable file.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[ListView](#)

[TileSize](#)

[ListView Control](#)

[ListView Control Overview](#)

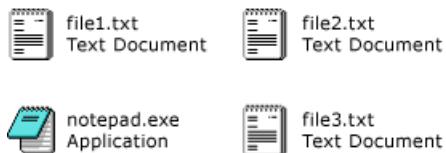
[Windows XP Features and Windows Forms Controls](#)

How to: Enable Tile View in a Windows Forms ListView Control Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

The tile view feature of the [ListView](#) control enables you to provide a visual balance between graphical and textual information. The textual information displayed for an item in tile view is the same as the column information defined for details view. Tile view functions in combination with either the grouping or insertion mark features in the [ListView](#) control.

The tile view uses a 32 x 32 icon and several lines of text, as shown in the following image.



Tile view properties and methods enable you to specify which column fields to display for each item, and to collectively control the size and appearance of all items within a tile view window. For clarity, the first line of text in a tile is always the item's name; it cannot be changed.

The following procedure requires a **Windows Application** project with a form containing a [ListView](#) control. For information about setting up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#).

NOTE

The tile view is available only on Windows XP Home Edition, Windows XP Professional, Windows Server 2003 when your application calls the [Application.EnableVisualStyles](#) method. On earlier operating systems, any code related to the tile view has no effect, and the [ListView](#) control displays in the large icon view. For more information, see [ListView.View](#).

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To set tile view in the designer

1. Select the [ListView](#) control on your form.
2. In the **Properties** window, select the [View](#) property and choose **Tile**.

See Also

[TileSize](#)

[Windows XP Features and Windows Forms Controls](#)

[ListView Control Overview](#)

How to: Group Items in a Windows Forms ListView Control

5/4/2018 • 1 min to read • [Edit Online](#)

With the grouping feature of the [ListView](#) control, you can display related sets of items in groups. These groups are separated on the screen by horizontal group headers that contain the group titles. You can use [ListView](#) groups to make navigating large lists easier by grouping items alphabetically, by date, or by any other logical grouping. The following image shows some grouped items.

odd

one
three
five

even

two
four
six

ListView grouped items

To enable grouping, you must first create one or more groups either in the designer or programmatically. After a group has been defined, you can assign [ListView](#) items to groups. You can also move items from one group to another programmatically.

NOTE

[ListView](#) groups are available only on Windows XP Home Edition, Windows XP Professional, Windows Server 2003 when your application calls the [Application.EnableVisualStyles](#) method. On earlier operating systems, any code relating to groups has no effect and the groups will not appear. For more information, see [ListView.Groups](#).

To add groups

1. Use the [Add](#) method of the [Groups](#) collection.

```
// Adds a new group that has a left-aligned header
listView1.Groups.Add(new ListViewGroup("List item text",
    HorizontalAlignment.Left));
```

```
' Adds a new group that has a left-aligned header
ListView1.Groups.Add(New ListViewGroup("Group 1",
    HorizontalAlignment.Left))
```

To remove groups

1. Use the [RemoveAt](#) or [Clear](#) method of the [Groups](#) collection.

The [RemoveAt](#) method removes a single group; the [Clear](#) method removes all groups from the list.

NOTE

Removing a group does not remove the items within that group.

```
// Removes the first group in the collection.  
listView1.Groups.RemoveAt(0);  
// Clears all groups.  
listView1.Groups.Clear();
```

```
' Removes the first group in the collection.  
ListView1.Groups.RemoveAt(0)  
' Clears all groups:  
ListView1.Groups.Clear()
```

To assign items to groups or move items between groups

1. Set the [ListViewItem.Group](#) property of individual items.

```
// Adds the first item to the first group  
listView1.Items[0].Group = listView1.Groups[0];
```

```
' Adds the first item to the first group  
ListView1.Items.Item(0).Group = ListView1.Groups(0)
```

See Also

[ListView](#)

[ListView.Groups](#)

[ListViewGroup](#)

[ListView Control](#)

[ListView Control Overview](#)

[Windows XP Features and Windows Forms Controls](#)

[How to: Add and Remove Items with the Windows Forms ListView Control](#)

How to: Group Items in a Windows Forms ListView Control Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

The grouping feature of the [ListView](#) control enables you to display related sets of items in groups. These groups are separated on the screen by horizontal group headers that contain the group titles. You can use [ListView](#) groups to make navigating large lists easier by grouping items alphabetically, by date, or by any other logical grouping. The following image shows some grouped items.

odd

one
three
five

even

two
four
six

The following procedure requires a **Windows Application** project with a form containing a [ListView](#) control. For information about setting up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#).

To enable grouping, you must first create one or more [ListViewGroup](#) objects either in the designer or programmatically. Once a group has been defined, you can assign items to it.

NOTE

[ListView](#) groups are available only on Windows XP Home Edition, Windows XP Professional, Windows Server 2003 when your application calls the [Application.EnableVisualStyles](#) method. On earlier operating systems, any code relating to groups has no effect and the groups will not appear. For more information, see [ListView.Groups](#).

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To add or remove groups in the designer

1. In the **Properties** window, click the **Ellipsis** (...) button next to the **Groups** property.

The **ListViewGroup Collection Editor** appears.

2. To add a group, click the **Add** button. You can then set properties of the new group, such as the **Header** and **HeaderAlignment** properties. To remove a group, select it and click the **Remove** button.

To assign items to groups in the designer

1. In the **Properties** window, click the **Ellipsis** (...) button next to the **Items** property.

The **ListViewItem Collection Editor** appears.

2. To add a new item, click the **Add** button. You can then set properties of the new item, such as the **Text** and **ImageIndex** properties.
3. Select the **Group** property and choose a group from the drop-down list.

See Also

[ListView](#)

[Groups](#)

[ListViewGroup](#)

[ListView Control](#)

[ListView Control Overview](#)

[Windows XP Features and Windows Forms Controls](#)

[How to: Add and Remove Items with the Windows Forms ListView Control](#)

How to: Display an Insertion Mark in a Windows Forms ListView Control

5/4/2018 • 8 min to read • [Edit Online](#)

The insertion mark in the [ListView](#) control shows users the point where dragged items will be inserted. When a user drags an item to a point between two other items, the insertion mark shows the item's expected new location.

NOTE

The insertion mark feature is available only on Windows XP Home Edition, Windows XP Professional, Windows Server 2003 when your application calls the [Application.EnableVisualStyles](#) method. On earlier operating systems, any code relating to the insertion mark has no effect and the insertion mark will not appear. For more information, see [ListViewInsertionMark](#).

The following image shows an insertion mark:



The following code example demonstrates how to use this feature.

Example

```
#using <System.dll>
#using <System.Windows.Forms.dll>
#using <System.Drawing.dll>

using namespace System;
using namespace System::Drawing;
using namespace System::Windows::Forms;
public ref class ListViewInsertionMarkExample: public Form
{
private:
    ListView^ myListview;

public:

    ListViewInsertionMarkExample()
    {
        // Initialize myListview.
        myListview = gcnew ListView;
        myListview->Dock = DockStyle::Fill;
        myListview->View = View::LargeIcon;
        myListview->MultiSelect = false;
        myListview->ListViewItemSorter = gcnew ListViewItemComparer;

        // Initialize the insertion mark.
        myListview->InsertionMark->Color = Color::Green;

        // Add items to myListview.
        myListview->Items->Add( "zero" );
        myListview->Items->Add( "one" );
        myListview->Items->Add( "two" );
        myListview->Items->Add( "three" );
        myListview->Items->Add( "four" );
        myListview->Items->Add( "five" );
    }
}
```

```

// Initialize the drag-and-drop operation when running
// under Windows XP or a later operating system.
if ( System::Environment::OSVersion->Version->Major > 5 || (System::Environment::OSVersion->Version-
>Major == 5 && System::Environment::OSVersion->Version->Minor >= 1) )
{
    myListview->AllowDrop = true;
    myListview->ItemDrag += gcnew ItemDragEventHandler( this,
&ListViewInsertionMarkExample::myListview_ItemDrag );
    myListview->DragEnter += gcnew DragEventHandler( this,
&ListViewInsertionMarkExample::myListview_DragEnter );
    myListview->DragOver += gcnew DragEventHandler( this,
&ListViewInsertionMarkExample::myListview_DragOver );
    myListview->DragLeave += gcnew EventHandler( this,
&ListViewInsertionMarkExample::myListview_DragLeave );
    myListview->DragDrop += gcnew DragEventHandler( this,
&ListViewInsertionMarkExample::myListview_DragDrop );
}

// Initialize the form.
this->Text = "ListView Insertion Mark Example";
this->Controls->Add( myListview );
}

private:

// Starts the drag-and-drop operation when an item is dragged.
void myListview_ItemDrag( Object^ /*sender*/, ItemDragEventArgs^ e )
{
    myListview->DoDragDrop( e->Item, DragDropEffects::Move );
}

// Sets the target drop effect.
void myListview_DragEnter( Object^ /*sender*/, DragEventArgs^ e )
{
    e->Effect = e->AllowedEffect;
}

// Moves the insertion mark as the item is dragged.
void myListview_DragOver( Object^ /*sender*/, DragEventArgs^ e )
{
    // Retrieve the client coordinates of the mouse pointer.
    Point targetPoint = myListview->PointToClient( Point(e->X,e->Y) );

    // Retrieve the index of the item closest to the mouse pointer.
    int targetIndex = myListview->InsertionMark->NearestIndex( targetPoint );

    // Confirm that the mouse pointer is not over the dragged item.
    if ( targetIndex > -1 )
    {
        // Determine whether the mouse pointer is to the left or
        // the right of the midpoint of the closest item and set
        // the InsertionMark.AppearsAfterItem property accordingly.
        Rectangle itemBounds = myListview->GetItemRect( targetIndex );
        if ( targetPoint.X > itemBounds.Left + (itemBounds.Width / 2) )
        {
            myListview->InsertionMark->AppearsAfterItem = true;
        }
        else
        {
            myListview->InsertionMark->AppearsAfterItem = false;
        }
    }

    // Set the location of the insertion mark. If the mouse is
    // over the dragged item, the targetIndex value is -1 and
    // the insertion mark disappears.
    myListview->InsertionMark->Index = targetIndex;
}

```

```

// Removes the insertion mark when the mouse leaves the control.
void myListview_DragLeave( Object^ /*sender*/, EventArgs^ /*e*/ )
{
    myListview->InsertionMark->Index = -1;
}

// Moves the item to the location of the insertion mark.
void myListview_DragDrop( Object^ /*sender*/, DragEventArgs^ e )
{
    // Retrieve the index of the insertion mark;
    int targetIndex = myListview->InsertionMark->Index;

    // If the insertion mark is not visible, exit the method.
    if ( targetIndex == -1 )
    {
        return;
    }

    // If the insertion mark is to the right of the item with
    // the corresponding index, increment the target index.
    if ( myListview->InsertionMark->AppearsAfterItem )
    {
        targetIndex++;
    }

    // Retrieve the dragged item.
    ListViewitem^ draggedItem = dynamic_cast<ListViewitem^>(e->Data->GetData( ListViewitem::typeid ));

    // Insert a copy of the dragged item at the target index.
    // A copy must be inserted before the original item is removed
    // to preserve item index values.
    myListview->Items->Insert( targetIndex, dynamic_cast<ListViewitem^>(draggedItem->Clone()) );

    // Remove the original copy of the dragged item.
    myListview->Items->Remove( draggedItem );
}

// Sorts ListViewItem objects by index.
ref class ListViewIndexComparer: public System::Collections::IComparer
{
public:
    virtual int Compare( Object^ x, Object^ y )
    {
        return (dynamic_cast<ListViewitem^>(x))->Index - (dynamic_cast<ListViewitem^>(y))->Index;
    }
};

[STAThread]
int main()
{
    Application::EnableVisualStyles();
    Application::Run( gcnew ListViewInsertionMarkExample );
}

```

```

using System;
using System.Drawing;
using System.Windows.Forms;

public class ListViewInsertionMarkExample : Form
{
    private ListView myListview;

    public ListViewInsertionMarkExample()
    {
        // Initialize mylistview.
    }
}

```

```

// Initialize myListview.
myListView = new ListView();
myListView.Dock = DockStyle.Fill;
myListView.View = View.LargeIcon;
myListView.MultiSelect = false;
myListView.ListViewItemSorter = new ListViewItemComparer();

// Initialize the insertion mark.
myListView.InsertionMark.Color = Color.Green;

// Add items to myListview.
myListView.Items.Add("zero");
myListView.Items.Add("one");
myListView.Items.Add("two");
myListView.Items.Add("three");
myListView.Items.Add("four");
myListView.Items.Add("five");

// Initialize the drag-and-drop operation when running
// under Windows XP or a later operating system.
if (OSFeature.Feature.isPresent(OSFeature.Themes))
{
    myListview.AllowDrop = true;
    myListview.ItemDrag += new ItemDragEventHandler(myListView_ItemDrag);
    myListview.DragEnter += new DragEventHandler(myListView_DragEnter);
    myListview.DragOver += new DragEventHandler(myListView_DragOver);
    myListview.DragLeave += new EventHandler(myListView_DragLeave);
    myListview.DragDrop += new DragEventHandler(myListView_DragDrop);
}

// Initialize the form.
this.Text = "ListView Insertion Mark Example";
this.Controls.Add(myListView);
}

[STAThread]
static void Main()
{
    Application.EnableVisualStyles();
    Application.Run(new ListViewInsertionMarkExample());
}

// Starts the drag-and-drop operation when an item is dragged.
private void myListview_ItemDrag(object sender, ItemDragEventArgs e)
{
    myListview.DoDragDrop(e.Item, DragDropEffects.Move);
}

// Sets the target drop effect.
private void myListview_DragEnter(object sender, DragEventArgs e)
{
    e.Effect = e.AllowedEffect;
}

// Moves the insertion mark as the item is dragged.
private void myListview_DragOver(object sender, DragEventArgs e)
{
    // Retrieve the client coordinates of the mouse pointer.
    Point targetPoint =
        myListview.PointToClient(new Point(e.X, e.Y));

    // Retrieve the index of the item closest to the mouse pointer.
    int targetIndex = myListview.InsertionMark.NearestIndex(targetPoint);

    // Confirm that the mouse pointer is not over the dragged item.
    if (targetIndex > -1)
    {
        // Determine whether the mouse pointer is to the left or
        // the right of the midpoint of the closest item and set
        // the InsertionMark.AllowDropFrontToBack property accordingly.
    }
}

```

```

// the InsertionMark.AppearsAfterItem property accordingly.
Rectangle itemBounds = myListview.GetItemRect(targetIndex);
if ( targetPoint.X > itemBounds.Left + (itemBounds.Width / 2) )
{
    myListview.InsertionMark.AppearsAfterItem = true;
}
else
{
    myListview.InsertionMark.AppearsAfterItem = false;
}

// Set the location of the insertion mark. If the mouse is
// over the dragged item, the targetIndex value is -1 and
// the insertion mark disappears.
myListview.InsertionMark.Index = targetIndex;
}

// Removes the insertion mark when the mouse leaves the control.
private void myListview_DragLeave(object sender, EventArgs e)
{
    myListview.InsertionMark.Index = -1;
}

// Moves the item to the location of the insertion mark.
private void myListview_DragDrop(object sender, DragEventArgs e)
{
    // Retrieve the index of the insertion mark;
    int targetIndex = myListview.InsertionMark.Index;

    // If the insertion mark is not visible, exit the method.
    if (targetIndex == -1)
    {
        return;
    }

    // If the insertion mark is to the right of the item with
    // the corresponding index, increment the target index.
    if (myListview.InsertionMark.AppearsAfterItem)
    {
        targetIndex++;
    }

    // Retrieve the dragged item.
    ListViewItem draggedItem =
        (ListViewItem)e.Data.GetData(typeof(ListViewItem));

    // Insert a copy of the dragged item at the target index.
    // A copy must be inserted before the original item is removed
    // to preserve item index values.
    myListview.Items.Insert(
        targetIndex, (ListViewItem)draggedItem.Clone());

    // Remove the original copy of the dragged item.
    myListview.Items.Remove(draggedItem);
}

// Sorts ListViewItem objects by index.
private class ListViewItemComparer : System.Collections.IComparer
{
    public int Compare(object x, object y)
    {
        return ((ListViewItem)x).Index - ((ListViewItem)y).Index;
    }
}
}

```

```

Imports System
Imports System.Drawing
Imports System.Windows.Forms

Public Class ListViewInsertionMarkExample
    Inherits Form

    Private myListview As ListView

    Public Sub New()
        ' Initialize myListview.
        myListview = New ListView()
        myListview.Dock = DockStyle.Fill
        myListview.View = View.LargeIcon
        myListview.MultiSelect = False
        myListview.ListViewItemSorter = New ListViewItemComparer()

        ' Initialize the insertion mark.
        myListview.InsertionMark.Color = Color.Green

        ' Add items to myListview.
        myListview.Items.Add("zero")
        myListview.Items.Add("one")
        myListview.Items.Add("two")
        myListview.Items.Add("three")
        myListview.Items.Add("four")
        myListview.Items.Add("five")

        ' Initialize the drag-and-drop operation when running
        ' under Windows XP or a later operating system.
        If OSFeature.Feature.isPresent(OSFeature.Themes)
            myListview.AllowDrop = True
            AddHandler myListview.ItemDrag, AddressOf myListview_ItemDrag
            AddHandler myListview.DragEnter, AddressOf myListview_DragEnter
            AddHandler myListview.DragOver, AddressOf myListview_DragOver
            AddHandler myListview.DragLeave, AddressOf myListview_DragLeave
            AddHandler myListview.DragDrop, AddressOf myListview_DragDrop
        End If

        ' Initialize the form.
        Me.Text = "ListView Insertion Mark Example"
        Me.Controls.Add(myListview)
    End Sub 'New

    <STAThread()> _
    Shared Sub Main()
        Application.EnableVisualStyles()
        Application.Run(New ListViewInsertionMarkExample())
    End Sub 'Main

    ' Starts the drag-and-drop operation when an item is dragged.
    Private Sub myListview_ItemDrag(sender As Object, e As ItemDragEventArgs)
        myListview.DoDragDrop(e.Item, DragDropEffects.Move)
    End Sub 'myListview_ItemDrag

    ' Sets the target drop effect.
    Private Sub myListview_DragEnter(sender As Object, e As DragEventArgs)
        e.Effect = e.AllowedEffect
    End Sub 'myListview_DragEnter

    ' Moves the insertion mark as the item is dragged.
    Private Sub myListview_DragOver(sender As Object, e As DragEventArgs)
        ' Retrieve the client coordinates of the mouse pointer.
        Dim targetPoint As Point = myListview.PointToClient(New Point(e.X, e.Y))

        ' Retrieve the index of the item closest to the mouse pointer.
        Dim targetIndex As Integer = _
            myListview.InsertionMark.NearestIndex(targetPoint)
    End Sub 'myListview_DragOver
End Class

```

```

' Confirm that the mouse pointer is not over the dragged item.
If targetIndex > -1 Then
    ' Determine whether the mouse pointer is to the left or
    ' the right of the midpoint of the closest item and set
    ' the InsertionMark.AppearsAfterItem property accordingly.
    Dim itemBounds As Rectangle = myListview.GetItemRect(targetIndex)
    If targetPoint.X > itemBounds.Left + (itemBounds.Width / 2) Then
        myListview.InsertionMark.AppearsAfterItem = True
    Else
        myListview.InsertionMark.AppearsAfterItem = False
    End If
End If

' Set the location of the insertion mark. If the mouse is
' over the dragged item, the targetIndex value is -1 and
' the insertion mark disappears.
myListview.InsertionMark.Index = targetIndex
End Sub 'myListview_DragOver

' Removes the insertion mark when the mouse leaves the control.
Private Sub myListview_DragLeave(sender As Object, e As EventArgs)
    myListview.InsertionMark.Index = -1
End Sub 'myListview_DragLeave

' Moves the item to the location of the insertion mark.
Private Sub myListview_DragDrop(sender As Object, e As DragEventArgs)
    ' Retrieve the index of the insertion mark;
    Dim targetIndex As Integer = myListview.InsertionMark.Index

    ' If the insertion mark is not visible, exit the method.
    If targetIndex = -1 Then
        Return
    End If

    ' If the insertion mark is to the right of the item with
    ' the corresponding index, increment the target index.
    If myListview.InsertionMark.AppearsAfterItem Then
        targetIndex += 1
    End If

    ' Retrieve the dragged item.
    Dim draggedItem As ListViewItem =
        CType(e.Data.GetData(GetType(ListViewItem)), ListViewItem)

    ' Insert a copy of the dragged item at the target index.
    ' A copy must be inserted before the original item is removed
    ' to preserve item index values.
    myListview.Items.Insert(targetIndex, _
        CType(draggedItem.Clone(), ListViewItem))

    ' Remove the original copy of the dragged item.
    myListview.Items.Remove(draggedItem)

End Sub 'myListview_DragDrop

' Sorts ListViewItem objects by index.
Private Class ListViewIndexComparer
    Implements System.Collections.IComparer

    Public Function Compare(x As Object, y As Object) As Integer _
        Implements System.Collections.IComparer.Compare
        Return CType(x, ListViewItem).Index - CType(y, ListViewItem).Index
    End Function 'Compare

End Class 'ListViewIndexComparer

End Class 'ListViewInsertionMarkExample

```

Compiling the Code

This example requires:

- References to the System and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[ListView](#)

[ListView.InsertionMark](#)

[ListViewInsertionMark](#)

[ListView Control](#)

[ListView Control Overview](#)

[Windows XP Features and Windows Forms Controls](#)

[Walkthrough: Performing a Drag-and-Drop Operation in Windows Forms](#)

How to: Select an Item in the Windows Forms ListView Control

5/4/2018 • 1 min to read • [Edit Online](#)

This example demonstrates how to programmatically select an item in a Windows Forms [ListView](#) control. Selecting an item programmatically does not automatically change the focus to the [ListView](#) control. For this reason, you will typically also want to set the item as focused when selecting an item.

Example

```
this.listView1.Items[0].Focused = true;  
this.listView1.Items[0].Selected = true;
```

```
me.ListView1.Items(0).Focused = True  
me.ListView1.Items(0).Selected = True
```

Compiling the Code

This example requires:

- A [ListView](#) control named `listView1` that contains at least one item.
- References to the [System](#) and [System.Windows.Forms](#) namespaces.

See Also

[ListView](#)

[ListViewItem.Selected](#)

Walkthrough: Creating an Explorer Style Interface with the ListView and TreeView Controls Using the Designer

5/4/2018 • 4 min to read • [Edit Online](#)

One of the benefits of Visual Studio is the ability to create professional-looking Windows Forms applications in a short amount of time. A common scenario is creating a user interface (UI) with [ListView](#) and [TreeView](#) controls that resembles the Windows Explorer feature of Windows operating systems. Windows Explorer displays a hierarchical structure of the files and folders on a user's computer.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To create the form containing a ListView and TreeView control

1. On the **File** menu, point to **New**, and then click **Project**.
2. In the **New Project** dialog box, do the following:
 - a. In the categories, choose either **Visual Basic** or **Visual C#**.
 - b. In the list of templates, choose **Windows Forms Application**.
3. Click **OK**. A new Windows Forms project is created.
4. Add a [SplitContainer](#) control to the form and set its [Dock](#) property to [Fill](#).
5. Add an [ImageList](#) named `imageList1` to the form and use the Properties window to add two images: a folder image and a document image, in that order.
6. Add a [TreeView](#) control named `treeview1` to the form, and position it on the left side of the [SplitContainer](#) control. In the Properties window for `treeview1` do the following:
 - a. Set the [Dock](#) property to [Fill](#).
 - b. Set the [ImageList](#) property to `imageList1`.
7. Add a [ListView](#) control named `listView1` to the form, and position it on the right side of the [SplitContainer](#) control. In the Properties window for `listview1` do the following:
 - a. Set the [Dock](#) property to [Fill](#).
 - b. Set the [View](#) property to [Details](#).
 - c. Open the ColumnHeader Collection Editor by clicking the ellipses (`...`) in the [Columns](#) property. Add three columns and set their [Text](#) property to `Name`, `Type`, and `Last Modified`, respectively. Click **OK** to close the dialog box.
 - d. Set the [SmallImageList](#) property to `imageList1`.
8. Implement the code to populate the [TreeView](#) with nodes and subnodes. Add this code to the `Form1` class.

```

private void PopulateTreeView()
{
    TreeNode rootNode;

    DirectoryInfo info = new DirectoryInfo(@"../../");
    if (info.Exists)
    {
        rootNode = new TreeNode(info.Name);
        rootNode.Tag = info;
        GetDirectories(info.GetDirectories(), rootNode);
        treeView1.Nodes.Add(rootNode);
    }
}

private void GetDirectories(DirectoryInfo[] subDirs,
TreeNode nodeToAddTo)
{
    TreeNode aNode;
    DirectoryInfo[] subSubDirs;
    foreach (DirectoryInfo subDir in subDirs)
    {
        aNode = new TreeNode(subDir.Name, 0, 0);
        aNode.Tag = subDir;
        aNode.ImageKey = "folder";
        subSubDirs = subDir.GetDirectories();
        if (subSubDirs.Length != 0)
        {
            GetDirectories(subSubDirs, aNode);
        }
        nodeToAddTo.Nodes.Add(aNode);
    }
}

```

```

Private Sub PopulateTreeView()
    Dim rootNode As TreeNode

    Dim info As New DirectoryInfo("../..")
    If info.Exists Then
        rootNode = New TreeNode(info.Name)
        rootNode.Tag = info
        GetDirectories(info.GetDirectories(), rootNode)
        treeView1.Nodes.Add(rootNode)
    End If

End Sub

Private Sub GetDirectories(ByVal subDirs() As DirectoryInfo, _
    ByVal nodeToAddTo As TreeNode)

    Dim aNode As TreeNode
    Dim subSubDirs() As DirectoryInfo
    Dim subDir As DirectoryInfo
    For Each subDir In subDirs
        aNode = New TreeNode(subDir.Name, 0, 0)
        aNode.Tag = subDir
        aNode.ImageKey = "folder"
        subSubDirs = subDir.GetDirectories()
        If subSubDirs.Length <> 0 Then
            GetDirectories(subSubDirs, aNode)
        End If
        nodeToAddTo.Nodes.Add(aNode)
    Next subDir

End Sub

```

9. Since the previous code uses the System.IO namespace, add the appropriate using or import statement at the top of the form.

```
using System.IO;
```

```
Imports System.IO
```

10. Call the set-up method from the previous step in the form's constructor or [Load](#) event-handling method.

Add this code to the form constructor.

```
public Form1()
{
    InitializeComponent();
    PopulateTreeView();
}
```

```
Public Sub New()
    InitializeComponent()
    PopulateTreeView()

End Sub 'New
```

11. Handle the [NodeMouseClick](#) event for `treeview1`, and implement the code to populate `listview1` with a node's contents when a node is clicked. Add this code to the `Form1` class.

```
void treeView1_NodeMouseClick(object sender,
    TreeNodeMouseClickEventArgs e)
{
    TreeNode newSelected = e.Node;
    listView1.Items.Clear();
    DirectoryInfo nodeDirInfo = (DirectoryInfo)newSelected.Tag;
    ListView.ListViewSubItem[] subItems;
    ListViewItem item = null;

    foreach (DirectoryInfo dir in nodeDirInfo.GetDirectories())
    {
        item = new ListViewItem(dir.Name, 0);
        subItems = new ListViewItem.ListViewSubItem[]
            {new ListViewItem.ListViewSubItem(item, "Directory"),
             new ListViewItem.ListViewSubItem(item,
                 dir.LastAccessTime.ToShortDateString())};
        item.SubItems.AddRange(subItems);
        listView1.Items.Add(item);
    }
    foreach (FileInfo file in nodeDirInfo.GetFiles())
    {
        item = new ListViewItem(file.Name, 1);
        subItems = new ListViewItem.ListViewSubItem[]
            { new ListViewItem.ListViewSubItem(item, "File"),
              new ListViewItem.ListViewSubItem(item,
                  file.LastAccessTime.ToShortDateString())};

        item.SubItems.AddRange(subItems);
        listView1.Items.Add(item);
    }

    listView1.AutoResizeColumns(ColumnHeaderAutoResizeStyle.HeaderSize);
}
```

```

Private Sub treeView1_NodeMouseClick(ByVal sender As Object, _
    ByVal e As TreeNodeMouseClickEventArgs) _
    Handles treeView1.NodeMouseClick

    Dim newSelected As TreeNode = e.Node
    listView1.Items.Clear()
    Dim nodeDirInfo As DirectoryInfo = _
        CType(newSelected.Tag, DirectoryInfo)
    Dim subItems() As ListView.ListViewSubItem
    Dim item As ListViewItem = Nothing

    Dim dir As DirectoryInfo
    For Each dir In nodeDirInfo.GetDirectories()
        item = New ListViewItem(dir.Name, 0)
        subItems = New ListViewItem.ListViewSubItem() _
            {New ListViewItem.ListViewSubItem(item, "Directory"), _
            New ListViewItem.ListViewSubItem(item, _
            dir.LastAccessTime.ToString())}

        item.SubItems.AddRange(subItems)
        listView1.Items.Add(item)
    Next dir
    Dim file As FileInfo
    For Each file In nodeDirInfo.GetFiles()
        item = New ListViewItem(file.Name, 1)
        subItems = New ListViewItem.ListViewSubItem() _
            {New ListViewItem.ListViewSubItem(item, "File"), _
            New ListViewItem.ListViewSubItem(item, _
            file.LastAccessTime.ToString())}

        item.SubItems.AddRange(subItems)
        listView1.Items.Add(item)
    Next file

    listView1.AutoResizeColumns(ColumnHeaderAutoResizeStyle.HeaderSize)

End Sub

```

If you are using C#, make sure you have the [NodeMouseClick](#) event associated with its event-handling method. Add this code to the form constructor.

```

this.treeView1.NodeMouseClick +=
    new TreeNodeMouseClickEventHandler(this.treeView1_NodeMouseClick);

```

Testing the Application

You can now test the form to make sure it behaves as expected.

To test the form

- Press F5 to run the application.

You will see a split form containing a [TreeView](#) control that displays your project directory on the left side, and a [ListView](#) control on the right side with three columns. You can traverse the [TreeView](#) by selecting directory nodes, and the [ListView](#) is populated with the contents of the selected directory.

Next Steps

This application gives you an example of a way you can use [TreeView](#) and [ListView](#) controls together. For more information on these controls, see the following topics:

- [How to: Add Custom Information to a TreeView or ListView Control \(Windows Forms\)](#)
- [How to: Add Search Capabilities to a ListView Control](#)
- [How to: Attach a ShortCut Menu to a TreeView Node](#)

See Also

[ListView](#)

[TreeView](#)

[ListView Control](#)

[How to: Add and Remove Nodes with the Windows Forms TreeView Control](#)

[How to: Add and Remove Items with the Windows Forms ListView Control](#)

[How to: Add Columns to the Windows Forms ListView Control](#)

MainMenu Component (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

NOTE

Although `MenuStrip` and `ContextMenuStrip` replace and add functionality to the `MainMenu` and `ContextMenu` controls of previous versions, `MainMenu` and `ContextMenu` are retained for both backward compatibility and future use if you choose.

The Windows Forms `MainMenu` component displays a menu at run time.

In This Section

[MainMenu Component Overview](#)

Explains what this component is and its key features and properties.

Reference

[MainMenu](#)

Describes this class and has links to all its members.

See Also

[MenuStrip](#)

[ContextMenuStrip](#)

MainMenu Component Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

IMPORTANT

Although [MenuStrip](#) and [ContextMenuStrip](#) replace and add functionality to the [MainMenu](#) and [ContextMenu](#) controls of previous versions, [MainMenu](#) and [ContextMenu](#) are retained for both backward compatibility and future use if you choose.

The Windows Forms [MainMenu](#) component displays a menu at run time. All submenus of the main menu and individual items are [MenuItem](#) objects.

Key Properties

A menu item can be designated as the default item by setting the [DefaultItem](#) property to `true`. The default item appears in bold text when the menu is clicked. The menu item's [Checked](#) property is either `true` or `false`, and indicates whether the menu item is selected. The menu item's [RadioCheck](#) property customizes the appearance of the selected item: if [RadioCheck](#) is set to `true`, a radio button appears next to the item; if [RadioCheck](#) is set to `false`, a check mark appears next to the item.

See Also

[MainMenu](#)

[Menu](#)

[MenuItem](#)

[MenuStrip](#)

[ContextMenuStrip](#)

[MenuStrip Control Overview](#)

MaskedTextBox Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

This topic links to others about the `MaskedTextBox` control.

In This Section

[Walkthrough: Working with the MaskedTextBox Control](#)

Demonstrates the key features of the `MaskedTextBox` control.

[How to: Bind Data to the MaskedTextBox Control](#)

Demonstrates how to reformat the data when data in the database does not match the format expected by your mask definition.

Reference

[MaskedTextBox](#)

The primary class for the implementation of the masked text box control.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

Walkthrough: Working with the MaskedTextBox Control

5/4/2018 • 2 min to read • [Edit Online](#)

Tasks illustrated in this walkthrough include:

- Initializing the [MaskedTextBox](#) control
- Using the [MaskInputRejected](#) event handler to alert the user when a character does not conform to the mask
- Assigning a type to the [ValidatingType](#) property and using the [TypeValidationCompleted](#) event handler to alert the user when the value they're attempting to commit is not valid for the type

Creating the Project and Adding a Control

To add a MaskedTextBox control to your form

1. Open the form on which you want to place the [MaskedTextBox](#) control.
2. Drag a [MaskedTextBox](#) control from the **Toolbox** to your form.
3. Right-click the control and choose **Properties**. In the **Properties** window, select the **Mask** property and click the ... (ellipsis) button next to the property name.
4. In the **Input Mask** dialog box, select the **Short Date** mask and click **OK**.
5. In the **Properties** window set the [BeepOnError](#) property to `true`. This property causes a short beep to sound every time the user attempts to input a character that violates the mask definition.

For a summary of the characters that the Mask property supports, see the Remarks section of the [Mask](#) property.

Alert the User to Input Errors

Add a balloon tip for rejected mask input

1. Return to the **Toolbox** and add a [ToolTip](#) to your form.
2. Create an event handler for the [MaskInputRejected](#) event that raises the [ToolTip](#) when an input error occurs. The balloon tip remains visible for five seconds, or until the user clicks it.

```
public void Form1_Load(Object sender, EventArgs e)
{
    ... // Other initialization code
    maskedTextBox1.Mask = "00/00/0000";
    maskedTextBox1.MaskInputRejected += new
    MaskInputRejectedEventHandler(maskedTextBox1_MaskInputRejected)
}

void maskedTextBox1_MaskInputRejected(object sender, MaskInputRejectedEventArgs e)
{
    toolTip1.ToolTipTitle = "Invalid Input";
    toolTip1.Show("We're sorry, but only digits (0-9) are allowed in dates.", maskedTextBox1,
    maskedTextBox1.Location, 5000);
}
```

```

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Me.ToolTip1.IsBalloon = True
    Me.MaskedTextBox1.Mask = "00/00/0000"
End Sub

Private Sub MaskedTextBox1_MaskInputRejected(sender as Object, e as MaskInputRejectedEventArgs) Handles
    MaskedTextBox1.MaskInputRejected
    ToolTip1.ToolTipTitle = "Invalid Input"
    ToolTip1.Show("We're sorry, but only digits (0-9) are allowed in dates.", MaskedTextBox1, 5000)
End Sub

```

Alert the User to a Type that Is Not Valid

Add a balloon tip for invalid data types

1. In your form's [Load](#) event handler, assign a [Type](#) object representing the [DateTime](#) type to the [MaskedTextBox](#) control's [ValidatingType](#) property:

```

private void Form1_Load(Object sender, EventArgs e)
{
    // Other code
    maskedTextBox1.ValidatingType = typeof(System.DateTime);
    maskedTextBox1.TypeValidationCompleted += new
    TypeValidationEventHandler(maskedTextBox1_TypeValidationCompleted);
}

```

```

Private Sub Form1_Load(sender as Object, e as EventArgs)
    // Other code
    MaskedTextBox1.ValidatingType = GetType(System.DateTime)
End Sub

```

2. Add an event handler for the [TypeValidationCompleted](#) event:

```

public void maskedTextBox1_TypeValidationCompleted(object sender, TypeValidationEventArgs e)
{
    if (!e.IsValidInput)
    {
        toolTip1.ToolTipTitle = "Invalid Date Value";
        toolTip1.Show("We're sorry, but the value you entered is not a valid date. Please change the
value.", maskedTextBox1, 5000);
        e.Cancel = true;
    }
}

```

```

Public Sub MaskedTextBox1_TypeValidationCompleted(sender as Object, e as TypeValidationEventArgs)
    If Not e.IsValidInput Then
        ToolTip1.ToolTipTitle = "Invalid Date Value"
        ToolTip1.Show("We're sorry, but the value you entered is not a valid date. Please change the
value.", maskedTextBox1, 5000)
        e.Cancel = True
    End If
End Sub

```

See Also

[MaskedTextBox](#)

[MaskedTextBox Control](#)

How to: Bind Data to the MaskedTextBox Control

5/4/2018 • 10 min to read • [Edit Online](#)

You can bind data to a [MaskedTextBox](#) control just as you can to any other Windows Forms control. However, if the format of your data in the database does not match the format expected by your mask definition, you will need to reformat the data. The following procedure demonstrates how to do this using the [Format](#) and [Parse](#) events of the [Binding](#) class to display separate phone number and phone extension database fields as a single editable field.

The following procedure requires that you have access to a SQL Server database with the Northwind sample database installed.

To bind data to a MaskedTextBox control

1. Create a new Windows Forms project.
2. Drag two [TextBox](#) controls onto your form; name them `FirstName` and `LastName`.
3. Drag a [MaskedTextBox](#) control onto your form; name it `PhoneMask`.
4. Set the [Mask](#) property of `PhoneMask` to `(000) 000-0000 x9999`.
5. Add the following namespace imports to the form.

```
using System.Data.SqlClient;
```

```
Imports System.Data.SqlClient
```

6. Right-click the form and choose **View Code**. Place this code anywhere in your form class.

```

Binding currentBinding, phoneBinding;
DataSet employeesTable = new DataSet();
SqlConnection sc;
SqlDataAdapter dataConnect;

private void Form1_Load(object sender, EventArgs e)
{
    DoMaskBinding();
}

private void DoMaskBinding()
{
    try
    {
        sc = new SqlConnection("Data Source=CLIENTUE;Initial Catalog=NORTHWIND;Integrated
Security=SSPI");
        sc.Open();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }

    dataConnect = new SqlDataAdapter("SELECT * FROM Employees", sc);
    dataConnect.Fill(employeesTable, "Employees");

    // Now bind MaskedTextBox to appropriate field. Note that we must create the Binding objects
    // before adding them to the control - otherwise, we won't get a Format event on the
    // initial load.
    try
    {
        currentBinding = new Binding("Text", employeesTable, "Employees.FirstName");
        firstName.DataBindings.Add(currentBinding);

        currentBinding = new Binding("Text", employeesTable, "Employees.LastName");
        lastName.DataBindings.Add(currentBinding);

        phoneBinding = new Binding("Text", employeesTable, "Employees.HomePhone");
        // We must add the event handlers before we bind, or the Format event will not get called
        // for the first record.
        phoneBinding.Format += new ConvertEventHandler(phoneBinding_Format);
        phoneBinding.Parse += new ConvertEventHandler(phoneBinding_Parse);
        phoneMask.DataBindings.Add(phoneBinding);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }
}

```

```

Dim WithEvents CurrentBinding, PhoneBinding As Binding
Dim EmployeesTable As New DataSet()
Dim sc As SqlConnection
Dim DataConnect As SqlDataAdapter

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    DoMaskedBinding()
End Sub

Private Sub DoMaskedBinding()
    Try
        sc = New SqlConnection("Data Source=SERVERNAME;Initial Catalog=NORTHWIND;Integrated
Security=SSPI")
        sc.Open()
    Catch ex As Exception
        MessageBox.Show(ex.Message)
        Exit Sub
    End Try

    DataConnect = New SqlDataAdapter("SELECT * FROM Employees", sc)
    DataConnect.Fill(EmployeesTable, "Employees")

    ' Now bind MaskedTextBox to appropriate field. Note that we must create the Binding objects
    ' before adding them to the control - otherwise, we won't get a Format event on the
    ' initial load.
    Try
        CurrentBinding = New Binding("Text", EmployeesTable, "Employees.FirstName")
        firstName.DataBindings.Add(CurrentBinding)
        CurrentBinding = New Binding("Text", EmployeesTable, "Employees.LastName")
        lastName.DataBindings.Add(CurrentBinding)
        PhoneBinding = New Binding("Text", EmployeesTable, "Employees.HomePhone")
        PhoneMask.DataBindings.Add(PhoneBinding)
    Catch ex As Exception
        MessageBox.Show(ex.Message)
        Application.Exit()
    End Try
End Sub

```

7. Add event handlers for the **Format** and **Parse** events to combine and separate the **PhoneNumber** and **Extension** fields from the bound **DataSet**.

```

private void phoneBinding_Format(Object sender, ConvertEventArgs e)
{
    String ext;

    DataRowView currentRow = (DataRowView)BindingContext[employeesTable, "Employees"].Current;
    if (currentRow["Extension"] == null)
    {
        ext = "";
    } else
    {
        ext = currentRow["Extension"].ToString();
    }

    e.Value = e.Value.ToString().Trim() + " x" + ext;
}

private void phoneBinding_Parse(Object sender, ConvertEventArgs e)
{
    String phoneNumberAndExt = e.Value.ToString();

    int extIndex = phoneNumberAndExt.IndexOf("x");
    String ext = phoneNumberAndExt.Substring(extIndex).Trim();
    String phoneNumber = phoneNumberAndExt.Substring(0, extIndex).Trim();

    //Get the current binding object, and set the new extension manually.
    DataRowView currentRow = (DataRowView)BindingContext[employeesTable, "Employees"].Current;
    // Remove the "x" from the extension.
    currentRow["Extension"] = ext.Substring(1);

    //Return the phone number.
    e.Value = phoneNumber;
}

```

```

Private Sub PhoneBinding_Format(ByVal sender As Object, ByVal e As ConvertEventArgs) Handles
PhoneBinding.Format
    Dim Ext As String

    Dim CurrentRow As DataRowView = CType(Me.BindingContext(EmployeesTable, "Employees").Current,
DataRowView)
    If (CurrentRow("Extension") Is Nothing) Then
        Ext = ""
    Else
        Ext = CurrentRow("Extension").ToString()
    End If

    e.Value = e.Value.ToString().Trim() & " x" & Ext
End Sub

Private Sub PhoneBinding_Parse(ByVal sender As Object, ByVal e As ConvertEventArgs) Handles
PhoneBinding.Parse
    Dim PhoneNumberAndExt As String = e.Value.ToString()

    Dim ExtIndex As Integer = PhoneNumberAndExt.IndexOf("x")
    Dim Ext As String = PhoneNumberAndExt.Substring(ExtIndex).Trim()
    Dim PhoneNumber As String = PhoneNumberAndExt.Substring(0, ExtIndex).Trim()

    ' Get the current binding object, and set the new extension manually.
    Dim CurrentRow As DataRowView = CType(Me.BindingContext(EmployeesTable, "Employees").Current,
DataRowView)
    ' Remove the "x" from the extension.
    CurrentRow("Extension") = CObj(Ext.Substring(1))

    ' Return the phone number.
    e.Value = PhoneNumber
End Sub

```

8. Add two **Button** controls to the form. Name them `previousButton` and `nextButton`. Double-click each button to add a **Click** event handler, and fill in the event handlers as shown in the following code example.

```
private void previousButton_Click(object sender, EventArgs e)
{
    BindingContext[employeesTable, "Employees"].Position = BindingContext[employeesTable,
"Employees"].Position - 1;
}

private void nextButton_Click(object sender, EventArgs e)
{
    BindingContext[employeesTable, "Employees"].Position = BindingContext[employeesTable,
"Employees"].Position + 1;
}
```

```
Private Sub PreviousButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
PreviousButton.Click
    Me.BindingContext(EmployeesTable, "Employees").Position = Me.BindingContext(EmployeesTable,
"Employees").Position - 1
End Sub

Private Sub NextButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
NextButton.Click
    Me.BindingContext(EmployeesTable, "Employees").Position = Me.BindingContext(EmployeesTable,
"Employees").Position + 1
End Sub
```

9. Run the sample. Edit the data, and use the **Previous** and **Next** buttons to see that the data is properly persisted to the **DataSet**.

Example

The following code example is the full code listing that results from completing the previous procedure.

```
#pragma region Using directives

#using <System.dll>
#using <System.Data.dll>
#using <System.Drawing.dll>
#using <System.Windows.Forms.dll>
#using <System.Xml.dll>
#using <System.EnterpriseServices.dll>
#using <System.Transactions.dll>

using namespace System;
using namespace System::Collections::Generic;
using namespace System::ComponentModel;
using namespace System::Data;
using namespace System::Drawing;
using namespace System::Windows::Forms;
using namespace System::Data::SqlClient;

#pragma endregion

namespace MaskedTextBoxDataCSharp
{
    public ref class Form1 : public Form
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
    private:
        System::.ComponentModel::.IContainer^ components;
```

```
System::ComponentModel::.IContainerComponents,
```

```
public:
```

```
    Form1()
    {
        InitializeComponent();
    }

    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
protected:
    ~Form1()
    {
        if (components != nullptr)
        {
            delete components;
        }
    }
```

```
#pragma region Windows Form Designer generated code
```

```
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
private:
    void InitializeComponent()
    {
        employeesTable = gcnew DataSet();
        components = nullptr;
        this->firstName = gcnew System::Windows::Forms::TextBox();
        this->lastName = gcnew System::Windows::Forms::TextBox();
        this->phoneMask = gcnew System::Windows::Forms::MaskedTextBox();
        this->previousButton = gcnew System::Windows::Forms::Button();
        this->nextButton = gcnew System::Windows::Forms::Button();
        this->SuspendLayout();
        //
        // firstName
        //
        this->firstName->Location = System::Drawing::Point(13, 14);
        this->firstName->Name = "firstName";
        this->firstName->Size = System::Drawing::Size(184, 20);
        this->firstName->TabIndex = 0;
        //
        // lastName
        //
        this->lastName->Location = System::Drawing::Point(204, 14);
        this->lastName->Name = "lastName";
        this->lastName->Size = System::Drawing::Size(184, 20);
        this->lastName->TabIndex = 1;
        //
        // phoneMask
        //
        this->phoneMask->Location = System::Drawing::Point(441, 14);
        this->phoneMask->Mask = "(009) 000-0000 x9999";
        this->phoneMask->Name = "phoneMask";
        this->phoneMask->Size = System::Drawing::Size(169, 20);
        this->phoneMask->TabIndex = 2;
        //
        // previousButton
        //
        this->previousButton->Location = System::Drawing::Point(630, 14);
        this->previousButton->Name = "previousButton";
        this->previousButton->TabIndex = 3;
        this->previousButton->Text = "Previous";
        this->previousButton->Click += gcnew System::EventHandler(this,&Form1::previousButton_Click);
    }
```

```

// nextButton
//
this->nextButton->Location = System::Drawing::Point(723, 14);
this->nextButton->Name = "nextButton";
this->nextButton->TabIndex = 4;
this->nextButton->Text = "Next";
this->nextButton->Click += gcnew System::EventHandler(this,&Form1::nextButton_Click);
//
// Form1
//
this->AutoScaleBaseSize = System::Drawing::Size(5, 13);
this->ClientSize = System::Drawing::Size(887, 46);
this->Controls->Add(this->nextButton);
this->Controls->Add(this->previousButton);
this->Controls->Add(this->phoneMask);
this->Controls->Add(this->lastName);
this->Controls->Add(this->firstName);
this->Name = "Form1";
this->Text = "Form1";
this->Load += gcnew System::EventHandler(this,&Form1::Form1_Load);
this->ResumeLayout(false);
this->PerformLayout();
}

#pragma endregion

private:
System::Windows::Forms::TextBox^ firstName;
System::Windows::Forms::TextBox^ lastName;
System::Windows::Forms::MaskedTextBox^ phoneMask;
System::Windows::Forms::Button^ previousButton;
System::Windows::Forms::Button^ nextButton;

private:
Binding^ currentBinding;
Binding^ phoneBinding;
DataSet^ employeesTable;
SqlConnection^ sc;
SqlDataAdapter^ dataConnect;

private:
void Form1_Load(Object^ sender, EventArgs^ e)
{
    DoMaskBinding();
}

private:
void DoMaskBinding()
{
try
{
    sc = gcnew SqlConnection("Data Source=localhost;" +
        "Initial Catalog=NORTHWIND;Integrated Security=SSPI");
    sc->Open();
}
catch (Exception^ ex)
{
    MessageBox::Show(ex->Message);
    return;
}

dataConnect = gcnew SqlDataAdapter("SELECT * FROM Employees", sc);
dataConnect->Fill(employeesTable, "Employees");

// Now bind MaskedTextBox to appropriate field. Note that we must
// create the Binding objects before adding them to the control -
// otherwise, we won't get a Format event on the initial load.

```

```

try
{
    currentBinding = gcnew Binding("Text", employeesTable,
        "Employees.FirstName");
    firstName->DataBindings->Add(currentBinding);

    currentBinding = gcnew Binding("Text", employeesTable,
        "Employees.LastName");
    lastName->DataBindings->Add(currentBinding);

    phoneBinding = gcnew Binding("Text", employeesTable,
        "Employees.HomePhone");
    // We must add the event handlers before we bind, or the
    // Format event will not get called for the first record.
    phoneBinding->Format += gcnew
        ConvertEventHandler(this, &Form1::phoneBinding_Format);
    phoneBinding->Parse += gcnew
        ConvertEventHandler(this, &Form1::phoneBinding_Parse);
    phoneMask->DataBindings->Add(phoneBinding);
}

catch (Exception^ ex)
{
    MessageBox::Show(ex->Message);
    return;
}

private:
void phoneBinding_Format(Object^ sender, ConvertEventArgs^ e)
{
    String^ ext;

    DataRowView^ currentRow = (DataRowView^) BindingContext[
        employeesTable, "Employees"]->Current;
    if (currentRow["Extension"] == nullptr)
    {
        ext = "";
    }
    else
    {
        ext = currentRow["Extension"]->ToString();
    }

    e->Value = e->Value->ToString()->Trim() + " x" + ext;
}

private:
void phoneBinding_Parse(Object^ sender, ConvertEventArgs^ e)
{
    String^ phoneNumberAndExt = e->Value->ToString();

    int extIndex = phoneNumberAndExt->IndexOf("x");
    String^ ext = phoneNumberAndExt->Substring(extIndex)->Trim();
    String^ phoneNumber =
        phoneNumberAndExt->Substring(0, extIndex)->Trim();

    //Get the current binding object, and set the new extension
    //manually.
    DataRowView^ currentRow =
        (DataRowView^) BindingContext[employeesTable,
        "Employees"]->Current;
    // Remove the "x" from the extension.
    currentRow["Extension"] = ext->Substring(1);

    //Return the phone number.
    e->Value = phoneNumber;
}

private:

```

```

void previousButton_Click(Object^ sender, EventArgs^ e)
{
    BindingContext[employeesTable, "Employees"]->Position =
        BindingContext[employeesTable, "Employees"]->Position - 1;
}

private:
    void nextButton_Click(Object^ sender, EventArgs^ e)
    {
        BindingContext[employeesTable, "Employees"]->Position =
            BindingContext[employeesTable, "Employees"]->Position + 1;
    }
};

[STAThread]
int main()
{
    Application::EnableVisualStyles();
    Application::Run(gcnew MaskedTextBoxDataCSharp::Form1());
}

```

```

#region Using directives

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Windows.Forms;
using System.Data.SqlClient;

#endregion

namespace MaskedTextBoxDataCSharp
{
    partial class Form1 : Form
    {
        Binding currentBinding, phoneBinding;
        DataSet employeesTable = new DataSet();
        SqlConnection sc;
        SqlDataAdapter dataConnect;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            DoMaskBinding();
        }

        private void DoMaskBinding()
        {
            try
            {
                sc = new SqlConnection("Data Source=localhost;Initial Catalog=NORTHWIND;Integrated Security=SSPI");
                sc.Open();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
                return;
            }
        }
    }
}

```

```

dataConnect = new SqlDataAdapter("SELECT * FROM Employees", sc);
dataConnect.Fill(employeesTable, "Employees");

// Now bind MaskedTextBox to appropriate field. Note that we must create the Binding objects
// before adding them to the control - otherwise, we won't get a Format event on the
// initial load.
try
{
    currentBinding = new Binding("Text", employeesTable, "Employees.FirstName");
    firstName.DataBindings.Add(currentBinding);

    currentBinding = new Binding("Text", employeesTable, "Employees.LastName");
    lastName.DataBindings.Add(currentBinding);

    phoneBinding = new Binding("Text", employeesTable, "Employees.HomePhone");
    // We must add the event handlers before we bind, or the Format event will not get called
    // for the first record.
    phoneBinding.Format += new ConvertEventHandler(phoneBinding_Format);
    phoneBinding.Parse += new ConvertEventHandler(phoneBinding_Parse);
    phoneMask.DataBindings.Add(phoneBinding);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
    return;
}

private void phoneBinding_Format(Object sender, ConvertEventArgs e)
{
    String ext;

    DataRowView currentRow = (DataRowView)BindingContext[employeesTable, "Employees"].Current;
    if (currentRow["Extension"] == null)
    {
        ext = "";
    }
    else
    {
        ext = currentRow["Extension"].ToString();
    }

    e.Value = e.Value.ToString().Trim() + " x" + ext;
}

private void phoneBinding_Parse(Object sender, ConvertEventArgs e)
{
    String phoneNumberAndExt = e.Value.ToString();

    int extIndex = phoneNumberAndExt.IndexOf("x");
    String ext = phoneNumberAndExt.Substring(extIndex).Trim();
    String phoneNumber = phoneNumberAndExt.Substring(0, extIndex).Trim();

    //Get the current binding object, and set the new extension manually.
    DataRowView currentRow = (DataRowView)BindingContext[employeesTable, "Employees"].Current;
    // Remove the "x" from the extension.
    currentRow["Extension"] = ext.Substring(1);

    //Return the phone number.
    e.Value = phoneNumber;
}

private void previousButton_Click(object sender, EventArgs e)
{
    BindingContext[employeesTable, "Employees"].Position = BindingContext[employeesTable,
    "Employees"].Position - 1;
}

private void nextButton_Click(object sender, EventArgs e)

```

```

    {
        BindingContext[employeesTable, "Employees"].Position = BindingContext[employeesTable,
"Employees"].Position + 1;
    }
}
}

```

```

Imports System.Data.SqlClient

Public Class Form1
    Dim WithEvents CurrentBinding, PhoneBinding As Binding
    Dim EmployeesTable As New DataSet()
    Dim sc As SqlConnection
    Dim DataConnect As SqlDataAdapter

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        DoMaskedBinding()
    End Sub

    Private Sub DoMaskedBinding()
        Try
            sc = New SqlConnection("Data Source=localhost;Initial Catalog=NORTHWIND;Integrated Security=SSPI")
            sc.Open()
        Catch ex As Exception
            MessageBox.Show(ex.Message)
            Application.Exit()
        End Try

        DataConnect = New SqlDataAdapter("SELECT * FROM Employees", sc)
        DataConnect.Fill(EmployeesTable, "Employees")

        ' Now bind MaskedTextBox to appropriate field. Note that we must create the Binding objects
        ' before adding them to the control - otherwise, we won't get a Format event on the
        ' initial load.
        Try
            CurrentBinding = New Binding("Text", EmployeesTable, "Employees.FirstName")
            firstName.DataBindings.Add(CurrentBinding)
            CurrentBinding = New Binding("Text", EmployeesTable, "Employees.LastName")
            lastName.DataBindings.Add(CurrentBinding)
            PhoneBinding = New Binding("Text", EmployeesTable, "Employees.HomePhone")
            PhoneMask.DataBindings.Add(PhoneBinding)
        Catch ex As Exception
            MessageBox.Show(ex.Message)
            Application.Exit()
        End Try
    End Sub

    Private Sub PhoneBinding_Format(ByVal sender As Object, ByVal e As ConvertEventArgs) Handles
PhoneBinding.Format
        Dim Ext As String

        Dim CurrentRow As DataRowView = CType(Me.BindingContext(EmployeesTable, "Employees").Current,
DataRowView)
        If (CurrentRow("Extension") Is Nothing) Then
            Ext = ""
        Else
            Ext = CurrentRow("Extension").ToString()
        End If

        e.Value = e.Value.ToString().Trim() & " x" & Ext
    End Sub

    Private Sub PhoneBinding_Parse(ByVal sender As Object, ByVal e As ConvertEventArgs) Handles
PhoneBinding.Parse
        Dim PhoneNumberAndExt As String = e.Value.ToString()

        Dim ExtIndex As Integer = PhoneNumberAndExt.IndexOf("x")

```

```

Dim Ext As String = PhoneNumberAndExt.Substring(ExtIndex).Trim()
Dim PhoneNumber As String = PhoneNumberAndExt.Substring(0, ExtIndex).Trim()

' Get the current binding object, and set the new extension manually.
Dim CurrentRow As DataRowView = CType(Me.BindingContext(EmployeesTable, "Employees").Current,
DataRowView)
' Remove the "x" from the extension.
CurrentRow("Extension") = CObj(Ext.Substring(1))

' Return the phone number.
e.Value = PhoneNumber
End Sub

Private Sub NextButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
NextButton.Click
    Me.BindingContext(EmployeesTable, "Employees").Position = Me.BindingContext(EmployeesTable,
"Employees").Position + 1
End Sub

Private Sub PreviousButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
PreviousButton.Click
    Me.BindingContext(EmployeesTable, "Employees").Position = Me.BindingContext(EmployeesTable,
"Employees").Position - 1
End Sub
End Class

```

Compiling the Code

- Create a Visual C# or Visual Basic project.
- Add the [TextBox](#) and [MaskedTextBox](#) controls to the form, as described in the previous procedure.
- Open the source code file for the project's default form.
- Replace the source code in this file with the code listed in the previous "Code" section.
- Compile the application.

See Also

[Walkthrough: Working with the MaskedTextBox Control](#)

How to: Set the Input Mask

5/4/2018 • 1 min to read • [Edit Online](#)

The masked text box control is an enhanced text box control that supports a declarative syntax for accepting or rejecting user input. By setting the **Mask** property, you can specify the allowable user input without writing any custom validation logic in your application. For more information, see the Remarks section of the [MaskedTextBox](#) class.

Setting the Mask Property Manually

If you are familiar with the characters that the **Mask** property supports, you can enter it manually. For a summary of the characters that the **Mask** property supports, see the Remarks section of the [Mask](#) property.

To set the **Mask** property manually

1. In **Design** view, select a [MaskedTextBox](#).
2. In the **Properties** window, locate the [Mask](#) property.
3. Type the mask that you want. For example, type `###`.

Using the Input Mask Dialog Box

The Input Mask dialog box provides some predefined input masks. You can also change the predefined masks or enter your own mask manually.

To open the Input Mask dialog box

1. In **Design** view, select a [MaskedTextBox](#).
 - a. Click the smart tag to open the **MaskedTextBox Tasks** panel.
 - b. Click **Set Mask**.

- or -

 - a. In the **Properties** window, select the [Mask](#) property.
 - b. Click the ellipsis button in the property value column.

The **Input Mask** dialog box appears.

To use the Input Mask dialog box

1. (Optional) Click one of the predefined masks in the list.
2. (Optional) Edit the predefined mask in the **Mask** box.
3. (Optional) Type a new mask in the **Mask** box. That is, you do not have to use one of the predefined masks.

NOTE

The Preview box displays the characters that the user sees in the [MaskedTextBox](#). These characters are a guide to help the user enter the data correctly.

4. Select or clear the **Use ValidatingType** check box. The **Use ValidatingType** check box specifies whether a data type is used to verify the data input by the user. For more information, see the [ValidatingType](#) property.

5. Click **OK**.

The mask is entered in the **Mask** property in the **Properties** window.

See Also

[Walkthrough: Working with the MaskedTextBox Control](#)

MenuStrip Control (Windows Forms)

5/4/2018 • 2 min to read • [Edit Online](#)

This control groups application commands and makes them easily accessible.

In This Section

[MenuStrip Control Overview](#)

Explains what the control is and its key features and properties.

[How to: Add Enhancements to ToolStripMenuItems](#)

Describes how to add check marks, images, shortcut keys, access keys, and separator bars to menus and menu commands.

[How to: Append a MenuStrip to an MDI Parent Window](#)

Describes how to set several properties to append the multiple-document interface (MDI) child menu to the MDI parent menu.

[How to: Create an MDI Window List with MenuStrip](#)

Demonstrates how to create a list of all the active child forms on the parent's Window menu.

[How to: Disable ToolStripMenuItems](#)

Describes how to disable both entire menus and individual menu commands.

[How to: Hide ToolStripMenuItems](#)

Describes how to hide both entire menus and individual menu commands.

[How to: Insert a MenuStrip into an MDI Drop-Down Menu](#)

Describes how to set several properties to insert a group of menu items from the MDI child menu into the drop-down part of the MDI parent menu.

[How to: Remove a ToolStripMenuItem from an MDI Drop-Down Menu](#)

Describes how to set several properties to remove a menu item from the drop-down part of the MDI parent menu.

[How to: Configure MenuStrip Check Margins and Image Margins](#)

Describes how to customize a [MenuStrip](#) by setting check and image margin properties in various ways.

[How to: Provide Standard Menu Items to a Form](#)

Describes how to use a [MenuStrip](#) control to create a form with a standard menu.

[How to: Display Option Buttons in a MenuStrip](#)

Describes how to implement option-button (or radio-button) behavior in a [ToolStripMenuItem](#).

[Merging Menu Items in the Windows Forms MenuStrip Control](#)

Describes general concepts and methods for menu merging.

[How to: Set Up Automatic Menu Merging for MDI Applications](#)

Describes how to merge menu items automatically at run time.

- [MenuStrip Items Collection Editor](#)
- [How to: Copy ToolStripMenuItems](#)
- [How to: Hide ToolStripMenuItems Using the Designer](#)

- [How to: Disable ToolStripMenuItems Using the Designer](#)
- [How to: Move ToolStripMenuItems](#)
- [Walkthrough: Providing Standard Menu Items to a Form](#)
- [MenuStrip Tasks Dialog Box](#)

Reference

[MenuStrip](#)

Describes the features of the [MenuStrip](#) class, which provides a menu system for a form.

[ContextMenuStrip](#)

Describes the features of the [ContextMenuStrip](#), which represents a shortcut menu.

[ToolStripMenuItem](#)

Describes the features of the [ToolStripMenuItem](#) class, which represents a selectable option displayed on a [MenuStrip](#) or [ContextMenuStrip](#).

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

MenuStrip Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

Menus expose functionality to your users by holding commands that are grouped by a common theme.

The [MenuStrip](#) control is new to this version of Visual Studio and the .NET Framework. With the control, you can easily create menus like those found in Microsoft Office.

The [MenuStrip](#) control supports the multiple-document interface (MDI) and menu merging, tool tips, and overflow. You can enhance the usability and readability of your menus by adding access keys, shortcut keys, check marks, images, and separator bars.

The [MenuStrip](#) control replaces and adds functionality to the [MainMenu](#) control; however, the [MainMenu](#) control is retained for backward compatibility and future use if you choose.

Ways to Use the MenuStrip Control

Use the [MenuStrip](#) control to:

- Create easily customized, commonly employed menus that support advanced user interface and layout features, such as text and image ordering and alignment, drag-and-drop operations, MDI, overflow, and alternate modes of accessing menu commands.
- Support the typical appearance and behavior of the operating system.
- Handle events consistently for all containers and contained items, in the same way you handle events for other controls.

The following table shows some particularly important properties of [MenuStrip](#) and associated classes.

PROPERTY	DESCRIPTION
MdiWindowListItem	Gets or sets the ToolStripMenuItem that is used to display a list of MDI child forms.
ToolStripItem.MergeAction	Gets or sets how child menus are merged with parent menus in MDI applications.
ToolStripItem.MergeIndex	Gets or sets the position of a merged item within a menu in MDI applications.
Form.IsMdiContainer	Gets or sets a value indicating whether the form is a container for MDI child forms.
ShowItemToolTips	Gets or sets a value indicating whether tool tips are shown for the MenuStrip .
CanOverflow	Gets or sets a value indicating whether the MenuStrip supports overflow functionality.
ShortcutKeys	Gets or sets the shortcut keys associated with the ToolStripMenuItem .

PROPERTY	DESCRIPTION
ShowShortcutKeys	Gets or sets a value indicating whether the shortcut keys that are associated with the ToolStripMenuItem are displayed next to the ToolStripMenuItem .

The following table shows the important [MenuStrip](#) companion classes.

CLASS	DESCRIPTION
ToolStripMenuItem	Represents a selectable option displayed on a MenuStrip or ContextMenuStrip .
ContextMenuStrip	Represents a shortcut menu.
ToolStripDropDown	Represents a control that allows the user to select a single item from a list that is displayed when the user clicks a ToolStripDropDownButton or a higher-level menu item.
ToolStripDropDownItem	Provides basic functionality for controls derived from ToolStripItem that display drop-down items when clicked.

See Also

[ToolStrip](#)
[MenuStrip](#)
[ContextMenuStrip](#)
[StatusStrip](#)
[ToolStripItem](#)
[ToolStripDropDown](#)

How to: Add Enhancements to ToolStripMenuItem

5/4/2018 • 2 min to read • [Edit Online](#)

You can enhance the usability of [MenuStrip](#) and [ContextMenuStrip](#) controls in the following ways:

- Add check marks to designate whether a feature is turned on or off, such as whether a ruler is displayed along the margin of a word-processing application, or to indicate which file in a list of files is being displayed, such as on a **Window** menu.
- Add images that visually represent menu commands.
- Display shortcut keys to provide a keyboard alternative to the mouse for performing commands. For example, pressing CTRL+C performs the **Copy** command.
- Display access keys to provide a keyboard alternative to the mouse for menu navigation. For example, pressing ALT+F chooses the **File** menu.
- Show separator bars to group related commands and make menus more readable.

To display a check mark on a menu command

- Set its [Checked](#) property to `true`.

This also sets the [CheckState](#) property to `true`. Use this procedure only if you want the menu command to appear as checked by default, regardless of whether it is selected.

To display a check mark that changes state with each click

- Set the menu command's [CheckOnClick](#) property to `true`.

To add an image to a menu command

- Set the menu command's [Image](#) property to the name of the image. If the [ToolStripItemDisplayStyle](#) property of this menu command is set to [Text](#) or [None](#), the image cannot be displayed.

NOTE

The image margin can also show a check mark if you so choose. Also, you can set the [Checked](#) property of the image to `true`, and the image will appear with a hatched border around it at run time.

To display a shortcut key for a menu command

- Set the menu command's [ShortcutKeys](#) property to the desired keyboard combination, such as CTRL+O for the **Open** menu command, and set the [ShowShortcutKeys](#) property to `true`.

To display custom shortcut keys for a menu command

- Set the menu command's [ShortcutKeyDisplayString](#) property to the desired keyboard combination, such as CTRL+SHIFT+O rather than SHIFT+CTRL+O, and set the [ShowShortcutKeys](#) property to `true`.

To display an access key for a menu command

- When you set the [Text](#) property for the menu command, enter an ampersand (&) before the letter you want to be underlined as the access key. For example, typing `&open` as the [Text](#) property of a menu item will result in a menu command that appears as **Open**.

To navigate to this menu command, press ALT to give focus to the [MenuStrip](#), and press the access key of the menu name. When the menu opens and shows items with access keys, you only need to press the access

key to select the menu command.

NOTE

Avoid defining duplicate access keys, such as defining ALT+F twice in the same menu system. The selection order of duplicate access keys cannot be guaranteed.

To display a separator bar between menu commands

- After you define your [MenuStrip](#) and the items it will contain, use the [AddRange](#) or [Add](#) method to add the menu commands and [ToolStripSeparator](#) controls to the [MenuStrip](#) in the order you want.

```
' This code adds a top-level File menu to the MenuStrip.  
Me.menuStrip1.Items.Add(New ToolStripMenuItem() _  
{Me.fileToolStripMenuItem})  
  
' This code adds the New and Open menu commands, a separator bar,  
' and the Save and Exit menu commands to the top-level File menu,  
' in that order.  
Me.fileToolStripMenuItem.DropDownItems.AddRange(New _  
ToolStripMenuItem() {Me.newToolStripMenuItem, _  
Me.openToolStripMenuItem, Me.toolStripSeparator1, _  
Me.saveToolStripMenuItem, Me.exitToolStripMenuItem})
```

```
// This code adds a top-level File menu to the MenuStrip.  
this.menuStrip1.Items.Add(new ToolStripItem[]_  
{this.fileToolStripMenuItem});  
  
// This code adds the New and Open menu commands, a separator bar,  
// and the Save and Exit menu commands to the top-level File menu,  
// in that order.  
this.fileToolStripMenuItem.DropDownItems.AddRange(new _  
ToolStripItem[] {  
this.newToolStripMenuItem,  
this.openToolStripMenuItem,  
this.toolStripSeparator1,  
this.saveToolStripMenuItem,  
this.exitToolStripMenuItem});
```

See Also

[MenuStrip](#)

[ToolStripMenuItem](#)

[MenuStrip Control Overview](#)

How to: Append a MenuStrip to an MDI Parent Window (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

In some applications, the kind of a multiple-document interface (MDI) child window can be different from the MDI parent window. For example, the MDI parent might be a spreadsheet, and the MDI child might be a chart. In that case, you want to update the contents of the MDI parent's menu with the contents of the MDI child's menu as MDI child windows of different kinds are activated.

The following procedure uses the `IsMdiContainer`, `AllowMerge`, `MergeAction`, and `MergeIndex` properties to append the MDI child menu to the MDI parent menu. Closing the MDI child window removes the appended menu from the MDI parent.

Also see [Multiple-Document Interface \(MDI\) Applications](#).

To append a menu item to an MDI parent

1. Create a form and set its `IsMdiContainer` property to `true`.
2. Add a `MenuStrip` to `Form1` and set the `AllowMerge` property of the `MenuStrip` to `true`.
3. Set the `Visible` property of the `Form1` `MenuStrip` to `false`.
4. Add a top-level menu item to the `Form1` `MenuStrip` and set its `Text` property to `&File`.
5. Add a submenu item to the `&File` menu item and set its `Text` property to `&Open`.
6. Add a form to the project, add a `MenuStrip` to the form, and set the `AllowMerge` property of the `Form2` `MenuStrip` to `true`.
7. Add a top-level menu item to the `Form2` `MenuStrip` and set its `Text` property to `&Special`.
8. Add two submenu items to the `&special` menu item and set their `Text` properties to `Command&1` and `Command&2`, respectively.
9. Set the `MergeAction` property of the `&Special`, `Command&1`, and `Command&2` menu items to `Append`.
10. Create an event handler for the `Click` event of the `&New ToolStripMenuItem`.
11. Within the event handler, insert code similar to the following code example to create and display new instances of `Form2` as MDI children of `Form1`.

```
Private Sub openToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles openToolStripMenuItem.Click
    Dim NewMDIChild As New Form2()
    'Set the parent form of the child window.
    NewMDIChild.MdiParent = Me
    'Display the new form.
    NewMDIChild.Show()
End Sub
```

```
private void openToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form2 newMDIChild = new Form2();
    // Set the parent form of the child window.
    newMDIChild.MdiParent = this;
    // Display the new form.
    newMDIChild.Show();
}
```

12. Place code similar to the following code example in the `&Open ToolStripMenuItem` to register the event handler.

```
Private Sub openToolStripMenuItem_Click(sender As Object, e As _
EventArgs) Handles openToolStripMenuItem.Click
```

```
this.openToolStripMenuItem.Click += new System.EventHandler(this.openToolStripMenuItem_Click);
```

Compiling the Code

This example requires:

- Two `Form` controls named `Form1` and `Form2`.
- A `MenuStrip` control on `Form1` named `menuStrip1`, and a `MenuStrip` control on `Form2` named `menuStrip2`.
- References to the `System` and `System.Windows.Forms` assemblies.

How to: Copy ToolStripMenuItem

5/4/2018 • 1 min to read • [Edit Online](#)

At design time, you can copy entire top-level menus and their submenu items to a different place on the [MenuStrip](#). You can also copy individual menu items between top-level menus or change the position of menu items within a menu.

To copy a top-level menu and its submenu items to another top-level location

1. Left-click the menu that you want to copy and press CTRL+C, or right-click the menu and select **Copy** from the shortcut menu.
2. Left-click the top-level menu that is after the intended new location and press CTRL+V, or right-click the top-level menu item that is before the intended new location and select **Paste** from the shortcut menu.

The menu that you copied is inserted before the selected top-level menu.

To copy a top-level menu and its submenu items to a drop-down location

1. Left-click the menu that you want to move and press CTRL+C, or right-click the menu and select **Copy** from the shortcut menu.
2. In the destination top-level menu, left-click the submenu item that is above the intended new location and press CTRL+V, or right-click the submenu item that is above the intended new location and select **Paste** from the shortcut menu.

The menu that you copied is inserted before the selected submenu item.

To copy a submenu item to another menu

1. Left-click the submenu item that you want to copy and press CTRL+C, or right-click the submenu item and choose **Copy** from the shortcut menu.
2. Left-click the menu that will contain the submenu item that you cut.
3. Left-click the submenu item that is before the intended new location and press CTRL+V, or right-click the submenu item that is before the intended new location and select **Paste** from the shortcut menu.

The submenu item that you copied is inserted before the selected submenu item.

See Also

[MenuStrip](#)

[ToolStripMenuItem](#)

[MenuStrip Control Overview](#)

How to: Create an MDI Window List with MenuStrip (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

Use the multiple-document interface (MDI) to create applications that can open several documents at the same time and copy and paste content from one document to the other.

This procedure shows you how to create a list of all the active child forms on the parent's Window menu.

To create an MDI Window list on a MenuStrip

1. Create a form and set its `IsMdiContainer` property to `true`.
2. Add a `MenuStrip` to the form.
3. Add two top-level menu items to the `MenuStrip` and set their `Text` properties to `&File` and `&Window`.
4. Add a submenu item to the `&File` menu item and set its `Text` property to `&Open`.
5. Set the `MdiWindowListItem` property of the `MenuStrip` to the `&Window ToolStripMenuItem`.
6. Add a form to the project and add the control you want to it, such as another `MenuStrip`.
7. Create an event handler for the `Click` event of the `&New ToolStripMenuItem`.
8. Within the event handler, insert code similar to the following to create and display new instances of `Form2` as MDI children of `Form1`.

```
Private Sub openToolStripMenuItem_Click(ByVal sender As _
System.Object, ByVal e As System.EventArgs) Handles _
openToolStripMenuItem.Click
    Dim NewMDIChild As New Form2()
    'Set the parent form of the child window.
    NewMDIChild.MdiParent = Me
    'Display the new form.
    NewMDIChild.Show()
End Sub
```

```
private void newToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form2 newMDIChild = new Form2();
    // Set the parent form of the child window.
    newMDIChild.MdiParent = this;
    // Display the new form.
    newMDIChild.Show();
}
```

9. Place code like the following in the `&New ToolStripMenuItem` to register the event handler.

```
Private Sub newToolStripMenuItem_Click(sender As Object, e As _
EventArgs) Handles newToolStripMenuItem.Click
```

```
this.newToolStripMenuItem.Click += new System.EventHandler(this.newToolStripMenuItem_Click);
```

Compiling the Code

This example requires:

- Two `Form` controls named `Form1` and `Form2`.
- A `MenuStrip` control on `Form1` named `menuStrip1`, and a `MenuStrip` control on `Form2` named `menuStrip2`.
- References to the `System` and `System.Windows.Forms` assemblies.

See Also

[How to: Create MDI Parent Forms](#)

[How to: Create MDI Child Forms](#)

[MenuStrip Control](#)

How to: Disable ToolStripMenuItems

5/4/2018 • 1 min to read • [Edit Online](#)

You can limit or broaden the commands a user may make by enabling and disabling menu items in response to user activities. Menu items are enabled by default when they are created, but this can be adjusted through the **Enabled** property. You can manipulate this property at design time in the **Properties** window or programmatically by setting it in code.

To disable a menu item programmatically

- Within the method where you set the properties of the menu item, add code to set the **Enabled** property to `false`.

```
MenuItem1.Enabled = False
```

```
menuItem1.Enabled = false;
```

```
menuItem1->Enabled = false;
```

TIP

Disabling the first or top-level menu item in a menu hides all the menu items contained within the menu, but does not disable them. Likewise, disabling a menu item that has submenu items hides the submenu items, but does not disable them. If all the commands on a given menu are unavailable to the user, it is considered good programming practice to both hide and disable the entire menu, as this presents a clean user interface. You should hide and disable the menu, and disable every item and submenu item in the menu, because hiding alone does not prevent access to a menu command via a shortcut key. Set the **Visible** property of a top-level menu item to `false` to hide the entire menu.

See Also

[MenuStrip](#)

[ToolStripMenuItem](#)

[How to: Hide ToolStripMenuItems](#)

[MenuStrip Control Overview](#)

How to: Disable ToolStripMenuItem Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

You can limit or broaden the commands a user may make by enabling and disabling menu items in response to user activities. Menu items are enabled by default when they are created, but this can be adjusted through the **Enabled** property. You can manipulate this property at design time in the **Properties** window or programmatically by setting it in code. For more information, see [How to: Disable ToolStripMenuItem](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To disable a menu item at design time

1. With the menu item selected on the form, set the **Enabled** property to `false`.

TIP

Disabling the first or top-level menu item in a menu disables all the menu items contained within the menu. Likewise, disabling a menu item that has submenu items disables the submenu items. If all the commands on a given menu are unavailable to the user, it is considered good programming practice to both hide and disable the entire menu, as this presents a clean user interface. You should both hide and disable the menu, as hiding alone does not prevent access to a menu command via a shortcut key. Set the **Visible** property of a top-level menu item to `false` to hide the entire menu.

See Also

[MenuStrip](#)

[ToolStripMenuItem](#)

[How to: Hide ToolStripMenuItem](#)

[MenuStrip Control Overview](#)

How to: Display Option Buttons in a ToolStrip (Windows Forms)

5/4/2018 • 17 min to read • [Edit Online](#)

Option buttons, also known as radio buttons, are similar to check boxes except that users can select only one at a time. Although by default the [ToolStripMenuItem](#) class does not provide option-button behavior, the class does provide check-box behavior that you can customize to implement option-button behavior for menu items in a [MenuStrip](#) control.

When the [CheckOnClick](#) property of a menu item is `true`, users can click the item to toggle the display of a check mark. The [Checked](#) property indicates the current state of the item. To implement basic option-button behavior, you must ensure that when an item is selected, you set the [Checked](#) property for the previously selected item to `false`.

The following procedures describe how to implement this and additional functionality in a class that inherits the [ToolStripMenuItem](#) class. The [ToolStripRadioButtonMenuItem](#) class overrides members such as [OnCheckedChanged](#) and [OnPaint](#) to provide the selection behavior and appearance of option buttons. Additionally, this class overrides the [Enabled](#) property so that options on a submenu are disabled unless the parent item is selected.

To implement option-button selection behavior

1. Initialize the [CheckOnClick](#) property to `true` to enable item selection.

```
// Called by all constructors to initialize CheckOnClick.  
private void Initialize()  
{  
    CheckOnClick = true;  
}
```

```
' Called by all constructors to initialize CheckOnClick.  
Private Sub Initialize()  
    CheckOnClick = True  
End Sub
```

2. Override the [OnCheckedChanged](#) method to clear the selection of the previously selected item when a new item is selected.

```

protected override void OnCheckedChanged(EventArgs e)
{
    base.OnCheckedChanged(e);

    // If this item is no longer in the checked state or if its
    // parent has not yet been initialized, do nothing.
    if (!Checked || this.Parent == null) return;

    // Clear the checked state for all siblings.
    foreach (ToolStripItem item in Parent.Items)
    {
        ToolStripRadioButtonMenuItem radioItem =
            item as ToolStripRadioButtonMenuItem;
        if (radioItem != null && radioItem != this && radioItem.Checked)
        {
            radioItem.Checked = false;

            // Only one item can be selected at a time,
            // so there is no need to continue.
            return;
        }
    }
}

```

```

Protected Overrides Sub OnCheckedChanged(ByVal e As EventArgs)

    MyBase.OnCheckedChanged(e)

    ' If this item is no longer in the checked state or if its
    ' parent has not yet been initialized, do nothing.
    If Not Checked OrElse Me.Parent Is Nothing Then Return

    ' Clear the checked state for all siblings.
    For Each item As ToolStripItem In Parent.Items

        Dim radioItem As ToolStripRadioButtonMenuItem = _
            TryCast(item, ToolStripRadioButtonMenuItem)
        If radioItem IsNot Nothing AndAlso _
            radioItem IsNot Me AndAlso _
            radioItem.Checked Then

            radioItem.Checked = False

            ' Only one item can be selected at a time,
            ' so there is no need to continue.
            Return

        End If
    Next

End Sub

```

3. Override the **OnClick** method to ensure that clicking an item that has already been selected will not clear the selection.

```
protected override void OnClick(EventArgs e)
{
    // If the item is already in the checked state, do not call
    // the base method, which would toggle the value.
    if (Checked) return;

    base.OnClick(e);
}
```

```
Protected Overrides Sub OnClick(ByVal e As EventArgs)

    ' If the item is already in the checked state, do not call
    ' the base method, which would toggle the value.
    If Checked Then Return

    MyBase.OnClick(e)
End Sub
```

To modify the appearance of the option-button items

1. Override the [OnPaint](#) method to replace the default check-mark with an option button by using the [RadioButtonRenderer](#) class.

```

// Let the item paint itself, and then paint the RadioButton
// where the check mark is normally displayed.
protected override void OnPaint(PaintEventArgs e)
{
    if (Image != null)
    {
        // If the client sets the Image property, the selection behavior
        // remains unchanged, but the RadioButton is not displayed and the
        // selection is indicated only by the selection rectangle.
        base.OnPaint(e);
        return;
    }
    else
    {
        // If the Image property is not set, call the base OnPaint method
        // with the CheckState property temporarily cleared to prevent
        // the check mark from being painted.
        CheckState currentState = this.CheckState;
        this.CheckState = CheckState.Unchecked;
        base.OnPaint(e);
        this.CheckState = currentState;
    }

    // Determine the correct state of the RadioButton.
    RadioButtonState buttonState = RadioButtonState.UncheckedNormal;
    if (Enabled)
    {
        if (mouseDownState)
        {
            if (Checked) buttonState = RadioButtonState.CheckedPressed;
            else buttonState = RadioButtonState.UncheckedPressed;
        }
        else if (mouseHoverState)
        {
            if (Checked) buttonState = RadioButtonState.CheckedHot;
            else buttonState = RadioButtonState.UncheckedHot;
        }
        else
        {
            if (Checked) buttonState = RadioButtonState.CheckedNormal;
        }
    }
    else
    {
        if (Checked) buttonState = RadioButtonState.CheckedDisabled;
        else buttonState = RadioButtonState.UncheckedDisabled;
    }

    // Calculate the position at which to display the RadioButton.
    Int32 offset = (ContentRectangle.Height -
        RadioButtonRenderer.GetGlyphSize(
        e.Graphics, buttonState).Height) / 2;
    Point imageLocation = new Point(
        ContentRectangle.Location.X + 4,
        ContentRectangle.Location.Y + offset);

    // Paint the RadioButton.
    RadioButtonRenderer.DrawRadioButton(
        e.Graphics, imageLocation, buttonState);
}

```

```

' Let the item paint itself, and then paint the RadioButton
' where the check mark is normally displayed.
Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)

    If Image IsNot Nothing Then
        ' If the client sets the Image property, the selection behavior
        ' remains unchanged, but the RadioButton is not displayed and the
        ' selection is indicated only by the selection rectangle.
        MyBase.OnPaint(e)
        Return
    Else
        ' If the Image property is not set, call the base OnPaint method
        ' with the CheckState property temporarily cleared to prevent
        ' the check mark from being painted.
        Dim currentState As CheckState = Me.CheckState
        Me.CheckState = CheckState.Unchecked
        MyBase.OnPaint(e)
        Me.CheckState = currentState
    End If

    ' Determine the correct state of the RadioButton.
    Dim buttonState As RadioButtonState = RadioButtonState.UncheckedNormal
    If Enabled Then
        If mouseDownState Then
            If Checked Then
                buttonState = RadioButtonState.CheckedPressed
            Else
                buttonState = RadioButtonState.UncheckedPressed
            End If
        ElseIf mouseHoverState Then
            If Checked Then
                buttonState = RadioButtonState.CheckedHot
            Else
                buttonState = RadioButtonState.UncheckedHot
            End If
        Else
            If Checked Then buttonState = RadioButtonState.CheckedNormal
            End If
        Else
            If Checked Then
                buttonState = RadioButtonState.CheckedDisabled
            Else
                buttonState = RadioButtonState.UncheckedDisabled
            End If
        End If
    End If

    ' Calculate the position at which to display the RadioButton.
    Dim offset As Int32 = CInt((ContentRectangle.Height - _
        RadioButtonRenderer.GetGlyphSize( _
        e.Graphics, buttonState).Height) / 2)
    Dim imageLocation As Point = New Point( _
        ContentRectangle.Location.X + 4, _
        ContentRectangle.Location.Y + offset)

    ' Paint the RadioButton.
    RadioButtonRenderer.DrawRadioButton( _
        e.Graphics, imageLocation, buttonState)
End Sub

```

2. Override the [OnMouseEnter](#), [OnMouseLeave](#), [OnMouseDown](#), and [OnMouseUp](#) methods to track the state of the mouse and ensure that the [OnPaint](#) method paints the correct option-button state.

```
private bool mouseHoverState = false;

protected override void OnMouseEnter(EventArgs e)
{
    mouseHoverState = true;

    // Force the item to repaint with the new RadioButton state.
    Invalidate();

    base.OnMouseEnter(e);
}

protected override void OnMouseLeave(EventArgs e)
{
    mouseHoverState = false;
    base.OnMouseLeave(e);
}

private bool mouseDownState = false;

protected override void OnMouseDown(MouseEventArgs e)
{
    mouseDownState = true;

    // Force the item to repaint with the new RadioButton state.
    Invalidate();

    base.OnMouseDown(e);
}

protected override void OnMouseUp(MouseEventArgs e)
{
    mouseDownState = false;
    base.OnMouseUp(e);
}
```

```

Private mouseHoverState As Boolean = False

Protected Overrides Sub OnMouseEnter(ByVal e As EventArgs)
    mouseHoverState = True

    ' Force the item to repaint with the new RadioButton state.
    Invalidate()

    MyBase.OnMouseEnter(e)
End Sub

Protected Overrides Sub OnMouseLeave(ByVal e As EventArgs)
    mouseHoverState = False
    MyBase.OnMouseLeave(e)
End Sub

Private mouseDownState As Boolean = False

Protected Overrides Sub OnMouseDown(ByVal e As MouseEventArgs)
    mouseDownState = True

    ' Force the item to repaint with the new RadioButton state.
    Invalidate()

    MyBase.OnMouseDown(e)
End Sub

Protected Overrides Sub OnMouseUp(ByVal e As MouseEventArgs)
    mouseDownState = False
    MyBase.OnMouseUp(e)
End Sub

```

To disable options on a submenu when the parent item is not selected

- Override the [Enabled](#) property so that the item is disabled when it has a parent item with both a [CheckOnClick](#) value of `true` and a [Checked](#) value of `false`.

```

// Enable the item only if its parent item is in the checked state
// and its Enabled property has not been explicitly set to false.
public override bool Enabled
{
    get
    {
        ToolStripMenuItem ownerMenuItem =
            OwnerItem as ToolStripMenuItem;

        // Use the base value in design mode to prevent the designer
        // from setting the base value to the calculated value.
        if (!DesignMode &&
            ownerMenuItem != null && ownerMenuItem.CheckOnClick)
        {
            return base.Enabled && ownerMenuItem.Checked;
        }
        else return base.Enabled;
    }
    set
    {
        base.Enabled = value;
    }
}

```

```

' Enable the item only if its parent item is in the checked state
' and its Enabled property has not been explicitly set to false.
Public Overrides Property Enabled() As Boolean
    Get
        Dim ownerMenuItem As ToolStripMenuItem = _
            TryCast(OwnerItem, ToolStripMenuItem)

        ' Use the base value in design mode to prevent the designer
        ' from setting the base value to the calculated value.
        If Not DesignMode AndAlso ownerMenuItem IsNot Nothing AndAlso _
            ownerMenuItem.CheckOnClick Then
            Return MyBase.Enabled AndAlso ownerMenuItem.Checked
        Else
            Return MyBase.Enabled
        End If
    End Get

    Set(ByVal value As Boolean)
        MyBase.Enabled = value
    End Set
End Property

```

2. Override the [OnOwnerChanged](#) method to subscribe to the [CheckedChanged](#) event of the parent item.

```

// When OwnerItem becomes available, if it is a ToolStripMenuItem
// with a CheckOnClick property value of true, subscribe to its
// CheckedChanged event.
protected override void OnOwnerChanged(EventArgs e)
{
    ToolStripMenuItem ownerMenuItem =
        OwnerItem as ToolStripMenuItem;
    if (ownerMenuItem != null && ownerMenuItem.CheckOnClick)
    {
        ownerMenuItem.CheckedChanged +=
            new EventHandler(OwnerMenuItem_CheckedChanged);
    }
    base.OnOwnerChanged(e);
}

```

```

' When OwnerItem becomes available, if it is a ToolStripMenuItem
' with a CheckOnClick property value of true, subscribe to its
' CheckedChanged event.
Protected Overrides Sub OnOwnerChanged(ByVal e As EventArgs)

    Dim ownerMenuItem As ToolStripMenuItem = _
        TryCast(OwnerItem, ToolStripMenuItem)

    If ownerMenuItem IsNot Nothing AndAlso _
        ownerMenuItem.CheckOnClick Then
        AddHandler ownerMenuItem.CheckedChanged, New _
            EventHandler(AddressOf OwnerMenuItem_CheckedChanged)
    End If

    MyBase.OnOwnerChanged(e)
End Sub

```

3. In the handler for the parent-item [CheckedChanged](#) event, invalidate the item to update the display with the new enabled state.

```

// When the checked state of the parent item changes,
// repaint the item so that the new Enabled state is displayed.
private void OwnerMenuItem_CheckedChanged(
    object sender, EventArgs e)
{
    Invalidate();
}

```

```

' When the checked state of the parent item changes,
' repaint the item so that the new Enabled state is displayed.
Private Sub OwnerMenuItem_CheckedChanged( _
    ByVal sender As Object, ByVal e As EventArgs)
    Invalidate()
End Sub

```

Example

The following code example provides the complete `ToolStripRadioButtonMenuItem` class, and a `Form` class and `Program` class to demonstrate the option-button behavior.

```

using System;
using System.Drawing;
using System.Windows.Forms;
using System.Windows.Forms.VisualStyles;

public class ToolStripRadioButtonMenuItem : ToolStripMenuItem
{
    public ToolStripRadioButtonMenuItem()
        : base()
    {
        Initialize();
    }

    public ToolStripRadioButtonMenuItem(string text)
        : base(text, null, (EventHandler)null)
    {
        Initialize();
    }

    public ToolStripRadioButtonMenuItem(Image image)
        : base(null, image, (EventHandler)null)
    {
        Initialize();
    }

    public ToolStripRadioButtonMenuItem(string text, Image image)
        : base(text, image, (EventHandler)null)
    {
        Initialize();
    }

    public ToolStripRadioButtonMenuItem(string text, Image image,
        EventHandler onClick)
        : base(text, image, onClick)
    {
        Initialize();
    }

    public ToolStripRadioButtonMenuItem(string text, Image image,
        EventHandler onClick, string name)
        : base(text, image, onClick, name)
    {
        Initialize();
    }
}

```

```

}

public ToolStripRadioButtonMenuItem(string text, Image image,
    params ToolStripItem[] dropDownItems)
    : base(text, image, dropDownItems)
{
    Initialize();
}

public ToolStripRadioButtonMenuItem(string text, Image image,
    EventHandler onClick, Keys shortcutKeys)
    : base(text, image, onClick)
{
    Initialize();
    this.ShortcutKeys = shortcutKeys;
}

// Called by all constructors to initialize CheckOnClick.
private void Initialize()
{
    CheckOnClick = true;
}

protected override void OnCheckedChanged(EventArgs e)
{
    base.OnCheckedChanged(e);

    // If this item is no longer in the checked state or if its
    // parent has not yet been initialized, do nothing.
    if (!Checked || this.Parent == null) return;

    // Clear the checked state for all siblings.
    foreach (ToolStripItem item in Parent.Items)
    {
        ToolStripRadioButtonMenuItem radioItem =
            item as ToolStripRadioButtonMenuItem;
        if (radioItem != null && radioItem != this && radioItem.Checked)
        {
            radioItem.Checked = false;

            // Only one item can be selected at a time,
            // so there is no need to continue.
            return;
        }
    }
}

protected override void OnClick(EventArgs e)
{
    // If the item is already in the checked state, do not call
    // the base method, which would toggle the value.
    if (Checked) return;

    base.OnClick(e);
}

// Let the item paint itself, and then paint the RadioButton
// where the check mark is normally displayed.
protected override void OnPaint(PaintEventArgs e)
{
    if (Image != null)
    {
        // If the client sets the Image property, the selection behavior
        // remains unchanged, but the RadioButton is not displayed and the
        // selection is indicated only by the selection rectangle.
        base.OnPaint(e);
        return;
    }
    else

```

```
    }

    // If the Image property is not set, call the base OnPaint method
    // with the CheckState property temporarily cleared to prevent
    // the check mark from being painted.
    CheckState currentState = this.CheckState;
    this.CheckState = CheckState.Unchecked;
    base.OnPaint(e);
    this.CheckState = currentState;
}

// Determine the correct state of the RadioButton.
RadioButtonState buttonState = RadioButtonState.UncheckedNormal;
if (Enabled)
{
    if (mouseDownState)
    {
        if (Checked) buttonState = RadioButtonState.CheckedPressed;
        else buttonState = RadioButtonState.UncheckedPressed;
    }
    else if (mouseHoverState)
    {
        if (Checked) buttonState = RadioButtonState.CheckedHot;
        else buttonState = RadioButtonState.UncheckedHot;
    }
    else
    {
        if (Checked) buttonState = RadioButtonState.CheckedNormal;
    }
}
else
{
    if (Checked) buttonState = RadioButtonState.CheckedDisabled;
    else buttonState = RadioButtonState.UncheckedDisabled;
}

// Calculate the position at which to display the RadioButton.
Int32 offset = (ContentRectangle.Height -
    RadioButtonRenderer.GetGlyphSize(
    e.Graphics, buttonState).Height) / 2;
Point imageLocation = new Point(
    ContentRectangle.Location.X + 4,
    ContentRectangle.Location.Y + offset);

// Paint the RadioButton.
RadioButtonRenderer.DrawRadioButton(
    e.Graphics, imageLocation, buttonState);
}

private bool mouseHoverState = false;

protected override void OnMouseEnter(EventArgs e)
{
    mouseHoverState = true;

    // Force the item to repaint with the new RadioButton state.
    Invalidate();

    base.OnMouseEnter(e);
}

protected override void OnMouseLeave(EventArgs e)
{
    mouseHoverState = false;
    base.OnMouseLeave(e);
}

private bool mouseDownState = false;

protected override void OnMouseDown(MouseEventArgs e)
```

```

protected override void OnMouseDown(MouseEventArgs e)
{
    mouseDownState = true;

    // Force the item to repaint with the new RadioButton state.
    Invalidate();

    base.OnMouseDown(e);
}

protected override void OnMouseUp(MouseEventArgs e)
{
    mouseDownState = false;
    base.OnMouseUp(e);
}

// Enable the item only if its parent item is in the checked state
// and its Enabled property has not been explicitly set to false.
public override bool Enabled
{
    get
    {
        ToolStripMenuItem ownerMenuItem =
            OwnerItem as ToolStripMenuItem;

        // Use the base value in design mode to prevent the designer
        // from setting the base value to the calculated value.
        if (!DesignMode &&
            ownerMenuItem != null && ownerMenuItem.CheckOnClick)
        {
            return base.Enabled && ownerMenuItem.Checked;
        }
        else return base.Enabled;
    }
    set
    {
        base.Enabled = value;
    }
}

// When OwnerItem becomes available, if it is a ToolStripMenuItem
// with a CheckOnClick property value of true, subscribe to its
// CheckedChanged event.
protected override void OnOwnerChanged(EventArgs e)
{
    ToolStripMenuItem ownerMenuItem =
        OwnerItem as ToolStripMenuItem;
    if (ownerMenuItem != null && ownerMenuItem.CheckOnClick)
    {
        ownerMenuItem.CheckedChanged +=
            new EventHandler(OwnerMenuItem_CheckedChanged);
    }
    base.OnOwnerChanged(e);
}

// When the checked state of the parent item changes,
// repaint the item so that the new Enabled state is displayed.
private void OwnerMenuItem_CheckedChanged(
    object sender, EventArgs e)
{
    Invalidate();
}

public class Form1 : Form
{
    private MenuStrip menuStrip1 = new MenuStrip();
    private ToolStripMenuItem mainToolStripMenuItem = new ToolStripMenuItem();
}

```

```

private ToolStripMenuItem toolStripMenuItem1 = new ToolStripMenuItem();
private ToolStripRadioButtonMenuItem toolStripRadioButtonMenuItem1 =
    new ToolStripRadioButtonMenuItem();
private ToolStripRadioButtonMenuItem toolStripRadioButtonMenuItem2 =
    new ToolStripRadioButtonMenuItem();
private ToolStripRadioButtonMenuItem toolStripRadioButtonMenuItem3 =
    new ToolStripRadioButtonMenuItem();
private ToolStripRadioButtonMenuItem toolStripRadioButtonMenuItem4 =
    new ToolStripRadioButtonMenuItem();
private ToolStripRadioButtonMenuItem toolStripRadioButtonMenuItem5 =
    new ToolStripRadioButtonMenuItem();
private ToolStripRadioButtonMenuItem toolStripRadioButtonMenuItem6 =
    new ToolStripRadioButtonMenuItem();

public Form1()
{
    mainToolStripMenuItem.Text = "main";
    toolStripRadioButtonMenuItem1.Text = "option 1";
    toolStripRadioButtonMenuItem2.Text = "option 2";
    toolStripRadioButtonMenuItem3.Text = "option 2-1";
    toolStripRadioButtonMenuItem4.Text = "option 2-2";
    toolStripRadioButtonMenuItem5.Text = "option 3-1";
    toolStripRadioButtonMenuItem6.Text = "option 3-2";
    toolStripMenuItem1.Text = "toggle";
    toolStripMenuItem1.CheckOnClick = true;

    mainToolStripMenuItem.DropDownItems.AddRange(new ToolStripItem[] {
        toolStripRadioButtonMenuItem1, toolStripRadioButtonMenuItem2,
        toolStripMenuItem1});
    toolStripRadioButtonMenuItem2.DropDownItems.AddRange(
        new ToolStripItem[] {toolStripRadioButtonMenuItem3,
        toolStripRadioButtonMenuItem4});
    toolStripMenuItem1.DropDownItems.AddRange(new ToolStripItem[] {
        toolStripRadioButtonMenuItem5, toolStripRadioButtonMenuItem6});

    menuStrip1.Items.AddRange(new ToolStripItem[] {mainToolStripMenuItem});
    Controls.Add(menuStrip1);
    MainMenuStrip = menuStrip1;
    Text = "ToolStripRadioButtonMenuItem demo";
}

static class Program
{
    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new Form1());
    }
}

```

```

Imports System
Imports System.Drawing
Imports System.Windows.Forms
Imports System.Windows.Forms.VisualStyles

Public Class ToolStripRadioButtonMenuItem
    Inherits ToolStripMenuItem

    Public Sub New()
        MyBase.New()
        Initialize()
    End Sub

    Public Sub New(ByVal text As String)

```

```

 MyBase.New(text, Nothing, CType(Nothing, EventHandler))
 Initialize()
 End Sub

 Public Sub New(ByVal image As Image)
 MyBase.New(Nothing, image, CType(Nothing, EventHandler))
 Initialize()
 End Sub

 Public Sub New(ByVal text As String, ByVal image As Image)
 MyBase.New(text, image, CType(Nothing, EventHandler))
 Initialize()
 End Sub

 Public Sub New(ByVal text As String, _
 ByVal image As Image, ByVal onClick As EventHandler)
 MyBase.New(text, image, onClick)
 Initialize()
 End Sub

 Public Sub New(ByVal text As String, ByVal image As Image, _
 ByVal onClick As EventHandler, ByVal name As String)
 MyBase.New(text, image, onClick, name)
 Initialize()
 End Sub

 Public Sub New(ByVal text As String, ByVal image As Image, _
 ByVal ParamArray dropDownItems() As ToolStripItem)
 MyBase.New(text, image, dropDownItems)
 Initialize()
 End Sub

 Public Sub New(ByVal text As String, ByVal image As Image, _
 ByVal onClick As EventHandler, ByVal shortcutKeys As Keys)
 MyBase.New(text, image, onClick)
 Initialize()
 Me.ShortcutKeys = shortcutKeys
 End Sub

 ' Called by all constructors to initialize CheckOnClick.
 Private Sub Initialize()
 CheckOnClick = True
 End Sub

 Protected Overrides Sub OnCheckedChanged(ByVal e As EventArgs)

 MyBase.OnCheckedChanged(e)

 ' If this item is no longer in the checked state or if its
 ' parent has not yet been initialized, do nothing.
 If Not Checked OrElse Me.Parent Is Nothing Then Return

 ' Clear the checked state for all siblings.
 For Each item As ToolStripItem In Parent.Items

 Dim radioItem As ToolStripRadioButtonMenuItem = _
 TryCast(item, ToolStripRadioButtonMenuItem)
 If radioItem IsNot Nothing AndAlso _
 radioItem IsNot Me AndAlso _
 radioItem.Checked Then

 radioItem.Checked = False

 ' Only one item can be selected at a time,
 ' so there is no need to continue.
 Return

 End If
 Next

```

```

End Sub

Protected Overrides Sub OnClick(ByVal e As EventArgs)

    ' If the item is already in the checked state, do not call
    ' the base method, which would toggle the value.
    If Checked Then Return

    MyBase.OnClick(e)
End Sub

' Let the item paint itself, and then paint the RadioButton
' where the check mark is normally displayed.
Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)

    If Image IsNot Nothing Then
        ' If the client sets the Image property, the selection behavior
        ' remains unchanged, but the RadioButton is not displayed and the
        ' selection is indicated only by the selection rectangle.
        MyBase.OnPaint(e)
        Return
    Else
        ' If the Image property is not set, call the base OnPaint method
        ' with the CheckState property temporarily cleared to prevent
        ' the check mark from being painted.
        Dim currentState As CheckState = Me.CheckState
        Me.CheckState = CheckState.Unchecked
        MyBase.OnPaint(e)
        Me.CheckState = currentState
    End If

    ' Determine the correct state of the RadioButton.
    Dim buttonState As RadioButtonState = RadioButtonState.UncheckedNormal
    If Enabled Then
        If mouseDownState Then
            If Checked Then
                buttonState = RadioButtonState.CheckedPressed
            Else
                buttonState = RadioButtonState.UncheckedPressed
            End If
        ElseIf mouseHoverState Then
            If Checked Then
                buttonState = RadioButtonState.CheckedHot
            Else
                buttonState = RadioButtonState.UncheckedHot
            End If
        Else
            If Checked Then buttonState = RadioButtonState.CheckedNormal
        End If
    Else
        If Checked Then
            buttonState = RadioButtonState.CheckedDisabled
        Else
            buttonState = RadioButtonState.UncheckedDisabled
        End If
    End If

    ' Calculate the position at which to display the RadioButton.
    Dim offset As Int32 = CInt((ContentRectangle.Height - _
        RadioButtonRenderer.GetGlyphSize( _
        e.Graphics, buttonState).Height) / 2)
    Dim imageLocation As Point = New Point( _
        ContentRectangle.Location.X + 4, _
        ContentRectangle.Location.Y + offset)

    ' Paint the RadioButton.
    RadioButtonRenderer.DrawRadioButton( _
        e.Graphics, imageLocation, buttonState)

```

```

End Sub

Private mouseHoverState As Boolean = False

Protected Overrides Sub OnMouseEnter(ByVal e As EventArgs)
    mouseHoverState = True

    ' Force the item to repaint with the new RadioButton state.
    Invalidate()

    MyBase.OnMouseEnter(e)
End Sub

Protected Overrides Sub OnMouseLeave(ByVal e As EventArgs)
    mouseHoverState = False
    MyBase.OnMouseLeave(e)
End Sub

Private mouseDownState As Boolean = False

Protected Overrides Sub OnMouseDown(ByVal e As MouseEventArgs)
    mouseDownState = True

    ' Force the item to repaint with the new RadioButton state.
    Invalidate()

    MyBase.OnMouseDown(e)
End Sub

Protected Overrides Sub OnMouseUp(ByVal e As MouseEventArgs)
    mouseDownState = False
    MyBase.OnMouseUp(e)
End Sub

' Enable the item only if its parent item is in the checked state
' and its Enabled property has not been explicitly set to false.
Public Overrides Property Enabled() As Boolean
    Get
        Dim ownerMenuItem As ToolStripMenuItem = _
            TryCast(OwnerItem, ToolStripMenuItem)

        ' Use the base value in design mode to prevent the designer
        ' from setting the base value to the calculated value.
        If Not DesignMode AndAlso ownerMenuItem IsNot Nothing AndAlso _
            ownerMenuItem.CheckOnClick Then
            Return MyBase.Enabled AndAlso ownerMenuItem.Checked
        Else
            Return MyBase.Enabled
        End If
    End Get
    Set(ByVal value As Boolean)
        MyBase.Enabled = value
    End Set
End Property

' When OwnerItem becomes available, if it is a ToolStripMenuItem
' with a CheckOnClick property value of true, subscribe to its
' CheckedChanged event.
Protected Overrides Sub OnOwnerChanged(ByVal e As EventArgs)

    Dim ownerMenuItem As ToolStripMenuItem = _
        TryCast(OwnerItem, ToolStripMenuItem)

    If ownerMenuItem IsNot Nothing AndAlso _
        ownerMenuItem.CheckOnClick Then
        AddHandler ownerMenuItem.CheckedChanged, New _
            EventHandler(AddressOf OwnerMenuItem_CheckedChanged)
    End If
End Sub

```

```

End If

 MyBase.OnOwnerChanged(e)

End Sub

' When the checked state of the parent item changes,
' repaint the item so that the new Enabled state is displayed.
Private Sub OwnerMenuItem_CheckedChanged( _
    ByVal sender As Object, ByVal e As EventArgs)
    Invalidate()
End Sub

End Class

Public Class Form1
    Inherits Form

    Private menuStrip1 As New MenuStrip()
    Private mainToolStripMenuItem As New ToolStripMenuItem()
    Private toolStripMenuItem1 As New ToolStripMenuItem()
    Private toolStripRadioButtonMenuItem1 As New ToolStripRadioButtonMenuItem()
    Private toolStripRadioButtonMenuItem2 As New ToolStripRadioButtonMenuItem()
    Private toolStripRadioButtonMenuItem3 As New ToolStripRadioButtonMenuItem()
    Private toolStripRadioButtonMenuItem4 As New ToolStripRadioButtonMenuItem()
    Private toolStripRadioButtonMenuItem5 As New ToolStripRadioButtonMenuItem()
    Private toolStripRadioButtonMenuItem6 As New ToolStripRadioButtonMenuItem()

    Public Sub New()

        Me.mainToolStripMenuItem.Text = "main"
        toolStripRadioButtonMenuItem1.Text = "option 1"
        toolStripRadioButtonMenuItem2.Text = "option 2"
        toolStripRadioButtonMenuItem3.Text = "option 2-1"
        toolStripRadioButtonMenuItem4.Text = "option 2-2"
        toolStripRadioButtonMenuItem5.Text = "option 3-1"
        toolStripRadioButtonMenuItem6.Text = "option 3-2"
        toolStripMenuItem1.Text = "toggle"
        toolStripMenuItem1.CheckOnClick = True

        mainToolStripMenuItem.DropDownItems.AddRange(New ToolStripItem() { _
            toolStripRadioButtonMenuItem1, toolStripRadioButtonMenuItem2, _
            toolStripMenuItem1})
        toolStripRadioButtonMenuItem2.DropDownItems.AddRange( _
            New ToolStripItem() {toolStripRadioButtonMenuItem3, _
            toolStripRadioButtonMenuItem4})
        toolStripMenuItem1.DropDownItems.AddRange(New ToolStripItem() { _
            toolStripRadioButtonMenuItem5, toolStripRadioButtonMenuItem6})

        menuStrip1.Items.AddRange(New ToolStripItem() {mainToolStripMenuItem})
        Controls.Add(menuStrip1)
        MainMenuStrip = menuStrip1
        Text = "ToolStripRadioButtonMenuItem demo"
    End Sub
End Class

Public Class Program

    <STAThread()> Public Shared Sub Main()
        Application.EnableVisualStyles()
        Application.SetCompatibleTextRenderingDefault(False)
        Application.Run(New Form1())
    End Sub

End Class

```

Compiling the Code

This example requires:

- References to the System, System.Drawing, and System.Windows.Forms assemblies.

See Also

[MenuStrip](#)

[ToolStripMenuItem](#)

[ToolStripMenuItem.CheckOnClick](#)

[ToolStripMenuItem.Checked](#)

[ToolStripMenuItem.OnCheckedChanged](#)

[ToolStripMenuItem.OnPaint](#)

[ToolStripMenuItem.Enabled](#)

[RadioButtonRenderer](#)

[MenuStrip Control](#)

[How to: Implement a Custom ToolStripRenderer](#)

How to: Hide ToolStripMenuItems

5/4/2018 • 1 min to read • [Edit Online](#)

Hiding menu items is a way to control the user interface of your application and restrict user commands. Often, you will want to hide an entire menu when all of the menu items on it are unavailable. This presents fewer distractions for the user. Furthermore, you might want to both hide and disable the menu or menu item, as hiding alone does not prevent the user from accessing a menu command by using a shortcut key.

To hide any menu item programmatically

- Within the method where you set the properties of the menu item, add code to set the [Visible](#) property to `false`.

```
MenuItem3.Visible = False
```

```
menuItem3.Visible = false;
```

```
menuItem3->Visible = false;
```

See Also

[Visible](#)

[MenuStrip](#)

[MenuStrip Control Overview](#)

[How to: Disable ToolStripMenuItems](#)

How to: Hide ToolStripMenuItems Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

Hiding menu items is a way to control the user interface (UI) of your application and restrict user commands. Often, you will want to hide an entire menu when all of the menu items on it are unavailable. This presents fewer distractions for the user. Furthermore, you might want to both hide and disable the menu or menu item, as hiding alone does not prevent the user from accessing a menu command by using a shortcut key. For more information on disabling menu items, see [How to: Disable ToolStripMenuItems Using the Designer](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To hide a top-level menu and its submenu items

1. Select the top-level menu item and set its **Visible** or **Available** property to `false`.

When you hide the top-level menu item, all menu items within that menu are also hidden. If you click somewhere other than on the **MenuStrip** after setting **Visible** to `false`, the entire top-level menu item and its submenu items disappear from your form, thus showing you the run-time effect of your action. To display the hidden top-level menu item at design time, click on the **MenuStrip** in the **Component Tray**, in **Document Outline**, or at the top of the property grid.

NOTE

You will rarely hide an entire menu except for multiple document interface (MDI) child menus in a merging scenario.

To hide a submenu item

1. Select the submenu item and set its **Visible** property to `false`.

When you hide a submenu item, it remains visible on your form at design time so that you can easily select it for further work. It will actually be hidden at run time.

See Also

[Visible](#)

[MenuStrip](#)

[Enabled](#)

[Available](#)

[Overflow](#)

[MenuStrip Control Overview](#)

[How to: Disable ToolStripMenuItems Using the Designer](#)

How to: Insert a MenuStrip into an MDI Drop-Down Menu (Windows Forms)

5/4/2018 • 2 min to read • [Edit Online](#)

In some applications, the kind of a multiple-document interface (MDI) child window can be different from the MDI parent window. For example, the MDI parent might be a spreadsheet, and the MDI child might be a chart. In that case, you want to update the contents of the MDI parent's menu with the contents of the MDI child's menu as MDI child windows of different kinds are activated.

The following procedure uses the `IsMdiContainer`, `AllowMerge`, `MergeAction`, and `MergeIndex` properties to insert a group of menu items from the MDI child menu into the drop-down part of the MDI parent menu. Closing the MDI child window removes the inserted menu items from the MDI parent.

To insert a MenuStrip into an MDI drop-down menu

1. Create a form and set its `IsMdiContainer` property to `true`.
2. Add a `MenuStrip` to `Form1` and set the `AllowMerge` property of the `MenuStrip` to `true`.
3. Add a top-level menu item to the `Form1` `MenuStrip` and set its `Text` property to `&File`.
4. Add three submenu items to the `&File` menu item and set their `Text` properties to `&Open`, `&Import from`, and `E&xit`.
5. Add two submenu items to the `&Import from` submenu item and set their `Text` properties to `&Word` and `&Excel`.
6. Add a form to the project, add a `MenuStrip` to the form, and set the `AllowMerge` property of the `Form2` `MenuStrip` to `true`.
7. Add a top-level menu item to the `Form2` `MenuStrip` and set its `Text` property to `&File`.
8. Add submenu items to the `&File` menu of `Form2` in the following order: a `ToolStripSeparator`, `&Save`, `&Close``` and `Save`, and another `ToolStripSeparator`.
9. Set the `MergeAction` and `MergeIndex` properties of the `Form2` menu items as shown in the following table.

FORM2 MENU ITEM	MERGEACTION VALUE	MERGEINDEX VALUE
File	MatchOnly	-1
Separator	Insert	2
Save	Insert	3
Save and Close	Insert	4
Separator	Insert	5

10. Create an event handler for the `Click` event of the `&Open` `ToolStripMenuItem`.
11. Within the event handler, insert code similar to the following code example to create and display new instances of `Form2` as MDI children of `Form1`.

```
Private Sub openToolStripMenuItem_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles openToolStripMenuItem.Click  
    Dim NewMDIChild As New Form2()  
    'Set the parent form of the child window.  
    NewMDIChild.MdiParent = Me  
    'Display the new form.  
    NewMDIChild.Show()  
End Sub
```

```
private void openToolStripMenuItem_Click(object sender, EventArgs e)  
{  
    Form2 newMDIChild = new Form2();  
    // Set the parent form of the child window.  
    newMDIChild.MdiParent = this;  
    // Display the new form.  
    newMDIChild.Show();  
}
```

12. Place code similar to the following code example in the `openToolStripMenuItem` to register the event handler.

```
Private Sub openToolStripMenuItem_Click(sender As Object, e As _  
EventArgs) Handles openToolStripMenuItem.Click
```

```
this.openToolStripMenuItem.Click += new System.EventHandler(this.openToolStripMenuItem_Click);
```

Compiling the Code

This example requires:

- Two `Form` controls named `Form1` and `Form2`.
- A `MenuStrip` control on `Form1` named `menuStrip1`, and a `MenuStrip` control on `Form2` named `menuStrip2`.
- References to the `System` and `System.Windows.Forms` assemblies.

See Also

[How to: Create MDI Parent Forms](#)

[How to: Create MDI Child Forms](#)

[MenuStrip Control Overview](#)

How to: Remove a ToolStripMenuItem from an MDI Drop-Down Menu (Windows Forms)

5/4/2018 • 2 min to read • [Edit Online](#)

In some applications, the kind of a multiple-document interface (MDI) child window can be different from the MDI parent window. For example, the MDI parent might be a spreadsheet, and the MDI child might be a chart. In that case, you want to update the contents of the MDI parent's menu with the contents of the MDI child's menu as MDI child windows of different kinds are activated.

The following procedure uses the `IsMdiContainer`, `AllowMerge`, `MergeAction`, and `MergeIndex` properties to remove a menu item from the drop-down part of the MDI parent menu. Closing the MDI child window restores the removed menu items to the MDI parent menu.

To remove a ToolStrip from an MDI drop-down menu

1. Create a form and set its `IsMdiContainer` property to `true`.
2. Add a `MenuStrip` to `Form1` and set the `AllowMerge` property of the `MenuStrip` to `true`.
3. Add a top-level menu item to the `Form1` `MenuStrip` and set its `Text` property to `&File`.
4. Add three submenu items to the `&File` menu item and set their `Text` properties to `&Open`, `&Import from`, and `E&xit`.
5. Add two submenu items to the `&Import from` submenu item and set their `Text` properties to `&Word` and `&Excel`.
6. Add a form to the project, add a `MenuStrip` to the form, and set the `AllowMerge` property of the `Form2` `MenuStrip` to `true`.
7. Add a top-level menu item to the `Form2` `MenuStrip` and set its `Text` property to `&File`.
8. Add an `&Import from` submenu item to the `&File` menu of `Form2`, and add an `&Word` submenu item to the `&File` menu.
9. Set the `MergeAction` and `MergeIndex` properties of the `Form2` menu items as shown in the following table.

FORM2 MENU ITEM	MERGEACTION VALUE	MERGEINDEX VALUE
File	MatchOnly	-1
Import from	MatchOnly	-1
Word	Remove	-1

10. In `Form1`, create an event handler for the `Click` event of the `&Open` `ToolStripMenuItem`.
11. Within the event handler, insert code similar to the following code example to create and display new instances of `Form2` as MDI children of `Form1`:

```
Private Sub openToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles openToolStripMenuItem.Click
    Dim NewMDIChild As New Form2()
    'Set the parent form of the child window.
    NewMDIChild.MdiParent = Me
    'Display the new form.
    NewMDIChild.Show()
End Sub
```

```
private void openToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form2 newMDIChild = new Form2();
    // Set the parent form of the child window.
    newMDIChild.MdiParent = this;
    // Display the new form.
    newMDIChild.Show();
}
```

12. Place code similar to the following code example in the `openToolStripMenuItem` to register the event handler.

```
Private Sub openToolStripMenuItem_Click(sender As Object, e As _
EventArgs) Handles openToolStripMenuItem.Click
```

```
this.openToolStripMenuItem.Click += new _  
System.EventHandler(this.openToolStripMenuItem_Click);
```

Compiling the Code

This example requires:

- Two `Form` controls named `Form1` and `Form2`.
- A `MenuStrip` control on `Form1` named `menuStrip1`, and a `MenuStrip` control on `Form2` named `menuStrip2`.
- References to the `System` and `System.Windows.Forms` assemblies.

See Also

[How to: Create MDI Parent Forms](#)

[How to: Create MDI Child Forms](#)

[MenuStrip Control Overview](#)

How to: Move ToolStripMenuItems

5/4/2018 • 2 min to read • [Edit Online](#)

At design time, you can move entire top-level menus and their menu items to a different place on the [MenuStrip](#). You can also move individual menu items between top-level menus or change the position of menu items within a menu.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To move a top-level menu and its menu items to another top-level location

1. Click and hold down the left mouse button on the menu that you want to move.
2. Drag the insertion point to the top-level menu that is before the intended new location and release the left mouse button.

The selected menu moves to the right of the insertion point.

To move a top-level menu and its menu items to a drop-down location

1. Left-click the menu that you want to move and press **CTRL+X**, or right-click the menu and select **Cut** from the shortcut menu.
2. In the destination top-level menu, left-click the menu item above the intended new location and press **CTRL+V**, or right-click the menu item above the intended new location and select **Paste** from the shortcut menu.

The menu that you cut is inserted after the selected menu item.

To move a menu item within a menu using the Items Collection Editor

1. Right-click the menu that contains the menu item you want to move.
2. From the shortcut menu, choose **Edit DropDownItems**.
3. In the **Items Collection Editor**, left-click the menu item you want to move.
4. Click the UP and DOWN ARROW keys to move the menu item within the menu.
5. Click **OK**.

To move a menu item within a menu using the keyboard

1. Press and hold down the **ALT** key.
2. Click and hold the left mouse button on the menu item that you want to move.
3. Drag the menu item to the new location and release the left mouse button.

To move a menu item to another menu

1. Left-click the menu item that you want to move and press **CTRL+X**, or right-click the menu item and choose **Cut** from the shortcut menu.
2. Left-click the menu that will contain the menu item that you cut.

3. Left-click the menu item that is before the intended new location and press CTRL+V, or right-click the menu item that is before the intended new location and select **Paste** from the shortcut menu.

The menu item that you cut is inserted after the selected menu item.

See Also

[MenuStrip](#)

[ToolStripMenuItem](#)

[MenuStrip Control Overview](#)

How to: Configure MenuStrip Check Margins and Image Margins

5/4/2018 • 5 min to read • [Edit Online](#)

You can customize a [MenuStrip](#) by setting the [ShowImageMargin](#) and [ShowCheckMargin](#) properties in various combinations.

Example

The following code example demonstrates how to set and customize the [ContextMenuStrip](#) check margins and image margins. The procedure is the same for a [ContextMenuStrip](#) or a [MenuStrip](#).

```
// This code example demonstrates how to set the check
// and image margins for a ToolStripMenuItem.
class Form5 : Form
{
    public Form5()
    {
        // Size the form to show three wide menu items.
        this.Width = 500;
        this.Text = "ToolStripContextMenuStrip: Image and Check Margins";

        // Create a new MenuStrip control.
        MenuStrip ms = new MenuStrip();

        // Create the ToolStripMenuItems for the MenuStrip control.
        ToolStripMenuItem bothMargins = new ToolStripMenuItem("BothMargins");
        ToolStripMenuItem imageMarginOnly = new ToolStripMenuItem("ImageMargin");
        ToolStripMenuItem checkMarginOnly = new ToolStripMenuItem("CheckMargin");
        ToolStripMenuItem noMargins = new ToolStripMenuItem("NoMargins");

        // Customize the DropDowns menus.
        // This ToolStripMenuItem has an image margin
        // and a check margin.
        bothMargins.DropDown = CreateCheckImageContextMenuStrip();
        ((ContextMenuStrip)bothMargins.DropDown).ShowImageMargin = true;
        ((ContextMenuStrip)bothMargins.DropDown).ShowCheckMargin = true;

        // This ToolStripMenuItem has only an image margin.
        imageMarginOnly.DropDown = CreateCheckImageContextMenuStrip();
        ((ContextMenuStrip)imageMarginOnly.DropDown).ShowImageMargin = true;
        ((ContextMenuStrip)imageMarginOnly.DropDown).ShowCheckMargin = false;

        // This ToolStripMenuItem has only a check margin.
        checkMarginOnly.DropDown = CreateCheckImageContextMenuStrip();
        ((ContextMenuStrip)checkMarginOnly.DropDown).ShowImageMargin = false;
        ((ContextMenuStrip)checkMarginOnly.DropDown).ShowCheckMargin = true;

        // This ToolStripMenuItem has no image and no check margin.
        noMargins.DropDown = CreateCheckImageContextMenuStrip();
        ((ContextMenuStrip)noMargins.DropDown).ShowImageMargin = false;
        ((ContextMenuStrip)noMargins.DropDown).ShowCheckMargin = false;

        // Populate the MenuStrip control with the ToolStripMenuItems.
        ms.Items.Add(bothMargins);
        ms.Items.Add(imageMarginOnly);
        ms.Items.Add(checkMarginOnly);
        ms.Items.Add(noMargins);
```

```
// Dock the ToolStrip control to the top of the form.  
ms.Dock = DockStyle.Top;  
  
// Add the ToolStrip control to the controls collection last.  
// This is important for correct placement in the z-order.  
this.Controls.Add(ms);  
}  
  
// This utility method creates a Bitmap for use in  
// a ToolStripMenuItem's image margin.  
internal Bitmap CreateSampleBitmap()  
{  
    // The Bitmap is a smiley face.  
    Bitmap sampleBitmap = new Bitmap(32, 32);  
    Graphics g = Graphics.FromImage(sampleBitmap);  
  
    using (Pen p = new Pen(ProfessionalColors.ButtonPressedBorder))  
    {  
        // Set the Pen width.  
        p.Width = 4;  
  
        // Set up the mouth geometry.  
        Point[] curvePoints = new Point[]{  
            new Point(4,14),  
            new Point(16,24),  
            new Point(28,14)};  
  
        // Draw the mouth.  
        g.DrawCurve(p, curvePoints);  
  
        // Draw the eyes.  
        g.DrawEllipse(p, new Rectangle(new Point(7, 4), new Size(3, 3)));  
        g.DrawEllipse(p, new Rectangle(new Point(22, 4), new Size(3, 3)));  
    }  
  
    return sampleBitmap;  
}  
  
// This utility method creates a ContextMenuStrip control  
// that has four ToolStripMenuItems showing the four  
// possible combinations of image and check margins.  
internal ContextMenuStrip CreateCheckImageContextMenuStrip()  
{  
    // Create a new ContextMenuStrip control.  
    ContextMenuStrip checkImageContextMenuStrip = new ContextMenuStrip();  
  
    // Create a ToolStripMenuItem with a  
    // check margin and an image margin.  
    ToolStripMenuItem yesCheckYesImage =  
        new ToolStripMenuItem("Check, Image");  
    yesCheckYesImage.Checked = true;  
    yesCheckYesImage.Image = CreateSampleBitmap();  
  
    // Create a ToolStripMenuItem with no  
    // check margin and with an image margin.  
    ToolStripMenuItem noCheckYesImage =  
        new ToolStripMenuItem("No Check, Image");  
    noCheckYesImage.Checked = false;  
    noCheckYesImage.Image = CreateSampleBitmap();  
  
    // Create a ToolStripMenuItem with a  
    // check margin and without an image margin.  
    ToolStripMenuItem yesCheckNoImage =  
        new ToolStripMenuItem("Check, No Image");  
    yesCheckNoImage.Checked = true;  
  
    // Create a ToolStripMenuItem with no  
    // check margin and no image margin.  
    ToolStripMenuItem noCheckNoImage =
```

```

        new ToolStripMenuItem("No Check, No Image");
        noCheckNoImage.Checked = false;

        // Add the ToolStripMenuItems to the ContextMenuStrip control.
        checkImageContextMenuStrip.Items.Add(yesCheckYesImage);
        checkImageContextMenuStrip.Items.Add(noCheckYesImage);
        checkImageContextMenuStrip.Items.Add(yesCheckNoImage);
        checkImageContextMenuStrip.Items.Add(noCheckNoImage);

        return checkImageContextMenuStrip;
    }
}

```

```

' This code example demonstrates how to set the check
' and image margins for a ToolStripMenuItem.

Class Form5
    Inherits Form

    Public Sub New()
        ' Size the form to show three wide menu items.
        Me.Width = 500
        Me.Text = "ToolStripContextMenuStrip: Image and Check Margins"

        ' Create a new MenuStrip control.
        Dim ms As New MenuStrip()

        ' Create the ToolStripMenuItems for the MenuStrip control.
        Dim bothMargins As New ToolStripMenuItem("BothMargins")
        Dim imageMarginOnly As New ToolStripMenuItem("ImageMargin")
        Dim checkMarginOnly As New ToolStripMenuItem("CheckMargin")
        Dim noMargins As New ToolStripMenuItem("NoMargins")

        ' Customize the DropDowns menus.
        ' This ToolStripMenuItem has an image margin
        ' and a check margin.
        bothMargins.DropDown = CreateCheckImageContextMenuStrip()
        CType(bothMargins.DropDown, ContextMenuStrip).ShowImageMargin = True
        CType(bothMargins.DropDown, ContextMenuStrip).ShowCheckMargin = True

        ' This ToolStripMenuItem has only an image margin.
        imageMarginOnly.DropDown = CreateCheckImageContextMenuStrip()
        CType(imageMarginOnly.DropDown, ContextMenuStrip).ShowImageMargin = True
        CType(imageMarginOnly.DropDown, ContextMenuStrip).ShowCheckMargin = False

        ' This ToolStripMenuItem has only a check margin.
        checkMarginOnly.DropDown = CreateCheckImageContextMenuStrip()
        CType(checkMarginOnly.DropDown, ContextMenuStrip).ShowImageMargin = False
        CType(checkMarginOnly.DropDown, ContextMenuStrip).ShowCheckMargin = True

        ' This ToolStripMenuItem has no image and no check margin.
        noMargins.DropDown = CreateCheckImageContextMenuStrip()
        CType(noMargins.DropDown, ContextMenuStrip).ShowImageMargin = False
        CType(noMargins.DropDown, ContextMenuStrip).ShowCheckMargin = False

        ' Populate the MenuStrip control with the ToolStripMenuItems.
        ms.Items.Add(bothMargins)
        ms.Items.Add(imageMarginOnly)
        ms.Items.Add(checkMarginOnly)
        ms.Items.Add(noMargins)

        ' Dock the MenuStrip control to the top of the form.
        ms.Dock = DockStyle.Top

        ' Add the MenuStrip control to the controls collection last.
        ' This is important for correct placement in the z-order.
        Me.Controls.Add(ms)
    End Sub

```

```

' This utility method creates a Bitmap for use in
' a ToolStripMenuItem's image margin.
Friend Function CreateSampleBitmap() As Bitmap

    ' The Bitmap is a smiley face.
    Dim sampleBitmap As New Bitmap(32, 32)
    Dim g As Graphics = Graphics.FromImage(sampleBitmap)

    Dim p As New Pen(ProfessionalColors.ButtonPressedBorder)
    Try
        ' Set the Pen width.
        p.Width = 4

        ' Set up the mouth geometry.
        Dim curvePoints() As Point = _
        {New Point(4, 14), New Point(16, 24), New Point(28, 14)}

        ' Draw the mouth.
        g.DrawCurve(p, curvePoints)

        ' Draw the eyes.
        g.DrawEllipse(p, New Rectangle(New Point(7, 4), New Size(3, 3)))
        g.DrawEllipse(p, New Rectangle(New Point(22, 4), New Size(3, 3)))
    Finally
        p.Dispose()
    End Try

    Return sampleBitmap
End Function

' This utility method creates a ContextMenuStrip control
' that has four ToolStripMenuItems showing the four
' possible combinations of image and check margins.
Friend Function CreateCheckImageContextMenuStrip() As ContextMenuStrip
    ' Create a new ContextMenuStrip control.
    Dim checkImageContextMenuStrip As New ContextMenuStrip()

    ' Create a ToolStripMenuItem with a
    ' check margin and an image margin.
    Dim yesCheckYesImage As New ToolStripMenuItem("Check, Image")
    yesCheckYesImage.Checked = True
    yesCheckYesImage.Image = CreateSampleBitmap()

    ' Create a ToolStripMenuItem with no
    ' check margin and with an image margin.
    Dim noCheckYesImage As New ToolStripMenuItem("No Check, Image")
    noCheckYesImage.Checked = False
    noCheckYesImage.Image = CreateSampleBitmap()

    ' Create a ToolStripMenuItem with a
    ' check margin and without an image margin.
    Dim yesCheckNoImage As New ToolStripMenuItem("Check, No Image")
    yesCheckNoImage.Checked = True

    ' Create a ToolStripMenuItem with no
    ' check margin and no image margin.
    Dim noCheckNoImage As New ToolStripMenuItem("No Check, No Image")
    noCheckNoImage.Checked = False

    ' Add the ToolStripMenuItems to the ContextMenuStrip control.
    checkImageContextMenuStrip.Items.Add(yesCheckYesImage)
    checkImageContextMenuStrip.Items.Add(noCheckYesImage)
    checkImageContextMenuStrip.Items.Add(yesCheckNoImage)
    checkImageContextMenuStrip.Items.Add(noCheckNoImage)

    Return checkImageContextMenuStrip
End Function
End Class

```

Compiling the Code

This example requires:

- References to the System, System.Drawing and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[MenuStrip](#)

[ContextMenuStrip](#)

[ToolStripDropDown](#)

[ToolStrip Control](#)

[How to: Enable Check Margins and Image Margins in ContextMenuStrip Controls](#)

How to: Provide Standard Menu Items to a Form

5/4/2018 • 11 min to read • [Edit Online](#)

You can provide a standard menu for your forms with the [MenuStrip](#) control.

There is extensive support for this feature in Visual Studio.

Also see [Walkthrough: Providing Standard Menu Items to a Form](#).

Example

The following code example demonstrates how to use a [MenuStrip](#) control to create a form with a standard menu. Menu item selections are displayed in a [StatusStrip](#) control.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace StandardMenuFormCS
{
    public class Form1 : Form
    {
        private StatusStrip statusStrip1;
        private ToolStripStatusLabel toolStripStatusLabel1;

        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        public Form1()
        {
            InitializeComponent();
        }

        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        // This utility method assigns the value of a ToolStripItem
        // control's Text property to the Text property of the
        // ToolStripStatusLabel.
        private void UpdateStatus(ToolStripItem item)
        {
            if (item != null)
            {
                string msg = String.Format("{0} selected", item.Text);
                this.statusStrip1.Items[0].Text = msg;
            }
        }

        // This method is the DropDownItemClicked event handler.
        // It passes the ClickedItem object to a utility method
        // called UpdateStatus, which updates the text displayed
    }
}
```

```
// in the StatusStrip control.
private void fileToolStripMenuItem_DropDownItemClicked(
    object sender, ToolStripItemClickedEventArgs e)
{
    this.UpdateStatus(e.ClickedItem);
}

#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(Form1));
    this.menuStrip1 = new System.Windows.Forms.MenuStrip();
    this.fileToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.newToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.openToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.toolStripSeparator = new System.Windows.Forms.ToolStripSeparator();
    this.saveToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.saveAsToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.toolStripSeparator1 = new System.Windows.Forms.ToolStripSeparator();
    this.printToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.printPreviewToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.toolStripSeparator2 = new System.Windows.Forms.ToolStripSeparator();
    this.exitToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.editToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.undoToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.redoToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.toolStripSeparator3 = new System.Windows.Forms.ToolStripSeparator();
    this.cutToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.copyToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.pasteToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.toolStripSeparator4 = new System.Windows.Forms.ToolStripSeparator();
    this.selectAllToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.toolsToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.customizeToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.optionsToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.helpToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.contentsToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.indexToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.searchToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.toolStripSeparator5 = new System.Windows.Forms.ToolStripSeparator();
    this.aboutToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.statusStrip1 = new System.Windows.Forms.StatusStrip();
    this.toolStripStatusLabel1 = new System.Windows.Forms.ToolStripStatusLabel();
    this.menuStrip1.SuspendLayout();
    this.statusStrip1.SuspendLayout();
    this.SuspendLayout();
    //
    // menuStrip1
    //
    this.menuStrip1.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
        this.fileToolStripMenuItem,
        this.editToolStripMenuItem,
        this.toolsToolStripMenuItem,
        this.helpToolStripMenuItem});
    this.menuStrip1.Location = new System.Drawing.Point(0, 0);
    this.menuStrip1.Name = "menuStrip1";
    this.menuStrip1.Size = new System.Drawing.Size(292, 24);
    this.menuStrip1.TabIndex = 0;
    this.menuStrip1.Text = "menuStrip1";
    //
    // fileToolStripMenuItem
    //
    this.fileToolStripMenuItem.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] {
        this.newToolStripMenuItem,
        this.openToolStripMenuItem,
        this.toolStripSeparator1,
        this.saveToolStripMenuItem,
        this.saveAsToolStripMenuItem,
        this.toolStripSeparator2,
        this.printToolStripMenuItem,
        this.printPreviewToolStripMenuItem,
        this.toolStripSeparator3,
        this.exitToolStripMenuItem});
    this.fileToolStripMenuItem.Name = "fileToolStripMenuItem";
    this.fileToolStripMenuItem.Size = new System.Drawing.Size(48, 22);
    this.fileToolStripMenuItem.Text = "&File";
    this.newToolStripMenuItem.Name = "newToolStripMenuItem";
    this.newToolStripMenuItem.ShortcutKeys = Keys.Control | Keys.N;
    this.newToolStripMenuItem.Size = new System.Drawing.Size(144, 22);
    this.newToolStripMenuItem.Text = "&New";
    this.openToolStripMenuItem.Name = "openToolStripMenuItem";
    this.openToolStripMenuItem.ShortcutKeys = Keys.Control | Keys.O;
    this.openToolStripMenuItem.Size = new System.Drawing.Size(144, 22);
    this.openToolStripMenuItem.Text = "&Open";
    this.toolStripSeparator1 отдelete;
```

```
this.menuStrip1.Items.Add(this.newToolStripMenuItem);
this.menuStrip1.Items.Add(this.openToolStripMenuItem);
this.menuStrip1.Items.Add(this.toolStripMenuItemSeparator);
this.menuStrip1.Items.Add(this.saveToolStripMenuItem);
this.menuStrip1.Items.Add(this.saveAsToolStripMenuItem);
this.menuStrip1.Items.Add(this.toolStripMenuItemSeparator1);
this.menuStrip1.Items.Add(this.printToolStripMenuItem);
this.menuStrip1.Items.Add(this.printPreviewToolStripMenuItem);
this.menuStrip1.Items.Add(this.toolStripMenuItemSeparator2);
this.menuStrip1.Items.Add(this.exitToolStripMenuItem);
this.menuStrip1.Name = "fileToolStripMenuItem";
this.menuStrip1.Size = new System.Drawing.Size(35, 20);
this.menuStrip1.Text = "&File";
this.menuStrip1.DropDownItemClicked += new
System.Windows.Forms.ToolStripItemClickedEventHandler(this.menuStrip1_DropDownItemClicked);
//  

// newToolStripMenuItem  

//  

this.newToolStripMenuItem.Image = ((System.Drawing.Image)
(resources.GetObject("newToolStripMenuItem.Image")));
this.newToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Magenta;
this.newToolStripMenuItem.Name = "newToolStripMenuItem";
this.newToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)
((System.Windows.Forms.Keys.Control | System.Windows.Forms.Keys.N)));
this.newToolStripMenuItem.Size = new System.Drawing.Size(152, 22);
this.newToolStripMenuItem.Text = "&New";
//  

// openToolStripMenuItem  

//  

this.openToolStripMenuItem.Image = ((System.Drawing.Image)
(resources.GetObject("openToolStripMenuItem.Image")));
this.openToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Magenta;
this.openToolStripMenuItem.Name = "openToolStripMenuItem";
this.openToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)
((System.Windows.Forms.Keys.Control | System.Windows.Forms.Keys.O)));
this.openToolStripMenuItem.Size = new System.Drawing.Size(152, 22);
this.openToolStripMenuItem.Text = "&Open";
//  

// toolStripSeparator  

//  

this.toolStripMenuItemSeparator.Name = "toolStripSeparator";
this.toolStripMenuItemSeparator.Size = new System.Drawing.Size(149, 6);
//  

// saveToolStripMenuItem  

//  

this.saveToolStripMenuItem.Image = ((System.Drawing.Image)
(resources.GetObject("saveToolStripMenuItem.Image")));
this.saveToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Magenta;
this.saveToolStripMenuItem.Name = "saveToolStripMenuItem";
this.saveToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)
((System.Windows.Forms.Keys.Control | System.Windows.Forms.Keys.S)));
this.saveToolStripMenuItem.Size = new System.Drawing.Size(152, 22);
this.saveToolStripMenuItem.Text = "&Save";
//  

// saveAsToolStripMenuItem  

//  

this.saveAsToolStripMenuItem.Name = "saveAsToolStripMenuItem";
this.saveAsToolStripMenuItem.Size = new System.Drawing.Size(152, 22);
this.saveAsToolStripMenuItem.Text = "Save &As";
//  

// toolStripSeparator1  

//  

this.toolStripMenuItemSeparator1.Name = "toolStripSeparator1";
this.toolStripMenuItemSeparator1.Size = new System.Drawing.Size(149, 6);
//  

// printToolStripMenuItem  

//  

this.printToolStripMenuItem.Image = ((System.Drawing.Image)
```

```

(resources.GetObject("printToolStripMenuItem.Image")));
    this.printToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Magenta;
    this.printToolStripMenuItem.Name = "printToolStripMenuItem";
    this.printToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)
((System.Windows.Forms.Keys.Control | System.Windows.Forms.Keys.P)));
    this.printToolStripMenuItem.Size = new System.Drawing.Size(152, 22);
    this.printToolStripMenuItem.Text = "&Print";
    //
    // printPreviewToolStripMenuItem
    //
    this.printPreviewToolStripMenuItem.Image = ((System.Drawing.Image)
(resources.GetObject("printPreviewToolStripMenuItem.Image")));
    this.printPreviewToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Magenta;
    this.printPreviewToolStripMenuItem.Name = "printPreviewToolStripMenuItem";
    this.printPreviewToolStripMenuItem.Size = new System.Drawing.Size(152, 22);
    this.printPreviewToolStripMenuItem.Text = "Print Pre&view";
    //
    // toolStripSeparator2
    //
    this.toolStripSeparator2.Name = "toolStripSeparator2";
    this.toolStripSeparator2.Size = new System.Drawing.Size(149, 6);
    //
    // exitToolStripMenuItem
    //
    this.exitToolStripMenuItem.Name = "exitToolStripMenuItem";
    this.exitToolStripMenuItem.Size = new System.Drawing.Size(152, 22);
    this.exitToolStripMenuItem.Text = "E&xit";
    //
    // editToolStripMenuItem
    //
    this.editToolStripMenuItem.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] {
this.undoToolStripMenuItem,
this.redoToolStripMenuItem,
this.toolStripSeparator3,
this.cutToolStripMenuItem,
this.copyToolStripMenuItem,
this.pasteToolStripMenuItem,
this.toolStripSeparator4,
this.selectAllToolStripMenuItem);
    this.editToolStripMenuItem.Name = "editToolStripMenuItem";
    this.editToolStripMenuItem.Size = new System.Drawing.Size(37, 20);
    this.editToolStripMenuItem.Text = "&Edit";
    //
    // undoToolStripMenuItem
    //
    this.undoToolStripMenuItem.Name = "undoToolStripMenuItem";
    this.undoToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)
((System.Windows.Forms.Keys.Control | System.Windows.Forms.Keys.Z)));
    this.undoToolStripMenuItem.Size = new System.Drawing.Size(150, 22);
    this.undoToolStripMenuItem.Text = "&Undo";
    //
    // redoToolStripMenuItem
    //
    this.redoToolStripMenuItem.Name = "redoToolStripMenuItem";
    this.redoToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)
((System.Windows.Forms.Keys.Control | System.Windows.Forms.Keys.Y)));
    this.redoToolStripMenuItem.Size = new System.Drawing.Size(150, 22);
    this.redoToolStripMenuItem.Text = "&Redo";
    //
    // toolStripSeparator3
    //
    this.toolStripSeparator3.Name = "toolStripSeparator3";
    this.toolStripSeparator3.Size = new System.Drawing.Size(147, 6);
    //
    // cutToolStripMenuItem
    //
    this.cutToolStripMenuItem.Image = ((System.Drawing.Image)
(resources.GetObject("cutToolStripMenuItem.Image")));
    this.cutToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Magenta;

```

```
this.cutToolStripMenuItem.Name = "cutToolStripMenuItem";
this.cutToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)
((System.Windows.Forms.Keys.Control | System.Windows.Forms.Keys.X)));
this.cutToolStripMenuItem.Size = new System.Drawing.Size(150, 22);
this.cutToolStripMenuItem.Text = "Cu&t";
//
// copyToolStripMenuItem
//
this.copyToolStripMenuItem.Image = ((System.Drawing.Image)
(resources.GetObject("copyToolStripMenuItem.Image")));
this.copyToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Magenta;
this.copyToolStripMenuItem.Name = "copyToolStripMenuItem";
this.copyToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)
((System.Windows.Forms.Keys.Control | System.Windows.Forms.Keys.C)));
this.copyToolStripMenuItem.Size = new System.Drawing.Size(150, 22);
this.copyToolStripMenuItem.Text = "&Copy";
//
// pasteToolStripMenuItem
//
this.pasteToolStripMenuItem.Image = ((System.Drawing.Image)
(resources.GetObject("pasteToolStripMenuItem.Image")));
this.pasteToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Magenta;
this.pasteToolStripMenuItem.Name = "pasteToolStripMenuItem";
this.pasteToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)
((System.Windows.Forms.Keys.Control | System.Windows.Forms.Keys.V)));
this.pasteToolStripMenuItem.Size = new System.Drawing.Size(150, 22);
this.pasteToolStripMenuItem.Text = "&Paste";
//
// toolStripSeparator4
//
this.toolStripSeparator4.Name = "toolStripSeparator4";
this.toolStripSeparator4.Size = new System.Drawing.Size(147, 6);
//
// selectAllToolStripMenuItem
//
this.selectAllToolStripMenuItem.Name = "selectAllToolStripMenuItem";
this.selectAllToolStripMenuItem.Size = new System.Drawing.Size(150, 22);
this.selectAllToolStripMenuItem.Text = "Select &All";
//
// toolsToolStripMenuItem
//
this.toolsToolStripMenuItem.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] {
this.customizeToolStripMenuItem,
this.optionsToolStripMenuItem});
this.toolsToolStripMenuItem.Name = "toolsToolStripMenuItem";
this.toolsToolStripMenuItem.Size = new System.Drawing.Size(44, 20);
this.toolsToolStripMenuItem.Text = "&Tools";
//
// customizeToolStripMenuItem
//
this.customizeToolStripMenuItem.Name = "customizeToolStripMenuItem";
this.customizeToolStripMenuItem.Size = new System.Drawing.Size(134, 22);
this.customizeToolStripMenuItem.Text = "&Customize";
//
// optionsToolStripMenuItem
//
this.optionsToolStripMenuItem.Name = "optionsToolStripMenuItem";
this.optionsToolStripMenuItem.Size = new System.Drawing.Size(134, 22);
this.optionsToolStripMenuItem.Text = "&Options";
//
// helpToolStripMenuItem
//
this.helpToolStripMenuItem.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] {
this.contentsToolStripMenuItem,
this.indexToolStripMenuItem,
this.searchToolStripMenuItem,
this.toolStripSeparator5,
this.aboutToolStripMenuItem});
this.helpToolStripMenuItem.Name = "helpToolStripMenuItem";
```

```
this.helpToolStripMenuItem.Size = new System.Drawing.Size(40, 20);
this.helpToolStripMenuItem.Text = "&Help";
//
// contentsToolStripMenuItem
//
this.contentsToolStripMenuItem.Name = "contentsToolStripMenuItem";
this.contentsToolStripMenuItem.Size = new System.Drawing.Size(129, 22);
this.contentsToolStripMenuItem.Text = "&Contents";
//
// indexToolStripMenuItem
//
this.indexToolStripMenuItem.Name = "indexToolStripMenuItem";
this.indexToolStripMenuItem.Size = new System.Drawing.Size(129, 22);
this.indexToolStripMenuItem.Text = "&Index";
//
// searchToolStripMenuItem
//
this.searchToolStripMenuItem.Name = "searchToolStripMenuItem";
this.searchToolStripMenuItem.Size = new System.Drawing.Size(129, 22);
this.searchToolStripMenuItem.Text = "&Search";
//
// toolStripSeparator5
//
this.toolStripSeparator5.Name = "toolStripSeparator5";
this.toolStripSeparator5.Size = new System.Drawing.Size(126, 6);
//
// aboutToolStripMenuItem
//
this.aboutToolStripMenuItem.Name = "aboutToolStripMenuItem";
this.aboutToolStripMenuItem.Size = new System.Drawing.Size(129, 22);
this.aboutToolStripMenuItem.Text = "&About... ";
//
// statusStrip1
//
this.statusStrip1.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
this.toolStripStatusLabel1});
this.statusStrip1.Location = new System.Drawing.Point(0, 251);
this.statusStrip1.Name = "statusStrip1";
this.statusStrip1.Size = new System.Drawing.Size(292, 22);
this.statusStrip1.TabIndex = 1;
this.statusStrip1.Text = "statusStrip1";
//
// toolStripStatusLabel1
//
this.toolStripStatusLabel1.Name = "toolStripStatusLabel1";
this.toolStripStatusLabel1.Size = new System.Drawing.Size(109, 17);
this.toolStripStatusLabel1.Text = "toolStripStatusLabel1";
//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(292, 273);
this.Controls.Add(this.statusStrip1);
this.Controls.Add(this.menuStrip1);
this.MainMenuStrip = this.menuStrip1;
this.Name = "Form1";
this.Text = "Form1";
this.menuStrip1.ResumeLayout(false);
this.menuStrip1.PerformLayout();
this.statusStrip1.ResumeLayout(false);
this.statusStrip1.PerformLayout();
this.ResumeLayout(false);
this.PerformLayout();

}

#endif
```

```

private System.Windows.Forms.MenuStrip menuStrip1;
private System.Windows.Forms.ToolStripMenuItem fileToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem newToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem openToolStripMenuItem;
private System.Windows.Forms.ToolStripSeparator toolStripSeparator;
private System.Windows.Forms.ToolStripMenuItem saveToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem saveAsToolStripMenuItem;
private System.Windows.Forms.ToolStripSeparator toolStripSeparator1;
private System.Windows.Forms.ToolStripMenuItem printToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem printPreviewToolStripMenuItem;
private System.Windows.Forms.ToolStripSeparator toolStripSeparator2;
private System.Windows.Forms.ToolStripMenuItem exitToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem editToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem undoToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem redoToolStripMenuItem;
private System.Windows.Forms.ToolStripSeparator toolStripSeparator3;
private System.Windows.Forms.ToolStripMenuItem cutToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem copyToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem pasteToolStripMenuItem;
private System.Windows.Forms.ToolStripSeparator toolStripSeparator4;
private System.Windows.Forms.ToolStripMenuItemselectAllToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem toolsToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem customizeToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem optionsToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem helpToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem contentsToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem indexToolStripMenuItem;
private System.Windows.Forms.ToolStripMenuItem searchToolStripMenuItem;
private System.Windows.Forms.ToolStripSeparator toolStripSeparator5;
private System.Windows.Forms.ToolStripMenuItem aboutToolStripMenuItem;

}

static class Program
{
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.Run(new Form1());
    }
}
}

```

```

Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Drawing
Imports System.Windows.Forms

Public Class Form1
    Inherits Form

    Public Sub New()
        Me.InitializeComponent()
    End Sub

    ' This utility method assigns the value of a ToolStripItem
    ' control's Text property to the Text property of the
    ' ToolStripStatusLabel.
    Private Sub UpdateStatus(ByVal item As ToolStripItem)

```

```

If item IsNot Nothing Then

    Dim msg As String = String.Format("{0} selected", item.Text)
    Me.StatusStrip1.Items(0).Text = msg

End If

End Sub

' This method is the DropDownItemClicked event handler.
' It passes the ClickedItem object to a utility method
' called UpdateStatus, which updates the text displayed
' in the StatusStrip control.
Private Sub FileToolStripMenuItem_DropDownItemClicked( _
 ByVal sender As System.Object, _
 ByVal e As System.Windows.Forms.ToolStripItemClickedEventArgs) _
 Handles FileToolStripMenuItem.DropDownItemClicked

    Me.UpdateStatus(e.ClickedItem)

End Sub

'Form overrides dispose to clean up the component list.
<System.Diagnostics.DebuggerNonUserCode()> _
Protected Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing AndAlso components IsNot Nothing Then
        components.Dispose()
    End If
    MyBase.Dispose(disposing)
End Sub

Friend WithEvents MenuStrip1 As System.Windows.Forms.MenuStrip
Friend WithEvents FileToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents NewToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents OpenToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents toolStripSeparator As System.Windows.Forms.ToolStripSeparator
Friend WithEvents SaveToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents SaveAsToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents toolStripSeparator1 As System.Windows.Forms.ToolStripSeparator
Friend WithEvents PrintToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents PrintPreviewToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents toolStripSeparator2 As System.Windows.Forms.ToolStripSeparator
Friend WithEvents ExitToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents EditToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents UndoToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents RedoToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents toolStripSeparator3 As System.Windows.Forms.ToolStripSeparator
Friend WithEvents CutToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents CopyToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents PasteToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents toolStripSeparator4 As System.Windows.Forms.ToolStripSeparator
Friend WithEvents SelectAllToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents ToolsToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents CustomizeToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents OptionsToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents HelpToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents ContentsToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents IndexToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents SearchToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents toolStripSeparator5 As System.Windows.Forms.ToolStripSeparator
Friend WithEvents AboutToolStripMenuItem As System.Windows.Forms.ToolStripItem
Friend WithEvents StatusStrip1 As System.Windows.Forms.StatusStrip
Friend WithEvents ToolStripStatusLabel1 As System.Windows.Forms.ToolStripStatusLabel

'Required by the Windows Form Designer
Private components As System.ComponentModel.IContainer

'NOTE: The following procedure is required by the Windows Form Designer
'It can be modified using the Windows Form Designer.

```

```

'Do not modify it using the code editor.
<System.Diagnostics.DebuggerStepThrough()> _
Private Sub InitializeComponent()
    Dim resources As System.ComponentModel.ComponentResourceManager = New
System.ComponentModel.ComponentResourceManager(GetType(Form1))
    Me.MenuStrip1 = New System.Windows.Forms.MenuStrip
    Me.FileToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.NewToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.OpenToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.toolStripSeparator = New System.Windows.Forms.ToolStripSeparator
    Me.SaveToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.SaveAsToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.toolStripSeparator1 = New System.Windows.Forms.ToolStripSeparator
    Me.PrintToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.PrintPreviewToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.toolStripSeparator2 = New System.Windows.Forms.ToolStripSeparator
    Me.ExitToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.EditToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.UndoToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.RedoToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.toolStripSeparator3 = New System.Windows.Forms.ToolStripSeparator
    Me.CutToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.CopyToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.PasteToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.toolStripSeparator4 = New System.Windows.Forms.ToolStripSeparator
    Me.SelectAllToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.ToolsToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.CustomizeToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.OptionsToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.HelpToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.ContentsToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.IndexToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.SearchToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.toolStripSeparator5 = New System.Windows.Forms.ToolStripSeparator
    Me.AboutToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.StatusStrip1 = New System.Windows.Forms.StatusStrip
    Me.ToolStripStatusLabel1 = New System.Windows.Forms.ToolStripStatusLabel
    Me.MenuStrip1.SuspendLayout()
    Me.StatusStrip1.SuspendLayout()
    Me.SuspendLayout()
    '
    'MenuStrip1
    '
    Me.MenuStrip1.Items.AddRange(New System.Windows.Forms.ToolStripItem() {Me.FileToolStripMenuItem,
Me.EditToolStripMenuItem, Me.ToolsToolStripMenuItem, Me.HelpToolStripMenuItem})
    Me.MenuStrip1.Location = New System.Drawing.Point(0, 0)
    Me.MenuStrip1.Name = "MenuStrip1"
    Me.MenuStrip1.Size = New System.Drawing.Size(292, 24)
    Me.MenuStrip1.TabIndex = 0
    Me.MenuStrip1.Text = "MenuStrip1"
    '
    'FileToolStripMenuItem
    '
    Me.FileToolStripMenuItem.DropDownItems.AddRange(New System.Windows.Forms.ToolStripItem()
{Me.NewToolStripMenuItem, Me.OpenToolStripMenuItem, Me.toolStripSeparator, Me.SaveToolStripMenuItem,
Me.SaveAsToolStripMenuItem, Me.toolStripSeparator1, Me.PrintToolStripMenuItem,
Me.PrintPreviewToolStripMenuItem, Me.toolStripSeparator2, Me.ExitToolStripMenuItem})
    Me.FileToolStripMenuItem.Name = "FileToolStripMenuItem"
    Me.FileToolStripMenuItem.Size = New System.Drawing.Size(35, 20)
    Me.FileToolStripMenuItem.Text = "&File"
    '
    'NewToolStripMenuItem
    '
    Me.NewToolStripMenuItem.Image = CType(resources.GetObject("NewToolStripMenuItem.Image"),
System.Drawing.Image)
    Me.NewToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Magenta
    Me.NewToolStripMenuItem.Name = "NewToolStripMenuItem"
    Me.NewToolStripMenuItem.ShortcutKeys = CType((System.Windows.Forms.Keys.Control Or
System.Windows.Forms.Keys.N), System.Windows.Forms.Keys)

```

```

Me.NewToolStripMenuItem.Size = New System.Drawing.Size(152, 22)
Me.NewToolStripMenuItem.Text = "&New"
'
'OpenToolStripMenuItem
'

Me.OpenToolStripMenuItem.Image = CType(resources.GetObject("OpenToolStripMenuItem.Image"),
System.Drawing.Image)
Me.OpenToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Magenta
Me.OpenToolStripMenuItem.Name = "OpenToolStripMenuItem"
Me.OpenToolStripMenuItem.ShortcutKeys = CType((System.Windows.Forms.Keys.Control Or
System.Windows.Forms.Keys.O), System.Windows.Forms.Keys)
Me.OpenToolStripMenuItem.Size = New System.Drawing.Size(152, 22)
Me.OpenToolStripMenuItem.Text = "&Open"
'
'toolStripSeparator
'

Me.toolStripButton.Name = "toolStripSeparator"
Me.toolStripButton.Size = New System.Drawing.Size(149, 6)
'
'SaveToolStripMenuItem
'

Me.SaveToolStripMenuItem.Image = CType(resources.GetObject("SaveToolStripMenuItem.Image"),
System.Drawing.Image)
Me.SaveToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Magenta
Me.SaveToolStripMenuItem.Name = "SaveToolStripMenuItem"
Me.SaveToolStripMenuItem.ShortcutKeys = CType((System.Windows.Forms.Keys.Control Or
System.Windows.Forms.Keys.S), System.Windows.Forms.Keys)
Me.SaveToolStripMenuItem.Size = New System.Drawing.Size(152, 22)
Me.SaveToolStripMenuItem.Text = "&Save"
'
'SaveAsToolStripMenuItem
'

Me.SaveAsToolStripMenuItem.Name = "SaveAsToolStripMenuItem"
Me.SaveAsToolStripMenuItem.Size = New System.Drawing.Size(152, 22)
Me.SaveAsToolStripMenuItem.Text = "Save &As"
'
'toolStripSeparator1
'

Me.toolStripButton1.Name = "toolStripSeparator1"
Me.toolStripButton1.Size = New System.Drawing.Size(149, 6)
'
'PrintToolStripMenuItem
'

Me.PrintToolStripMenuItem.Image = CType(resources.GetObject("PrintToolStripMenuItem.Image"),
System.Drawing.Image)
Me.PrintToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Magenta
Me.PrintToolStripMenuItem.Name = "PrintToolStripMenuItem"
Me.PrintToolStripMenuItem.ShortcutKeys = CType((System.Windows.Forms.Keys.Control Or
System.Windows.Forms.Keys.P), System.Windows.Forms.Keys)
Me.PrintToolStripMenuItem.Size = New System.Drawing.Size(152, 22)
Me.PrintToolStripMenuItem.Text = "&Print"
'
'PrintPreviewToolStripMenuItem
'

Me.PrintPreviewToolStripMenuItem.Image =
 CType(resources.GetObject("PrintPreviewToolStripMenuItem.Image"), System.Drawing.Image)
Me.PrintPreviewToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Magenta
Me.PrintPreviewToolStripMenuItem.Name = "PrintPreviewToolStripMenuItem"
Me.PrintPreviewToolStripMenuItem.Size = New System.Drawing.Size(152, 22)
Me.PrintPreviewToolStripMenuItem.Text = "Print Pre&view"
'
'toolStripSeparator2
'

Me.toolStripButton2.Name = "toolStripSeparator2"
Me.toolStripButton2.Size = New System.Drawing.Size(149, 6)
'
'ExitToolStripMenuItem
'

Me.ExitToolStripMenuItem.Name = "ExitToolStripMenuItem"

```

```

Me.ExitToolStripMenuItem.Name = "ExitToolStripMenuItem"
Me.ExitToolStripMenuItem.Size = New System.Drawing.Size(152, 22)
Me.ExitToolStripMenuItem.Text = "E&xit"
'

'EditToolStripMenuItem
'

Me.EditToolStripMenuItem.DropDownItems.AddRange(New System.Windows.Forms.ToolStripItem()
{Me.UndoToolStripMenuItem, Me.RedoToolStripMenuItem, Me.toolStripButton3, Me.CutToolStripMenuItem,
Me.CopyToolStripMenuItem, Me.PasteToolStripMenuItem, Me.toolStripButton4, Me.SelectAllToolStripMenuItem})
    Me.EditToolStripMenuItem.Name = "EditToolStripMenuItem"
    Me.EditToolStripMenuItem.Size = New System.Drawing.Size(37, 20)
    Me.EditToolStripMenuItem.Text = "&Edit"
'

'UndoToolStripMenuItem
'

Me.UndoToolStripMenuItem.Name = "UndoToolStripMenuItem"
Me.UndoToolStripMenuItem.ShortcutKeys = CType((System.Windows.Forms.Keys.Control Or
System.Windows.Forms.Keys.Z), System.Windows.Forms.Keys)
    Me.UndoToolStripMenuItem.Size = New System.Drawing.Size(150, 22)
    Me.UndoToolStripMenuItem.Text = "&Undo"
'

'RedoToolStripMenuItem
'

Me.RedoToolStripMenuItem.Name = "RedoToolStripMenuItem"
Me.RedoToolStripMenuItem.ShortcutKeys = CType((System.Windows.Forms.Keys.Control Or
System.Windows.Forms.Keys.Y), System.Windows.Forms.Keys)
    Me.RedoToolStripMenuItem.Size = New System.Drawing.Size(150, 22)
    Me.RedoToolStripMenuItem.Text = "&Redo"
'

'toolStripSeparator3
'

Me.toolStripButton3.Name = "toolStripSeparator3"
Me.toolStripButton3.Size = New System.Drawing.Size(147, 6)
'

'CutToolStripMenuItem
'

Me.CutToolStripMenuItem.Image = CType(resources.GetObject("CutToolStripMenuItem.Image"),
System.Drawing.Image)
    Me.CutToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Magenta
    Me.CutToolStripMenuItem.Name = "CutToolStripMenuItem"
    Me.CutToolStripMenuItem.ShortcutKeys = CType((System.Windows.Forms.Keys.Control Or
System.Windows.Forms.Keys.X), System.Windows.Forms.Keys)
        Me.CutToolStripMenuItem.Size = New System.Drawing.Size(150, 22)
        Me.CutToolStripMenuItem.Text = "Cu&t"
'

'CopyToolStripMenuItem
'

Me.CopyToolStripMenuItem.Image = CType(resources.GetObject("CopyToolStripMenuItem.Image"),
System.Drawing.Image)
    Me.CopyToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Magenta
    Me.CopyToolStripMenuItem.Name = "CopyToolStripMenuItem"
    Me.CopyToolStripMenuItem.ShortcutKeys = CType((System.Windows.Forms.Keys.Control Or
System.Windows.Forms.Keys.C), System.Windows.Forms.Keys)
        Me.CopyToolStripMenuItem.Size = New System.Drawing.Size(150, 22)
        Me.CopyToolStripMenuItem.Text = "&Copy"
'

'PasteToolStripMenuItem
'

Me.PasteToolStripMenuItem.Image = CType(resources.GetObject("PasteToolStripMenuItem.Image"),
System.Drawing.Image)
    Me.PasteToolStripMenuItem.ImageTransparentColor = System.Drawing.Color.Magenta
    Me.PasteToolStripMenuItem.Name = "PasteToolStripMenuItem"
    Me.PasteToolStripMenuItem.ShortcutKeys = CType((System.Windows.Forms.Keys.Control Or
System.Windows.Forms.Keys.V), System.Windows.Forms.Keys)
        Me.PasteToolStripMenuItem.Size = New System.Drawing.Size(150, 22)
        Me.PasteToolStripMenuItem.Text = "&Paste"
'

'toolStripSeparator4
'

```

```

Me.toolStripSeparator4.Name = "toolStripSeparator4"
Me.toolStripSeparator4.Size = New System.Drawing.Size(147, 6)
'

'SelectAllToolStripMenuItem

Me.SelectAllToolStripMenuItem.Name = "SelectAllToolStripMenuItem"
Me.SelectAllToolStripMenuItem.Size = New System.Drawing.Size(150, 22)
Me.SelectAllToolStripMenuItem.Text = "Select &All"
'

'ToolsToolStripMenuItem

Me.ToolsToolStripMenuItem.DropDownItems.AddRange(New System.Windows.Forms.ToolStripItem()
{Me.CustomizeToolStripMenuItem, Me.OptionsToolStripMenuItem})
Me.ToolsToolStripMenuItem.Name = "ToolsToolStripMenuItem"
Me.ToolsToolStripMenuItem.Size = New System.Drawing.Size(44, 20)
Me.ToolsToolStripMenuItem.Text = "&Tools"
'

'CustomizeToolStripMenuItem

Me.CustomizeToolStripMenuItem.Name = "CustomizeToolStripMenuItem"
Me.CustomizeToolStripMenuItem.Size = New System.Drawing.Size(134, 22)
Me.CustomizeToolStripMenuItem.Text = "&Customize"
'

'OptionsToolStripMenuItem

Me.OptionsToolStripMenuItem.Name = "OptionsToolStripMenuItem"
Me.OptionsToolStripMenuItem.Size = New System.Drawing.Size(134, 22)
Me.OptionsToolStripMenuItem.Text = "&Options"
'

'HelpToolStripMenuItem

Me.HelpToolStripMenuItem.DropDownItems.AddRange(New System.Windows.Forms.ToolStripItem()
{Me.ContentsToolStripMenuItem, Me.IndexToolStripMenuItem, Me.SearchToolStripMenuItem, Me.toolStripSeparator5,
Me.AboutToolStripMenuItem})
Me.HelpToolStripMenuItem.Name = "HelpToolStripMenuItem"
Me.HelpToolStripMenuItem.Size = New System.Drawing.Size(40, 20)
Me.HelpToolStripMenuItem.Text = "&Help"
'

'ContentsToolStripMenuItem

Me.ContentsToolStripMenuItem.Name = "ContentsToolStripMenuItem"
Me.ContentsToolStripMenuItem.Size = New System.Drawing.Size(152, 22)
Me.ContentsToolStripMenuItem.Text = "&Contents"
'

'IndexToolStripMenuItem

Me.IndexToolStripMenuItem.Name = "IndexToolStripMenuItem"
Me.IndexToolStripMenuItem.Size = New System.Drawing.Size(152, 22)
Me.IndexToolStripMenuItem.Text = "&Index"
'

'SearchToolStripMenuItem

Me.SearchToolStripMenuItem.Name = "SearchToolStripMenuItem"
Me.SearchToolStripMenuItem.Size = New System.Drawing.Size(152, 22)
Me.SearchToolStripMenuItem.Text = "&Search"
'

'toolStripSeparator5

Me.toolStripSeparator5.Name = "toolStripSeparator5"
Me.toolStripSeparator5.Size = New System.Drawing.Size(149, 6)
'

'AboutToolStripMenuItem

Me.AboutToolStripMenuItem.Name = "AboutToolStripMenuItem"
Me.AboutToolStripMenuItem.Size = New System.Drawing.Size(152, 22)
Me.AboutToolStripMenuItem.Text = "&About..."
'

'StatusStrip1
'

```

```

Me.StatusStrip1.Items.AddRange(New System.Windows.Forms.ToolStripItem() {Me.ToolStripStatusLabel1})
Me.StatusStrip1.Location = New System.Drawing.Point(0, 251)
Me.StatusStrip1.Name = "StatusStrip1"
Me.StatusStrip1.Size = New System.Drawing.Size(292, 22)
Me.StatusStrip1.TabIndex = 1
Me.StatusStrip1.Text = "StatusStrip1"
'
'ToolStripStatusLabel1
'
Me.ToolStripStatusLabel1.Name = "ToolStripStatusLabel1"
Me.ToolStripStatusLabel1.Size = New System.Drawing.Size(111, 17)
Me.ToolStripStatusLabel1.Text = "ToolStripStatusLabel1"
'
'Form1
'

Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)
Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
Me.ClientSize = New System.Drawing.Size(292, 273)
Me.Controls.Add(Me.StatusStrip1)
Me.Controls.Add(Me.MenuStrip1)
Me.MainMenuStrip = Me.MenuStrip1
Me.Name = "Form1"
Me.Text = "Form1"
Me.MenuStrip1.ResumeLayout(False)
Me.MenuStrip1.PerformLayout()
Me.StatusStrip1.ResumeLayout(False)
Me.StatusStrip1.PerformLayout()
Me.ResumeLayout(False)
Me.PerformLayout()

End Sub

<STAThread()> _
Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())
End Sub

End Class

```

Compiling the Code

This example requires:

- References to the System, System.Drawing and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[MenuStrip](#)

[ToolStrip](#)

[StatusStrip](#)

[MenuStrip Control](#)

How to: Set Up Automatic Menu Merging for MDI Applications

5/4/2018 • 1 min to read • [Edit Online](#)

The following procedure gives the basic steps for setting up automatic merging in a multiple-document interface (MDI) application with [MenuStrip](#).

To set up automatic menu merging

1. Create the MDI parent form by setting its [IsMdiContainer](#) property to `true`.
2. Add a [MenuStrip](#) to the MDI parent, setting its [MainMenuStrip](#) property to that [MenuStrip](#).
3. Create an MDI child form, and set its [MdiParent](#) property to the name of the parent form.
4. Add a [MenuStrip](#) to the MDI child form.
5. On the child form, set the [Visible](#) property of the [MenuStrip](#) to `false`.
6. Add menu items to the child form's [MenuStrip](#) that you want to merge into the parent form's [MenuStrip](#) when the child form is activated.
7. Use the [MergeAction](#) property on the menu items in the child form's [MenuStrip](#) to control how they merge into the parent form.

See Also

[MenuStrip](#)

[ToolStripMenuItem](#)

[MenuStrip Control Overview](#)

Merging Menu Items in the Windows Forms ToolStrip Control

5/4/2018 • 3 min to read • [Edit Online](#)

If you have a multiple-document interface (MDI) application, you can merge menu items or whole menus from the child form into the menus of the parent form.

This topic describes the basic concepts associated with merging menu items in an MDI application.

General Concepts

Merging procedures involve both a target and a source control:

- The target is the [MenuStrip](#) control on the main or MDI parent form into which you are merging menu items.
- The source is the [MenuStrip](#) control on the MDI child form that contains the menu items you want to merge into the target menu.

The [MdiWindowListItem](#) property identifies the menu item whose drop-down list you will populate with the titles of the current MDI parent form's MDI children. For example, you typically list MDI children that are currently open on the **Window** menu.

The [IsMdiWindowListEntry](#) property identifies which menu items come from a [MenuStrip](#) on an MDI child form.

You can merge menu items manually or automatically. The menu items merge in the same way for both methods, but the merge is activated differently, as discussed in the "Manual Merging" and "Automatic Merging" sections later in this topic. In both manual and automatic merging, each merge action affects the next merge action.

[MenuStrip](#) merging moves menu items from one [ToolStrip](#) to another rather than cloning them, as was the case with [MainMenu](#).

MergeAction Values

You set the merge action on menu items in the source [MenuStrip](#) using the [MergeAction](#) property.

The following table describes the meaning and typical use of the available merge actions.

MERGEACTION VALUE	DESCRIPTION	TYPICAL USE
Append	(Default) Adds the source item to the end of the target item's collection.	Adding menu items to the end of the menu when some part of the program is activated.
Insert	Adds the source item to the target item's collection, in the location specified by the MergeIndex property set on the source item.	Adding menu items to the middle or the beginning of the menu when some part of the program is activated. If the value of MergeIndex is the same for both menu items, they are added in reverse order. Set MergeIndex appropriately to preserve the original order.

MERGEACTION VALUE	DESCRIPTION	TYPICAL USE
Replace	Finds a text match, or uses the MergeIndex value if no text match is found, and then replaces the matching target menu item with the source menu item.	Replacing a target menu item with a source menu item of the same name that does something different.
MatchOnly	Finds a text match, or uses the MergeIndex value if no text match is found, and then adds all the drop-down items from the source to the target.	Building a menu structure that inserts or adds menu items into a submenu, or removes menu items from a submenu. For example, you can add a menu item from an MDI child to a main MenuStripSave As menu. MatchOnly allows you to navigate through the menu structure without taking any action. It provides a way to evaluate the subsequent items.
Remove	Finds a text match, or uses the MergeIndex value if no text match is found, and then removes the item from the target.	Removing a menu item from the target MenuStrip .

Manual Merging

Only [MenuStrip](#) controls participate in automatic merging. To combine the items of other controls, such as [ToolStrip](#) and [StatusStrip](#) controls, you must merge them manually, by calling the [Merge](#) and [RevertMerge](#) methods in your code as required.

Automatic Merging

You can use automatic merging for MDI applications by activating the source form. To use a [MenuStrip](#) in an MDI application, set the [MainMenuStrip](#) property to the target [MenuStrip](#) so that merging actions performed on the source [MenuStrip](#) are reflected in the target [MenuStrip](#).

You can trigger automatic merging by activating the [MenuStrip](#) on the MDI source. Upon activation, the source [MenuStrip](#) is merged into the MDI target. When a new form becomes active, the merge is reverted on the last form and triggered on the new form. You can control this behavior by setting the [MergeAction](#) property as needed on each [ToolStripItem](#), and by setting the [AllowMerge](#) property on each [MenuStrip](#).

See Also

[ToolStripManager](#)

[MenuStrip](#)

[MenuStrip Control](#)

[How to: Create an MDI Window List with MenuStrip](#)

[How to: Set Up Automatic Menu Merging for MDI Applications](#)

Walkthrough: Providing Standard Menu Items to a Form

5/4/2018 • 3 min to read • [Edit Online](#)

You can provide a standard menu for your forms with the [MenuStrip](#) control.

This walkthrough demonstrates how to use a [MenuStrip](#) control to create a standard menu. The form also responds when a user selects a menu item. The following tasks are illustrated in this walkthrough:

- Creating a Windows Forms project.
- Creating a standard menu.
- Creating a [StatusStrip](#) control.
- Handling menu item selection.

When you are finished, you will have a form with a standard menu that displays menu item selections in a [StatusStrip](#) control.

To copy the code in this topic as a single listing, see [How to: Provide Standard Menu Items to a Form](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

Prerequisites

In order to complete this walkthrough, you will need:

- Sufficient permissions to be able to create and run Windows Forms application projects on the computer where Visual Studio is installed.

Creating the Project

The first step is to create the project and set up the form.

To create the project

1. Create a Windows application project called **StandardMenuItemForm**.

For more information, see [How to: Create a Windows Application Project](#).

2. In the Windows Forms Designer, select the form.

Creating a Standard Menu

The Windows Forms Designer can automatically populate a [MenuStrip](#) control with standard menu items.

To create a standard menu

1. From the **Toolbox**, drag a [MenuStrip](#) control onto your form.
2. Click the [MenuStrip](#) control's smart tag glyph (ⓘ) and select **Insert Standard Items**.

The [MenuStrip](#) control is populated with the standard menu items.

3. Click the **File** menu item to see its default menu items and corresponding icons.

Creating a StatusStrip Control

Use the [StatusStrip](#) control to display status for your Windows Forms applications. In the current example, menu items selected by the user are displayed in a [StatusStrip](#) control.

To create a StatusStrip control

1. From the **Toolbox**, drag a [StatusStrip](#) control onto your form.

The [StatusStrip](#) control automatically docks to the bottom of the form.

2. Click the [StatusStrip](#) control's drop-down button and select **StatusLabel** to add a [ToolStripStatusLabel](#) control to the [StatusStrip](#) control.

Handling Item Selection

Handle the [DropDownItemClicked](#) event to respond when the user selects a menu item.

To handle item selection

1. Click the **File** menu item that you created in the Creating a Standard Menu section.
2. In the **Properties** window, click **Events**.
3. Double-click the [DropDownItemClicked](#) event.

The Windows Forms Designer generates an event handler for the [DropDownItemClicked](#) event.

4. Insert the following code into the event handler.

```
// This method is the DropDownItemClicked event handler.  
// It passes the ClickedItem object to a utility method  
// called UpdateStatus, which updates the text displayed  
// in the StatusStrip control.  
private void fileToolStripMenuItem_DropDownItemClicked(  
    object sender, ToolStripItemClickedEventArgs e)  
{  
    this.UpdateStatus(e.ClickedItem);  
}
```

```
' This method is the DropDownItemClicked event handler.  
' It passes the ClickedItem object to a utility method  
' called UpdateStatus, which updates the text displayed  
' in the StatusStrip control.  
Private Sub FileToolStripMenuItem_DropDownItemClicked( _  
    ByVal sender As System.Object, _  
    ByVal e As System.Windows.Forms.ToolStripItemClickedEventArgs) _  
Handles FileToolStripMenuItem.DropDownItemClicked  
  
    Me.UpdateStatus(e.ClickedItem)  
  
End Sub
```

5. Insert the [UpdateStatus](#) utility method definition into the form.

```
// This utility method assigns the value of a ToolStripItem
// control's Text property to the Text property of the
// ToolStripStatusLabel.
private void UpdateStatus(ToolStripItem item)
{
    if (item != null)
    {
        string msg = String.Format("{0} selected", item.Text);
        this.statusStrip1.Items[0].Text = msg;
    }
}
```

```
' This utility method assigns the value of a ToolStripItem
' control's Text property to the Text property of the
' ToolStripStatusLabel.
Private Sub UpdateStatus(ByVal item As ToolStripItem)

    If item IsNot Nothing Then

        Dim msg As String = String.Format("{0} selected", item.Text)
        Me.StatusStrip1.Items(0).Text = msg

    End If

End Sub
```

Checkpoint

To test your form

1. Press F5 to compile and run your form.
2. Click the **File** menu item to open the menu.
3. On the **File** menu, click one of the items to select it.

The [StatusStrip](#) control displays the selected item.

Next Steps

In this walkthrough, you have created a form with a standard menu. You can use the [ToolStrip](#) family of controls for many other purposes:

- Create shortcut menus for your controls with [ContextMenuStrip](#). For more information, see [ContextMenu Component Overview](#).
- Create a multiple document interface (MDI) form with docking [ToolStrip](#) controls. For more information, see [Walkthrough: Creating an MDI Form with Menu Merging and ToolStrip Controls](#).
- Give your [ToolStrip](#) controls a professional appearance. For more information, see [How to: Set the ToolStrip Renderer for an Application](#).

See Also

[MenuStrip](#)

[ToolStrip](#)

[StatusStrip](#)

[MenuStrip Control](#)

MonthCalendar Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `MonthCalendar` control presents an intuitive graphical interface for users to view and set date information. The control displays a grid containing the numbered days of the month, arranged in columns underneath the days of the week. You can select a different month by clicking the arrow buttons on either side of the month caption. Unlike the similar [DateTimePicker](#) control, you can select a range of dates with this control; however, the [DateTimePicker](#) control allows you to set times as well as dates.

In This Section

[MonthCalendar Control Overview](#)

Introduces the general concepts of the `MonthCalendar` control, which allows users to view and set date information for an application.

[How to: Change the Windows Forms MonthCalendar Control's Appearance](#)

Describes how to customize the appearance of the `MonthCalendar` control.

[How to: Display More than One Month in the Windows Forms MonthCalendar Control](#)

Describes how to configure the `MonthCalendar` control to display several months simultaneously.

[How to: Display Specific Days in Bold with the Windows Forms MonthCalendar Control](#)

Explains how to set certain dates to appear bold.

[How to: Select a Range of Dates in the Windows Forms MonthCalendar Control](#)

Explains how to programmatically select a range of dates from the `MonthCalendar` control.

Reference

[MonthCalendar](#)

Provides reference information on the class and its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

[DateTimePicker Control](#)

Describes a control similar to [MonthCalendar](#), although the [DateTimePicker](#) control also allows you to select a time and does not allow you to select a range of dates.

MonthCalendar Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [MonthCalendar](#) control presents an intuitive graphical interface for users to view and set date information. The control displays a calendar: a grid containing the numbered days of the month, arranged in columns underneath the days of the week, with the selected range of dates highlighted. You can select a different month by clicking the arrow buttons on either side of the month caption. Unlike the similar [DateTimePicker](#) control, you can select more than one date with this control. For more information about the [DateTimePicker](#) control, see [DateTimePicker Control](#).

Configuring the MonthCalendar Control

The [MonthCalendar](#) control's appearance is highly configurable. By default, today's date is displayed as circled, and is also noted at the bottom of the grid. You can change this feature by setting the [ShowToday](#) and [ShowTodayCircle](#) properties to `false`. You can also add week numbers to the calendar by setting the [ShowWeekNumbers](#) property to `true`. By setting the [CalendarDimensions](#) property, you can have multiple months displayed horizontally and vertically. By default, Sunday is shown as the first day of the week, but any day can be designated using the [FirstDayOfWeek](#) property.

You can also set certain dates to be displayed in bold on a one-time basis, annually, or monthly, by adding [DateTime](#) objects to the [BoldedDates](#), [AnnuallyBoldedDates](#), and [MonthlyBoldedDates](#) properties. For more information, see [How to: Display Specific Days in Bold with the Windows Forms MonthCalendar Control](#).

The key property of the [MonthCalendar](#) control is [SelectionRange](#), the range of dates selected in the control. The [SelectionRange](#) value cannot exceed the maximum number of days that can be selected, set in the [MaxSelectionCount](#) property. The earliest and latest dates the user can select are determined by the [MaxDate](#) and [MinDate](#) properties.

See Also

[MonthCalendar](#)
[MonthCalendar Control](#)

How to: Change the Windows Forms MonthCalendar Control's Appearance

5/4/2018 • 2 min to read • [Edit Online](#)

The Windows Forms [MonthCalendar](#) control allows you to customize the calendar's appearance in many ways. For example, you can set the color scheme and choose to display or hide week numbers and the current date.

To change the month calendar's color scheme

- Set properties such as [TitleBackColor](#), [TitleForeColor](#) and [TrailingForeColor](#). The [TitleBackColor](#) property also determines the font color for the days of the week. The [TrailingForeColor](#) property determines the color of the dates that precede and follow the displayed month or months.

```
MonthCalendar1.TitleBackColor = System.Drawing.Color.Blue  
MonthCalendar1.TrailingForeColor = System.Drawing.Color.Red  
MonthCalendar1.TitleForeColor = System.Drawing.Color.Yellow
```

```
monthCalendar1.TitleBackColor = System.Drawing.Color.Blue;  
monthCalendar1.TrailingForeColor = System.Drawing.Color.Red;  
monthCalendar1.TitleForeColor = System.Drawing.Color.Yellow;
```

```
monthCalendar1->TitleBackColor = System::Drawing::Color::Blue;  
monthCalendar1->TrailingForeColor = System::Drawing::Color::Red;  
monthCalendar1->TitleForeColor = System::Drawing::Color::Yellow;
```

NOTE

Starting with Windows Vista and depending on the theme, setting some properties might not change the appearance of the calendar. For example, if Windows is set to use the Aero theme, setting the [BackColor](#), [TitleBackColor](#), [TitleForeColor](#), or [TrailingForeColor](#) properties has no effect. This is because an updated version of the calendar is rendered with an appearance that is derived at run time from the current operating system theme. If you want to use these properties and enable the earlier version of the calendar, you can disable visual styles for your application. Disabling visual styles might affect the appearance and behavior of other controls in your application. To disable visual styles in Visual Basic, open the Project Designer and uncheck the **Enable XP visual styles** check box. To disable visual styles in C#, open Program.cs and comment out `Application.EnableVisualStyles();`. For more information about visual styles, see [How to: Enable Windows XP Visual Styles](#).

To display the current date at the bottom of the control

- Set the [ShowToday](#) property to `true`. The example below toggles between displaying and omitting today's date when the form is double-clicked.

```
Private Sub Form1_DoubleClick(ByVal sender As Object, _  
    ByVal e As System.EventArgs) Handles MyBase.DoubleClick  
    ' Toggle between True and False.  
    MonthCalendar1.ShowToday = Not MonthCalendar1.ShowToday  
End Sub
```

```
private void Form1_DoubleClick(object sender, System.EventArgs e)
{
    // Toggle between True and False.
    monthCalendar1.ShowToday = !monthCalendar1.ShowToday;
}
```

```
private:
    System::Void Form1_DoubleClick(System::Object ^  sender,
                                   System::EventArgs ^  e)
    {
        // Toggle between True and False.
        monthCalendar1->ShowToday = !monthCalendar1->ShowToday;
    }
```

(Visual C#, Visual C++) Place the following code in the form's constructor to register the event handler.

```
this.DoubleClick += new System.EventHandler(this.Form1_DoubleClick);
```

```
this->DoubleClick += gcnew System::EventHandler(this,
    &Form1::Form1_DoubleClick);
```

To display week numbers

- Set the [ShowWeekNumbers](#) property to `true`. You can set this property either in code or in the Properties window.

Week numbers appear in a separate column to the left of the first day of the week.

```
MonthCalendar1.ShowWeekNumbers = True
```

```
monthCalendar1.ShowWeekNumbers = true;
```

```
monthCalendar1->ShowWeekNumbers = true;
```

See Also

[MonthCalendar Control](#)

[How to: Select a Range of Dates in the Windows Forms MonthCalendar Control](#)

[How to: Display Specific Days in Bold with the Windows Forms MonthCalendar Control](#)

[How to: Display More than One Month in the Windows Forms MonthCalendar Control](#)

How to: Display More than One Month in the Windows Forms MonthCalendar Control

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [MonthCalendar](#) control can display up to 12 months at a time. By default, the control displays only one month, but you can specify how many months are displayed and how they are arranged within the control. When you change the calendar dimensions, the control is resized, so be sure there is enough room on the form for the new dimensions.

To display multiple months

- Set the [CalendarDimensions](#) property to the number of months to display horizontally and vertically.

```
MonthCalendar1.CalendarDimensions = New System.Drawing.Size (3,2)
```

```
monthCalendar1.CalendarDimensions = new System.Drawing.Size (3,2);
```

```
monthCalendar1->CalendarDimensions = System::Drawing::Size (3,2);
```

See Also

[MonthCalendar Control](#)

[How to: Select a Range of Dates in the Windows Forms MonthCalendar Control](#)

[How to: Change the Windows Forms MonthCalendar Control's Appearance](#)

How to: Display Specific Days in Bold with the Windows Forms MonthCalendar Control

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [MonthCalendar](#) control can display days in bold type, either as singular dates or on a repeating basis. You might do this to draw attention to special dates, such as holidays and weekends.

Three properties control this feature. The [BoldedDates](#) property contains single dates. The [AnnuallyBoldedDates](#) property contains dates that appear in bold every year. The [MonthlyBoldedDates](#) property contains dates that appear in bold every month. Each of these properties contains an array of [DateTime](#) objects. To add or remove a date from one of these lists, you must add or remove a [DateTime](#) object.

To make a date appear in bold type

1. Create the [DateTime](#) objects.

```
Dim myVacation1 As Date = New DateTime(2001, 6, 10)
Dim myVacation2 As Date = New DateTime(2001, 6, 17)
```

```
DateTime myVacation1 = new DateTime(2001, 6, 10);
DateTime myVacation2 = new DateTime(2001, 6, 17);
```

```
DateTime myVacation1 = DateTime(2001, 6, 10);
DateTime myVacation2 = DateTime(2001, 6, 17);
```

2. Make a single date bold by calling the [AddBoldedDate](#), [AddAnnuallyBoldedDate](#), or [AddMonthlyBoldedDate](#) method of the [MonthCalendar](#) control.

```
MonthCalendar1.AddBoldedDate(myVacation1)
MonthCalendar1.AddBoldedDate(myVacation2)
```

```
monthCalendar1.AddBoldedDate(myVacation1);
monthCalendar1.AddBoldedDate(myVacation2);
```

```
monthCalendar1->AddBoldedDate(myVacation1);
monthCalendar1->AddBoldedDate(myVacation2);
```

—or—

Make a set of dates bold all at once by creating an array of [DateTime](#) objects and assigning it to one of the properties.

```
Dim VacationDates As DateTime() = {myVacation1, myVacation2}
MonthCalendar1.BoldedDates = VacationDates
```

```
DateTime[] VacationDates = {myVacation1, myVacation2};  
monthCalendar1.BoldedDates = VacationDates;
```

```
Array<DateTime>^ VacationDates = {myVacation1, myVacation2};  
monthCalendar1->BoldedDates = VacationDates;
```

To make a date appear in the regular font

1. Make a single bolded date appear in the regular font by calling the [RemoveBoldedDate](#), [RemoveAnnuallyBoldedDate](#), or [RemoveMonthlyBoldedDate](#) method.

```
MonthCalendar1.RemoveBoldedDate(myVacation1)  
MonthCalendar1.RemoveBoldedDate(myVacation2)
```

```
monthCalendar1.RemoveBoldedDate(myVacation1);  
monthCalendar1.RemoveBoldedDate(myVacation2);
```

```
monthCalendar1->RemoveBoldedDate(myVacation1);  
monthCalendar1->RemoveBoldedDate(myVacation2);
```

—or—

Remove all the bolded dates from one of the three lists by calling the [RemoveAllBoldedDates](#), [RemoveAllAnnuallyBoldedDates](#), or [RemoveAllMonthlyBoldedDates](#) method.

```
MonthCalendar1.RemoveAllBoldedDates()
```

```
monthCalendar1.RemoveAllBoldedDates();
```

```
monthCalendar1->RemoveAllBoldedDates();
```

2. Update the appearance of the font by calling the [UpdateBoldedDates](#) method.

```
MonthCalendar1.UpdateBoldedDates()
```

```
monthCalendar1.UpdateBoldedDates();
```

```
monthCalendar1->UpdateBoldedDates();
```

See Also

[MonthCalendar Control](#)

[How to: Select a Range of Dates in the Windows Forms MonthCalendar Control](#)

[How to: Change the Windows Forms MonthCalendar Control's Appearance](#)

[How to: Display More than One Month in the Windows Forms MonthCalendar Control](#)

How to: Select a Range of Dates in the Windows Forms MonthCalendar Control

5/4/2018 • 1 min to read • [Edit Online](#)

An important feature of the Windows Forms [MonthCalendar](#) control is that the user can select a range of dates. This feature is an improvement over the date-selection feature of the [DateTimePicker](#) control, which only enables the user to select a single date/time value. You can set a range of dates or get a selection range set by the user by using properties of the [MonthCalendar](#) control. The following code example demonstrates how to set a selection range.

To select a range of dates

1. Create [DateTime](#) objects that represent the first and last dates in a range.

```
Dim projectStart As Date = New DateTime(2001, 2, 13)
Dim projectEnd As Date = New DateTime(2001, 2, 28)
```

```
DateTime projectStart = new DateTime(2001, 2, 13);
DateTime projectEnd = new DateTime(2001, 2, 28);
```

```
DateTime projectStart = DateTime(2001, 2, 13);
DateTime projectEnd = DateTime(2001, 2, 28);
```

2. Set the [SelectionRange](#) property.

```
MonthCalendar1.SelectionRange = New SelectionRange(projectStart, projectEnd)
```

```
monthCalendar1.SelectionRange = new SelectionRange(projectStart, projectEnd);
```

```
monthCalendar1->SelectionRange = gcnew
    SelectionRange(projectStart, projectEnd);
```

—or—

Set the [SelectionStart](#) and [SelectionEnd](#) properties.

```
MonthCalendar1.SelectionStart = projectStart
MonthCalendar1.SelectionEnd = projectEnd
```

```
monthCalendar1.SelectionStart = projectStart;
monthCalendar1.SelectionEnd = projectEnd;
```

```
monthCalendar1->SelectionStart = projectStart;
monthCalendar1->SelectionEnd = projectEnd;
```

See Also

[MonthCalendar Control](#)

[How to: Change the Windows Forms MonthCalendar Control's Appearance](#)

[How to: Display Specific Days in Bold with the Windows Forms MonthCalendar Control](#)

[How to: Display More than One Month in the Windows Forms MonthCalendar Control](#)

NotifyIcon Component (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `NotifyIcon` component displays icons in the status notification area of the taskbar for processes that run in the background and would not otherwise have user interfaces. An example would be a virus protection program that can be accessed by clicking an icon in the status notification area of the taskbar.

In This Section

[NotifyIcon Component Overview](#)

Introduces the general concepts of the `NotifyIcon` component, which allows users to see icons for processes running in the background that do not have a user interface.

[How to: Add Application Icons to the TaskBar with the Windows Forms NotifyIcon Component](#)

Provides steps for setting the icon displayed by the `NotifyIcon` component.

[How to: Associate a Shortcut Menu with a Windows Forms NotifyIcon Component](#)

Provides steps for adding a shortcut menu to a `NotifyIcon` component.

Reference

[NotifyIcon](#)

Provides reference information on the class and its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

NotifyIcon Component Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [NotifyIcon](#) component is typically used to display icons for processes that run in the background and do not show a user interface much of the time. An example would be a virus protection program that can be accessed by clicking an icon in the status notification area of the taskbar.

Key Properties of NotifyIcons

Each [NotifyIcon](#) component displays a single icon in the status area. If you have three background processes and wish to display an icon for each, you must add three [NotifyIcon](#) components to the form. The key properties of the [NotifyIcon](#) component are [Icon](#) and [Visible](#). The [Icon](#) property sets the icon that appears in the status area. In order for the icon to appear, the [Visible](#) property must be set to `true`.

If you are using Visual Studio 2005, you have access to a large library of standard images that you can use with the [NotifyIcon](#) control.

NotifyIcons Options

You can associate balloon tips, shortcut menus, and ToolTips with a [NotifyIcon](#) to assist the user.

You can display balloon tips for a [NotifyIcon](#) by calling the [ShowBalloonTip](#) method specifying the time span you wish the balloon tip to display. You can also specify the text, icon and title of the balloon tip with the [BalloonTipText](#), [BalloonTipIcon](#) and [BalloonTipTitle](#), respectively. [NotifyIcon](#) components can also have associated ToolTips and shortcut menus. For more information, see [ToolTip Component Overview](#) and [ContextMenu Component Overview](#).

See Also

[NotifyIcon](#)

[NotifyIcon Component](#)

How to: Add Application Icons to the TaskBar with the Windows Forms NotifyIcon Component

5/4/2018 • 2 min to read • [Edit Online](#)

The Windows Forms [NotifyIcon](#) component displays a single icon in the status notification area of the taskbar. To display multiple icons in the status area, you must have multiple [NotifyIcon](#) components on your form. To set the icon displayed for a control, use the [Icon](#) property. You can also write code in the [DoubleClick](#) event handler so that something happens when the user double-clicks the icon. For example, you could make a dialog box appear for the user to configure the background process represented by the icon.

NOTE

The [NotifyIcon](#) component is used for notification purposes only, to alert users that an action or event has occurred or there has been a change in status of some sort. You should use menus, toolbars, and other user-interface elements for standard interaction with applications.

To set the icon

1. Assign a value to the [Icon](#) property. The value must be of type [System.Drawing.Icon](#) and can be loaded from an .ico file. You can specify the icon file in code or by clicking the ellipsis button (...) next to the [Icon](#) property in the **Properties** window, and then selecting the file in the **Open** dialog box that appears.
2. Set the [Visible](#) property to `true`.
3. Set the [Text](#) property to an appropriate ToolTip string.

In the following code example, the path set for the location of the icon is the **My Documents** folder. This location is used because you can assume that most computers running the Windows operating system will include this folder. Choosing this location also enables users with minimal system access levels to safely run the application. The following example requires a form with a [NotifyIcon](#) control already added. It also requires an icon file named `Icon.ico`.

```
' You should replace the bold icon in the sample below
' with an icon of your own choosing.
NotifyIcon1.Icon = New _
    System.Drawing.Icon(System.Environment.GetFolderPath _
    (System.Environment.SpecialFolder.Personal) _
    & "\Icon.ico")
NotifyIcon1.Visible = True
NotifyIcon1.Text = "Antivirus program"
```

```
// You should replace the bold icon in the sample below
// with an icon of your own choosing.
// Note the escape character used (@) when specifying the path.
notifyIcon1.Icon =
    new System.Drawing.Icon (System.Environment.GetFolderPath
    (System.Environment.SpecialFolder.Personal)
    + @"\Icon.ico");
notifyIcon1.Visible = true;
notifyIcon1.Text = "Antivirus program";
```

```
// You should replace the bold icon in the sample below
// with an icon of your own choosing.
notifyIcon1->Icon = gcnew
    System::Drawing::Icon(String::Concat
    (System::Environment::GetFolderPath
    (System::Environment::SpecialFolder::Personal),
    "\Icon.ico"));
notifyIcon1->Visible = true;
notifyIcon1->Text = "Antivirus program";
```

See Also

[NotifyIcon](#)

[Icon](#)

[How to: Associate a Shortcut Menu with a Windows Forms NotifyIcon Component](#)

[NotifyIcon Component](#)

[NotifyIcon Component Overview](#)

How to: Associate a Shortcut Menu with a Windows Forms NotifyIcon Component

5/4/2018 • 2 min to read • [Edit Online](#)

NOTE

Although [MenuStrip](#) and [ContextMenuStrip](#) replace and add functionality to the [MainMenu](#) and [ContextMenu](#) controls of previous versions, [MainMenu](#) and [ContextMenu](#) are retained for both backward compatibility and future use if you choose.

The [NotifyIcon](#) component displays an icon in the status notification area of the taskbar. Commonly, applications enable you to right-click this icon to send commands to the application it represents. By associating a [ContextMenu](#) component with the [NotifyIcon](#) component, you can add this functionality to your applications.

NOTE

If you want your application to be minimized at startup while displaying an instance of the [NotifyIcon](#) component in the taskbar, set the main form's [WindowState](#) property to [Minimized](#) and be sure the [NotifyIcon](#) component's [Visible](#) property is set to `true`.

To associate a shortcut menu with the NotifyIcon component at design time

1. Add a [NotifyIcon](#) component to your form, and set the important properties, such as the [Icon](#) and [Visible](#) properties.

For more information, see [How to: Add Application Icons to the TaskBar with the Windows Forms NotifyIcon Component](#).

2. Add a [ContextMenu](#) component to your Windows Form.

Add menu items to the shortcut menu representing the commands you want to make available at run time. This is also a good time to add menu enhancements to these menu items, such as access keys.

3. Set the [ContextMenu](#) property of the [NotifyIcon](#) component to the shortcut menu that you added.

With this property set, the shortcut menu will be displayed when the icon on the taskbar is clicked.

To associate a shortcut menu with the NotifyIcon component programmatically

1. Create an instance of the [NotifyIcon](#) class and a [ContextMenu](#) class, with whatever property settings are necessary for the application ([Icon](#) and [Visible](#) properties for the [NotifyIcon](#) component, menu items for the [ContextMenu](#) component).
2. Set the [ContextMenu](#) property of the [NotifyIcon](#) component to the shortcut menu that you added.

With this property set, the shortcut menu will be displayed when the icon on the taskbar is clicked.

NOTE

The following code example creates a basic menu structure. You will need to customize the menu choices to those that fit the application you are developing. Additionally, you will want to write code to handle the [Click](#) events for these menu items.

```

Public ContextMenu1 As New ContextMenu
Public NotifyIcon1 As New NotifyIcon

Public Sub CreateIconMenuStructure()
    ' Add menu items to shortcut menu.
    ContextMenu1.MenuItems.Add("&Open Application")
    ContextMenu1.MenuItems.Add("S&uspend Application")
    ContextMenu1.MenuItems.Add("E&xit")

    ' Set properties of NotifyIcon component.
    NotifyIcon1.Icon = New System.Drawing.Icon _
        (System.Environment.GetFolderPath _
        (System.Environment.SpecialFolder.Personal) _
        & "\Icon.ico")
    NotifyIcon1.Text = "Right-click me!"
    NotifyIcon1.Visible = True
    NotifyIcon1.ContextMenu = ContextMenu1
End Sub

```

```

public NotifyIcon notifyIcon1 = new NotifyIcon();
public ContextMenu contextMenu1 = new ContextMenu();

public void createIconMenuStructure()
{
    // Add menu items to shortcut menu.
    contextMenu1.MenuItems.Add("&Open Application");
    contextMenu1.MenuItems.Add("S&uspend Application");
    contextMenu1.MenuItems.Add("E&xit");

    // Set properties of NotifyIcon component.
    notifyIcon1.Icon = new System.Drawing.Icon
        (System.Environment.GetFolderPath
        (System.Environment.SpecialFolder.Personal)
        + @"\Icon.ico");
    notifyIcon1.Visible = true;
    notifyIcon1.Text = "Right-click me!";
    notifyIcon1.Visible = true;
    notifyIcon1.ContextMenu = contextMenu1;
}

```

```

public:
    System::Windows::Forms::NotifyIcon ^ notifyIcon1;
    System::Windows::Forms::ContextMenu ^ contextMenu1;

    void createIconMenuStructure()
    {
        // Add menu items to shortcut menu.
        contextMenu1->MenuItems->Add("&Open Application");
        contextMenu1->MenuItems->Add("S&uspend Application");
        contextMenu1->MenuItems->Add("E&xit");

        // Set properties of NotifyIcon component.
        notifyIcon1->Icon = gcnew System::Drawing::Icon
            (String::Concat(System::Environment::GetFolderPath
            (System::Environment::SpecialFolder::Personal),
            "\Icon.ico"));
        notifyIcon1->Text = "Right-click me!";
        notifyIcon1->Visible = true;
        notifyIcon1->ContextMenu = contextMenu1;
    }
}

```

NOTE

You must initialize `notifyIcon1` and `contextMenu1`, which you can do by including the following statements in the constructor of your form:

```
notifyIcon1 = gcnew System::Windows::Forms::NotifyIcon();
contextMenu1 = gcnew System::Windows::Forms::ContextMenu();
```

See Also

[NotifyIcon](#)

[Icon](#)

[How to: Add Application Icons to the TaskBar with the Windows Forms NotifyIcon Component](#)

[NotifyIcon Component](#)

[NotifyIcon Component Overview](#)

NumericUpDown Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `NumericUpDown` control looks like a combination of a text box and a pair of arrows that the user can click to adjust a value. The control displays and sets a single numeric value from a list of choices. The user can increase and decrease the number by clicking up and down buttons, by pressing the UP and DOWN ARROW keys, or by typing a number. Clicking the UP ARROW key moves the value toward its maximum; clicking the DOWN ARROW key moves the position toward the minimum. An example where this kind of control might be useful is for a volume control on a music player. Numeric up-down controls are used in some Windows control panel applications.

In This Section

[NumericUpDown Control Overview](#)

Introduces the general concepts of the `NumericUpDown` control, which allows users to browse through and select from a list of numeric values.

[How to: Set and Return Numeric Values with the Windows Forms NumericUpDown Control](#)

Describes how to test for the value of the control.

[How to: Set the Format for the Windows Forms NumericUpDown Control](#)

Describes how to configure how values are displayed in the control.

Reference

[NumericUpDown](#)

Provides reference information on the `NumericUpDown` class and its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

[DomainUpDown Control](#)

Introduces a control similar to `NumericUpDown`, except that the `DomainUpDown` control displays string instead of numeric values.

NumericUpDown Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The [NumericUpDown](#) control looks like a combination of a text box and a pair of arrows that the user can click to adjust a value. The control displays and sets a single numeric value from a list of fixed numeric-value choices. The user can increase and decrease the number by clicking the up and down arrows, by pressing the UP and DOWN ARROW keys, or by typing a number in the text box part of the control. Clicking the UP ARROW key moves the number toward the maximum; clicking the DOWN ARROW key moves the number toward the minimum.

Because of its versatile functionality, this control is an obvious choice, for example, if you want to create a volume control for a music player application. The [NumericUpDown](#) control is used in many Windows Control Panel applications.

Key Properties and Methods

The numbers displayed in the control's text box can be in a variety of formats, including hexadecimal. For more information, see [How to: Set the Format for the Windows Forms NumericUpDown Control](#). The key properties of the control are [Value](#), [Maximum](#) (default value 100), [Minimum](#) (default value 0), and [Increment](#) (default value 1). The [Value](#) property sets the current number selected in the control. The [Increment](#) property sets the amount that the number is adjusted by when the user clicks an up or down arrow. When focus moves off the control, any typed input will be validated against the minimum and maximum numeric values. You can increase the speed that the control moves through numbers, when the user continuously presses the up or down arrow, with the [Accelerations](#) property. The key methods of the control are [UpButton](#) and [DownButton](#).

See Also

[NumericUpDown](#)

[NumericUpDown Control](#)

[How to: Set the Format for the Windows Forms NumericUpDown Control](#)

[TextBox Control](#)

How to: Set and Return Numeric Values with the Windows Forms NumericUpDown Control

5/4/2018 • 1 min to read • [Edit Online](#)

The numeric value of the Windows Forms [NumericUpDown](#) control is determined by its [Value](#) property. You can write conditional tests for the control's value just as with any other property. Once the [Value](#) property is set, you can adjust it directly by writing code to perform operations on it, or you can call the [UpButton](#) and [DownButton](#) methods.

To set the numeric value

1. Assign a value to the [Value](#) property in code or in the Properties window.

```
NumericUpDown1.Value = 55
```

```
numericUpDown1.Value = 55;
```

```
numericUpDown1->Value = 55;
```

-or-

2. Call the [UpButton](#) or [DownButton](#) method to increase or decrease the value by the amount specified in the [Increment](#) property.

```
NumericUpDown1.UpButton()
```

```
numericUpDown1.UpButton();
```

```
numericUpDown1->UpButton();
```

To return the numeric value

- Access the [Value](#) property in code.

```
If NumericUpDown1.Value >= 65 Then  
    MessageBox.Show("Age is: " & NumericUpDown1.Value.ToString)  
Else  
    MessageBox.Show("The customer is ineligible for a senior citizen discount.")  
End If
```

```
if(numericUpDown1.Value >= 65)
{
    MessageBox.Show("Age is: " + numericUpDown1.Value.ToString());
}
else
{
    MessageBox.Show("The customer is ineligible for a senior citizen discount.");
}
```

```
if(numericUpDown1->Value >= 65)
{
    MessageBox::Show(String::Concat("Age is: ",
        numericUpDown1->Value.ToString()));
}
else
{
    MessageBox::Show
    ("The customer is ineligible for a senior citizen discount.");
}
```

See Also

[NumericUpDown](#)

[NumericUpDown.Value](#)

[NumericUpDown.Increment](#)

[NumericUpDown.UpButton](#)

[NumericUpDown.DownButton](#)

[NumericUpDown Control](#)

[NumericUpDown Control Overview](#)

How to: Set the Format for the Windows Forms NumericUpDown Control

5/4/2018 • 1 min to read • [Edit Online](#)

You can configure how values are displayed in the Windows Forms [NumericUpDown](#) control. The [DecimalPlaces](#) property determines how many numbers appear after the decimal point; the default is 0. The [ThousandsSeparator](#) property determines whether a separator will be inserted between every three decimal digits; the default is `false`. The control can display values in hexadecimal instead of decimal format, if the [Hexadecimal](#) property is set to `true`; the default is `false`.

To format the numeric value

- Display a decimal value by setting the [DecimalPlaces](#) property to an integer and setting the [ThousandsSeparator](#) property to `true` or `false`.

```
NumericUpDown1.DecimalPlaces = 2  
NumericUpDown1.ThousandsSeparator = True
```

```
numericUpDown1.DecimalPlaces = 2;  
numericUpDown1.ThousandsSeparator = true;
```

```
numericUpDown1->DecimalPlaces = 2;  
numericUpDown1->ThousandsSeparator = true;
```

-or-

- Display a hexadecimal value by setting the [Hexadecimal](#) property to `true`.

```
NumericUpDown1.Hexadecimal = True
```

```
numericUpDown1.Hexadecimal = true;
```

```
numericUpDown1->Hexadecimal = true;
```

NOTE

Even if the value is displayed on the form as hexadecimal, any tests you perform on the [Value](#) property will be testing its decimal value.

See Also

[NumericUpDown](#)

[NumericUpDown Control](#)

[NumericUpDown Control Overview](#)

OpenFileDialog Component (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [OpenFileDialog](#) component is a pre-configured dialog box. It is the same **Open File** dialog box exposed by the Windows operating system. It inherits from the [CommonDialog](#) class.

In This Section

[OpenFileDialog Component Overview](#)

Introduces the general concepts of the [OpenFileDialog](#) component, which allows you to display a pre-configured dialog box that users can use to open files.

[How to: Open Files Using the OpenFileDialog Component](#)

Explains how to open a file at run time via an instance of the [OpenFileDialog](#) component.

Reference

[OpenFileDialog](#)

Provides reference information on the [OpenFileDialog](#) class and its members.

Related Sections

[Dialog-Box Controls and Components](#)

Describes a set of controls and components that allow users to perform standard interactions with the application or system.

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

[Essential Code for Windows Forms Dialog Boxes](#)

Discusses the Windows Forms dialog box controls and components and the code necessary for executing their basic functions. (MSDN Online Library technical article)

OpenFileDialog Component Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [OpenFileDialog](#) component is a pre-configured dialog box. It is the same **Open File** dialog box exposed by the Windows operating system. It inherits from the [CommonDialog](#) class.

Using this Component

Use this component within your Windows-based application as a simple solution for file selection in lieu of configuring your own dialog box. By relying on standard Windows dialog boxes, you create applications whose basic functionality is immediately familiar to users. Be aware, however, that when using the [OpenFileDialog](#) component, you must write your own file-opening logic.

Use the [ShowDialog](#) method to display the dialog at run time. You can enable users to multi-select files to be opened with the [Multiselect](#) property. Additionally, you can use the [ShowReadOnly](#) property to determine if a read-only check box appears in the dialog box. The [ReadOnlyChecked](#) property indicates whether the read-only check box is selected. Finally, the [Filter](#) property sets the current file name filter string, which determines the choices that appear in the "Files of type" box in the dialog box.

When it is added to a form, the [OpenFileDialog](#) component appears in the tray at the bottom of the Windows Forms Designer.

See Also

[OpenFileDialog](#)
 [OpenFileDialog Component](#)

How to: Open Files Using the OpenFileDialog Component

5/4/2018 • 4 min to read • [Edit Online](#)

The [OpenFileDialog](#) component allows users to browse the folders of their computer or any computer on the network and select one or more files to open. The dialog box returns the path and name of the file the user selected in the dialog box.

Once the user has selected the file to be opened, there are two approaches to the mechanism of opening the file. If you prefer to work with file streams, you can create an instance of the [StreamReader](#) class. Alternately, you can use the [OpenFile](#) method to open the selected file.

The first example below involves a [FileIOPermission](#) permission check (as described in the "Security Note" below), but gives you access to the filename. You can use this technique from the Local Machine, Intranet, and Internet zones. The second method also does a [FileIOPermission](#) permission check, but is better suited for applications in the Intranet or Internet zones.

To open a file as a stream using the OpenFileDialog component

1. Display the **Open File** dialog box and call a method to open the file selected by the user.

One approach is to use the [ShowDialog](#) method to display the Open File dialog box, and use an instance of the [StreamReader](#) class to open the file.

The example below uses the [Button](#) control's [Click](#) event handler to open an instance of the [OpenFileDialog](#) component. When a file is chosen and the user clicks **OK**, the file selected in the dialog box opens. In this case, the contents are displayed in a message box, just to show that the file stream has been read.

IMPORTANT

To get or set the [FileName](#) property, your assembly requires a privilege level granted by the [System.Security.Permissions.FileIOPermission](#) class. If you are running in a partial-trust context, the process might throw an exception due to insufficient privileges. For more information, see [Code Access Security Basics](#).

The example assumes your form has a [Button](#) control and an [OpenFileDialog](#) component.

```
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click
    If OpenFileDialog1.ShowDialog() = System.Windows.Forms.DialogResult.OK Then
        Dim sr As New System.IO.StreamReader(OpenFileDialog1.FileName)
        MessageBox.Show(sr.ReadToEnd)
        sr.Close()
    End If
End Sub
```

```
private void button1_Click(object sender, System.EventArgs e)
{
    if(openFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK)
    {
        System.IO.StreamReader sr = new
            System.IO.StreamReader(openFileDialog1.FileName);
        MessageBox.Show(sr.ReadToEnd());
        sr.Close();
    }
}
```

```
private:
void button1_Click(System::Object ^ sender,
System::EventArgs ^ e)
{
    if(openFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK)
    {
        System::IO::StreamReader ^ sr = gcnew
            System::IO::StreamReader(openFileDialog1->FileName);
        MessageBox::Show(sr->ReadToEnd());
        sr->Close();
    }
}
```

(Visual C# and Visual C++) Place the following code in the form's constructor to register the event handler.

```
this.button1.Click += new System.EventHandler(this.button1_Click);
```

```
this->button1->Click += gcnew
    System::EventHandler(this, &Form1::button1_Click);
```

NOTE

For more information about reading from file streams, see [BeginRead](#) and [Read](#).

To open a file as a file using the OpenFileDialog component

1. Use the [ShowDialog](#) method to display the dialog box and the [OpenFile](#) method to open the file.

The [OpenFileDialog](#) component's [OpenFile](#) method returns the bytes that compose the file. These bytes give you a stream to read from. In the example below, an [OpenFileDialog](#) component is instantiated with a "cursor" filter on it, allowing the user to choose only files with the file name extension `.cur`. If a `.cur` file is chosen, the form's cursor is set to the selected cursor.

IMPORTANT

To call the [OpenFile](#) method, your assembly requires a privilege level granted by the [System.Security.Permissions.FileIOPermission](#) class. If you are running in a partial-trust context, the process might throw an exception due to insufficient privileges. For more information, see [Code Access Security Basics](#).

The example assumes your form has a [Button](#) control.

```

Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click
    ' Displays an OpenFileDialog so the user can select a Cursor.
    Dim openFileDialog1 As New OpenFileDialog()
    openFileDialog1.Filter = "Cursor Files|*.cur"
    openFileDialog1.Title = "Select a Cursor File"

    ' Show the Dialog.
    ' If the user clicked OK in the dialog and
    ' a .CUR file was selected, open it.
    If openFileDialog1.ShowDialog() = System.Windows.Forms.DialogResult.OK Then
        ' Assign the cursor in the Stream to the Form's Cursor property.
        Me.Cursor = New Cursor(openFileDialog1.OpenFile())
    End If
End Sub

```

```

private void button1_Click(object sender, System.EventArgs e)
{
    // Displays an OpenFileDialog so the user can select a Cursor.
    OpenFileDialog openFileDialog1 = new OpenFileDialog();
    openFileDialog1.Filter = "Cursor Files|*.cur";
    openFileDialog1.Title = "Select a Cursor File";

    // Show the Dialog.
    // If the user clicked OK in the dialog and
    // a .CUR file was selected, open it.
    if (openFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK)
    {
        // Assign the cursor in the Stream to the Form's Cursor property.
        this.Cursor = new Cursor(openFileDialog1.OpenFile());
    }
}

```

```

private:
void button1_Click(System::Object ^ sender,
    System::EventArgs ^ e)
{
    // Displays an OpenFileDialog so the user can select a Cursor.
    OpenFileDialog ^ openFileDialog1 = new OpenFileDialog();
    openFileDialog1->Filter = "Cursor Files|*.cur";
    openFileDialog1->Title = "Select a Cursor File";

    // Show the Dialog.
    // If the user clicked OK in the dialog and
    // a .CUR file was selected, open it.
    if (openFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK)
    {
        // Assign the cursor in the Stream to
        // the Form's Cursor property.
        this->Cursor = gcnew
            System::Windows::Forms::Cursor(
                openFileDialog1->OpenFile());
    }
}

```

(Visual C# and Visual C++) Place the following code in the form's constructor to register the event handler.

```

this.button1.Click += new System.EventHandler(this.button1_Click);

```

```
this->button1->Click += gcnew  
System::EventHandler(this, &Form1::button1_Click);
```

See Also

[OpenFileDialog](#)
 [OpenFileDialog Component](#)

PageSetupDialog Component (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [PageSetupDialog](#) component is a pre-configured dialog box used to set page details for printing in Windows-based applications. Use it within your Windows-based application as a simple solution for users to set page preferences in lieu of configuring your own dialog box. You can enable users to set border and margin adjustments, headers and footers, and portrait vs. landscape orientation. By relying on standard Windows dialog boxes, you create applications whose basic functionality is immediately familiar to users.

In This Section

[PageSetupDialog Component Overview](#)

Introduces the general concepts of the [PageSetupDialog](#) component, which you can use to display a pre-configured dialog box that users can use to manipulate page settings.

[How to: Determine Page Properties Using the PageSetupDialog Component](#)

Explains how to set page properties by using an instance of the [PageSetupDialog](#) component at run time.

Reference

[PageSetupDialog](#)

Provides reference information on the class and its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

[Dialog-Box Controls and Components](#)

Describes a set of controls and components that allow users to perform standard interactions with the application or system.

[Essential Code for Windows Forms Dialog Boxes](#)

Discusses the Windows Forms dialog box controls and components and the code necessary for executing their basic functions. (MSDN Online Library technical article)

PageSetupDialog Component Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [PageSetupDialog](#) component is a pre-configured dialog box used to set page details for printing in Windows-based applications. Use it within your Windows-based application as a simple solution for users to set page preferences in lieu of configuring your own dialog box. You can enable users to set border and margin adjustments, headers and footers, and portrait or landscape orientation. By relying on standard Windows dialog boxes, you create applications whose basic functionality is immediately familiar to users.

Key Properties and Methods

Use the [ShowDialog](#) method to display the dialog at run time. This component has properties you can set that relate to either a single page ([PrintDocument](#) class) or any document ([PageSettings](#) class). Additionally, the [PageSetupDialog](#) component can be used to determine specific printer settings, which are stored in the [PrinterSettings](#) class.

When it is added to a form, the [PageSetupDialog](#) component appears in the tray at the bottom of the Windows Forms Designer.

See Also

[PageSetupDialog](#)
[PageSetupDialog Component](#)

How to: Determine Page Properties Using the PageSetupDialog Component

5/4/2018 • 2 min to read • [Edit Online](#)

The [PageSetupDialog](#) component presents layout, paper size, and other page layout choices to the user for a document.

You need to specify an instance of the [PrintDocument](#) class—this is the document to be printed. Additionally, users must have a printer installed on their computer, either locally or through a network, as this is partly how the [PageSetupDialog](#) component determines the page formatting choices presented to the user.

An important aspect of working with the [PageSetupDialog](#) component is how it interacts with the [PageSettings](#) class. The [PageSettings](#) class is used to specify settings that modify the way a page will be printed, such as paper orientation, the size of the page, and the margins. Each of these settings is represented as a property of the [PageSettings](#) class. The [PageSetupDialog](#) class modifies these property values for a given instance of the [PageSettings](#) class that is associated with the document (and is represented as a [DefaultPageSettings](#) property).

To set page properties using the PageSetupDialog component

1. Use the [ShowDialog](#) method to display the dialog box, specifying the [PrintDocument](#) to use.

In the example below, the [Button](#) control's [Click](#) event handler opens an instance of the [PageSetupDialog](#) component. An existing document is specified in the [Document](#) property, and its [PageSettings.Color](#) property is set to `false`.

The example assumes your form has a [Button](#) control, a [PrintDocument](#) component named `myDocument`, and a [PageSetupDialog](#) component.

```
Private Sub Button1_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles Button1.Click  
    ' The print document 'myDocument' used below  
    ' is merely for an example.  
    'You will have to specify your own print document.  
    PageSetupDialog1.Document = myDocument  
    ' Sets the print document's color setting to false,  
    ' so that the page will not be printed in color.  
    PageSetupDialog1.Document.DefaultPageSettings.Color = False  
    PageSetupDialog1.ShowDialog()  
End Sub
```

```
private void button1_Click(object sender, System.EventArgs e)  
{  
    // The print document 'myDocument' used below  
    // is merely for an example.  
    // You will have to specify your own print document.  
    pageSetupDialog1.Document = myDocument;  
    // Sets the print document's color setting to false,  
    // so that the page will not be printed in color.  
    pageSetupDialog1.Document.DefaultPageSettings.Color = false;  
    pageSetupDialog1.ShowDialog();  
}
```

```
private:  
    System::Void button1_Click(System::Object ^  sender,  
                               System::EventArgs ^  e)  
{  
    // The print document 'myDocument' used below  
    // is merely for an example.  
    // You will have to specify your own print document.  
    pageSetupDialog1->Document = myDocument;  
    // Sets the print document's color setting to false,  
    // so that the page will not be printed in color.  
    pageSetupDialog1->Document->DefaultPageSettings->Color = false;  
    pageSetupDialog1->ShowDialog();  
}
```

(Visual C# and Visual C++) Place the following code in the form's constructor to register the event handler.

```
this.button1.Click += new System.EventHandler(this.button1_Click);
```

```
this->button1->Click += gcnew  
    System::EventHandler(this, &Form1::button1_Click);
```

See Also

[PageSetupDialog](#)

[How to: Create Standard Windows Forms Print Jobs](#)

[PageSetupDialog Component](#)

Panel Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

Windows Forms `Panel` controls are used to provide an identifiable grouping for other controls. Typically, you use panels to subdivide a form by function. The `Panel` control is similar to the `GroupBox` control; however, only the `Panel` control can have scroll bars, and only the `GroupBox` control displays a caption.

In This Section

[Panel Control Overview](#)

Explains what this control is and its key features and properties.

[How to: Group Controls with the Windows Forms Panel Control Using the Designer](#)

Describes how to group controls with a panel using the designer.

[How to: Set the Background of a Windows Forms Panel Using the Designer](#)

Describes how to display a background color and a background image on a panel using the designer.

[How to: Set the Background of a Panel](#)

Describes how to display a background color and a background image on a panel.

Reference

[Panel](#)

Describes this class and has links to all its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

[How to: Add to or Remove from a Collection of Controls at Run Time](#)

Describes how to add controls to and remove controls from any container control on your forms.

Panel Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

Windows Forms [Panel](#) controls are used to provide an identifiable grouping for other controls. Typically, you use panels to subdivide a form by function. For example, you may have an order form that specifies mailing options such as which overnight carrier to use. Grouping all options in a panel gives the user a logical visual cue. At design time all the controls can be moved easily — when you move the [Panel](#) control, all its contained controls move, too. The controls grouped in a panel can be accessed through its [Controls](#) property. This property returns a collection of [Control](#) instances, so you will typically need to cast a control retrieved this way to its specific type.

Panel Versus GroupBox

The [Panel](#) control is similar to the [GroupBox](#) control; however, only the [Panel](#) control can have scroll bars, and only the [GroupBox](#) control displays a caption.

Key Properties

To display scroll bars, set the [AutoScroll](#) property to `true`. You can also customize the appearance of the panel by setting the [BackColor](#), [BackgroundImage](#), and [BorderStyle](#) properties. For more information on the [BackColor](#) and [BackgroundImage](#) properties, see [How to: Set the Background of a Panel](#). The [BorderStyle](#) property determines if the panel is outlined with no visible border ([None](#)), a plain line ([FixedSingle](#)), or a shadowed line ([Fixed3D](#)).

See Also

[Panel](#)

[GroupBox Control](#)

[How to: Group Controls with the Windows Forms Panel Control Using the Designer](#)

[How to: Set the Background of a Windows Forms Panel Using the Designer](#)

How to: Group Controls with the Windows Forms Panel Control Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

Windows Forms [Panel](#) controls are used to group other controls. There are three reasons to group controls. One is visual grouping of related form elements for a clear user interface; another is programmatic grouping, of radio buttons for example; the last is for moving the controls as a unit at design time.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To create a group of controls

1. Drag a [Panel](#) control from the **Windows Forms** tab of the Toolbox onto a form.
2. Add other controls to the panel, drawing each inside the panel.

If you have existing controls that you want to enclose in a panel, you can select all the controls, cut them to the Clipboard, select the [Panel](#) control, and then paste them into the panel. You can also drag them into the panel.

3. (Optional) If you want to add a border to a panel, set its [BorderStyle](#) property. There are three choices: [Fixed3D](#), [FixedSingle](#), and [None](#).

See Also

[Panel Control](#)

[Panel Control Overview](#)

[How to: Set the Background of a Panel](#)

How to: Set the Background of a Windows Forms Panel

5/4/2018 • 1 min to read • [Edit Online](#)

A Windows Forms [Panel](#) control can display both a background color and a background image. The [BackColor](#) property sets the background color for the contained controls, such as labels and radio buttons. If the [BackgroundImage](#) property is not set, the [BackColor](#) selection will fill the entire panel. If the [BackgroundImage](#) property is set, the image will be displayed behind the contained controls.

To set the background programmatically

1. Set the panel's [BackColor](#) property to a value of type [System.Drawing.Color](#).

```
Panel1.BackColor = Color.AliceBlue
```

```
panel1.BackColor = Color.AliceBlue;
```

```
panel1->BackColor = Color::AliceBlue;
```

2. Set the panel's [BackgroundImage](#) property using the [FromFile](#) method of the [System.Drawing.Image](#) class.

```
' You should replace the bolded image
' in the sample below with an image of your own choosing.
Panel1.BackgroundImage = Image.FromFile _
    (System.Environment.GetFolderPath _
    (System.Environment.SpecialFolder.Personal) _
    & "\Image.gif")
```

```
// You should replace the bolded image
// in the sample below with an image of your own choosing.
// Note the escape character used (@) when specifying the path.
panel1.BackgroundImage = Image.FromFile
    (System.Environment.GetFolderPath
    (System.Environment.SpecialFolder.Personal)
    + @"\Image.gif");
```

```
// You should replace the bolded image
// in the sample below with an image of your own choosing.
panel1->BackgroundImage = Image::FromFile(String::Concat(
    System::Environment::GetFolderPath
    (System::Environment::SpecialFolder::Personal),
    "\\Image.gif"));
```

See Also

[BackColor](#)
[BackgroundImage](#)
[Panel Control](#)

Panel Control Overview

How to: Set the Background of a Windows Forms Panel Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

A Windows Forms [Panel](#) control can display both a background color and a background image. The [BackColor](#) property sets the background color for controls that are contained in the panel, such as labels and radio buttons. If the [BackgroundImage](#) property is not set, the [BackColor](#) selection will fill all of the panel. If the [BackgroundImage](#) property is set, the image will be displayed behind the controls that are contained in the panel.

The following procedure requires a **Windows Application** project with a form that contains a [Panel](#) control. For information about how to set up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To set the background in the Windows Forms Designer

1. Select the [Panel](#) control.
2. In the **Properties** window, click the arrow button next to the [BackColor](#) property to display a window with three tabs.
3. Select the **Custom** tab to display a palette of colors.
4. Select the **Web** or **System** tab to display a list of predefined names for colors, and then select a color.
5. In the **Properties** window, click the arrow button next to the [BackgroundImage](#) property.
6. In the **Open** dialog box, select the file that you want to display.

See Also

[BackColor](#)

[BackgroundImage](#)

[Panel Control](#)

[Panel Control Overview](#)

[How to: Group Controls with the Windows Forms Panel Control Using the Designer](#)

PictureBox Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `PictureBox` control is used to display graphics in bitmap, GIF, JPEG, metafile, or icon format.

In This Section

[PictureBox Control Overview](#)

Explains what this control is and its key features and properties.

[How to: Modify the Size or Placement of a Picture at Run Time](#)

Explains what the `SizeMode` property does and how to set it.

[How to: Set Pictures at Run Time](#)

Describes how to display and clear a picture at run time.

[How to: Load a Picture Using the Designer](#)

Describes how to load and display a picture on a form at design time.

Reference

[PictureBox](#)

Describes this class and has links to all its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

PictureBox Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [PictureBox](#) control is used to display graphics in bitmap, GIF, JPEG, metafile, or icon format.

Key Properties and Methods

The picture that is displayed is determined by the [Image](#) property, which can be set at run time or at design time. You can alternatively specify the image by setting the [ImageLocation](#) property and then load the image synchronously using the [Load](#) method or asynchronously using the [LoadAsync](#) method. The [SizeMode](#) property controls how the image and control fit with each other. For more information, see [How to: Modify the Size or Placement of a Picture at Run Time](#).

See Also

[PictureBox](#)

[How to: Load a Picture Using the Designer](#)

[How to: Modify the Size or Placement of a Picture at Run Time](#)

[How to: Set Pictures at Run Time](#)

[PictureBox Control](#)

How to: Load a Picture Using the Designer (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

With the Windows Forms [PictureBox](#) control, you can load and display a picture on a form at design time by setting the [Image](#) property to a valid picture. The following table shows the acceptable file types.

TYPE	FILE NAME EXTENSION
Bitmap	.bmp
Icon	.ico
GIF	.gif
Metafile	.wmf
JPEG	.jpg

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To display a picture at design time

1. Draw a [PictureBox](#) control on a form.
2. On the Properties window, select the [Image](#) property, then click the ellipsis button to display the **Open** dialog box.
3. If you are looking for a specific file type (for example, .gif files), select it in the **Files of type** box.
4. Select the file you want to display.

To clear the picture at design time

1. On the **Properties** window, select the [Image](#) property and right-click the small thumbnail image that appears to the left of the name of the image object. Choose **Reset**.

See Also

[PictureBox](#)

[PictureBox Control Overview](#)

[How to: Modify the Size or Placement of a Picture at Run Time](#)

[How to: Set Pictures at Run Time](#)

[PictureBox Control](#)

How to: Modify the Size or Placement of a Picture at Run Time (Windows Forms)

5/4/2018 • 2 min to read • [Edit Online](#)

If you use the Windows Forms [PictureBox](#) control on a form, you can set the [SizeMode](#) property on it to:

- Align the picture's upper left corner with the control's upper left corner
- Center the picture within the control
- Adjust the size of the control to fit the picture it displays
- Stretch any picture it displays to fit the control

Stretching a picture (especially one in bitmap format) can produce a loss in image quality. Metafiles, which are lists of graphics instructions for drawing images at run time, are better suited for stretching than bitmaps are.

To set the [SizeMode](#) property at run time

1. Set [SizeMode](#) to [Normal](#) (the default), [AutoSize](#), [CenterImage](#), or [StretchImage](#). [Normal](#) means that the image is placed in the control's upper-left corner; if the image is larger than the control, its lower and right edges are clipped. [CenterImage](#) means that the image is centered within the control; if the image is larger than the control, the picture's outside edges are clipped. [AutoSize](#) means that the size of the control is adjusted to the size of the image. [StretchImage](#) is the reverse, and means that the size of the image is adjusted to the size of the control.

In the example below, the path set for the location of the image is the My Documents folder. This is done, because you can assume that most computers running the Windows operating system will include this directory. This also allows users with minimal system access levels to safely run the application. The example below assumes a form with a [PictureBox](#) control already added.

```
Private Sub StretchPic()
    ' Stretch the picture to fit the control.
    PictureBox1.SizeMode = PictureBoxSizeMode.StretchImage
    ' Load the picture into the control.
    ' You should replace the bold image
    ' in the sample below with an icon of your own choosing.
    PictureBox1.Image = Image.FromFile _
        (System.Environment.GetFolderPath _
        (System.Environment.SpecialFolder.Personal) _
        & "\Image.gif")
End Sub
```

```
private void StretchPic(){
    // Stretch the picture to fit the control.
    PictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
    // Load the picture into the control.
    // You should replace the bold image
    // in the sample below with an icon of your own choosing.
    // Note the escape character used (@) when specifying the path.
    PictureBox1.Image = Image.FromFile _
        (System.Environment.GetFolderPath _
        (System.Environment.SpecialFolder.Personal) _
        + @"\Image.gif")
}
```

```
private:  
    void StretchPic()  
    {  
        // Stretch the picture to fit the control.  
        pictureBox1->SizeMode = PictureBoxSizeMode::StretchImage;  
        // Load the picture into the control.  
        // You should replace the bold image  
        // in the sample below with an icon of your own choosing.  
        pictureBox1->Image = Image::FromFile(String::Concat(  
            System::Environment::GetFolderPath(  
                System::Environment::SpecialFolder::Personal),  
            "\\Image.gif"));  
    }  
}
```

See Also

[PictureBox](#)

[How to: Load a Picture Using the Designer](#)

[PictureBox Control Overview](#)

[How to: Set Pictures at Run Time](#)

[PictureBox Control](#)

How to: Set Pictures at Run Time (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

You can programmatically set the image displayed by a Windows Forms [PictureBox](#) control.

To set a picture programmatically

- Set the [Image](#) property using the [FromFile](#) method of the [Image](#) class.

In the example below, the path set for the location of the image is the My Documents folder. This is done, because you can assume that most computers running the Windows operating system will include this directory. This also allows users with minimal system access levels to safely run the application. The example below assumes a form with a [PictureBox](#) control already added.

```
Private Sub LoadNewPict()
    ' You should replace the bold image
    ' in the sample below with an icon of your own choosing.
    PictureBox1.Image = Image.FromFile _
        (System.Environment.GetFolderPath _
            (System.Environment.SpecialFolder.Personal) _
            & "\Image.gif")
End Sub
```

```
private void LoadNewPict(){
    // You should replace the bold image
    // in the sample below with an icon of your own choosing.
    // Note the escape character used (@) when specifying the path.
    pictureBox1.Image = Image.FromFile
        (System.Environment.GetFolderPath
            (System.Environment.SpecialFolder.Personal)
            + @"\Image.gif");
}
```

```
private:
void LoadNewPict()
{
    // You should replace the bold image
    // in the sample below with an icon of your own choosing.
    pictureBox1->Image = Image::FromFile(String::Concat(
        System::Environment::GetFolderPath(
            System::Environment::SpecialFolder::Personal),
        "\\Image.gif"));
}
```

To clear a graphic

- First, release the memory being used by the image, and then clear the graphic. Garbage collection will free up the memory later if memory management becomes a problem.

```
If Not (PictureBox1.Image Is Nothing) Then
    PictureBox1.Image.Dispose()
    PictureBox1.Image = Nothing
End If
```

```
if (pictureBox1.Image != null)
{
    pictureBox1.Image.Dispose();
    pictureBox1.Image = null;
}
```

```
if (pictureBox1->Image != nullptr)
{
    pictureBox1->Image->Dispose();
    pictureBox1->Image = nullptr;
}
```

NOTE

For more information on why you should use the [Dispose](#) method in this way, see [Cleaning Up Unmanaged Resources](#).

This code will clear the image even if a graphic was loaded into the control at design time.

See Also

[PictureBox](#)

[Image.FromFile](#)

[PictureBox Control Overview](#)

[How to: Load a Picture Using the Designer](#)

[How to: Modify the Size or Placement of a Picture at Run Time](#)

[PictureBox Control](#)

PrintDialog Component (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `PrintDialog` component is a pre-configured dialog box used to select a printer, choose the pages to print, and determine other print-related settings in Windows-based applications. Use it as a simple solution for printer and print-related settings selection in lieu of configuring your own dialog box. You can enable users to print many parts of their documents: print all, print a specified page range, or print a selection. By relying on standard Windows dialog boxes, you create applications whose basic functionality is immediately familiar to users.

In This Section

[PrintDialog Component Overview](#)

Introduces the general concepts of the `PrintDialog` component, which allows you to display a pre-configured dialog box that users can use to select a printer, choose pages to print, and determine print-related settings.

[How to: Display the PrintDialog Component](#)

Explains how to display the dialog and where it saves properties.

Reference

[PrintDialog](#)

Provides reference information on the class and its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

PrintDialog Component Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [PrintDialog](#) component is a pre-configured dialog box used to select a printer, choose the pages to print, and determine other print-related settings in Windows-based applications. Use it as a simple solution for printer and print-related settings selection in lieu of configuring your own dialog box. You can enable users to print many parts of their documents: print all, print a selected page range, or print a selection. By relying on standard Windows dialog boxes, you create applications whose basic functionality is immediately familiar to users. The [PrintDialog](#) component inherits from the [CommonDialog](#) class.

Working with the Component

Use the [ShowDialog](#) method to display the dialog box at run time. This component has properties that relate to either a single print job ([PrintDocument](#) class) or the settings of an individual printer ([PrinterSettings](#) class). Either of these, in turn, may be shared by multiple printers.

When it is added to a form, the [PrintDialog](#) component appears in the tray at the bottom of the Windows Forms Designer.

See Also

[PrintDialog](#)

[PrintDialog Component](#)

How to: Display the PrintDialog Component

5/4/2018 • 1 min to read • [Edit Online](#)

The [PrintDialog](#) component is the standard Windows print dialog box that many of your users will be familiar with. Because your users will be immediately comfortable with it, it would be beneficial for you to use the [PrintDialog](#) component.

To display the PrintDialog component

- Call the [ShowDialog](#) method from within the code of your application.

Once the component is shown, users will interact with it, setting the properties of the print job. These are saved in the [PrinterSetting](#) class (and the [PageSettings](#) class, if the user accesses the [PageSetupDialog](#) Component through the [PrintDialog](#) component) associated with that print job. You can then make calls to the properties they set to determine the specifics of the print job.

See Also

[How to: Create Standard Windows Forms Print Jobs](#)

[How to: Capture User Input from a PrintDialog at Run Time](#)

[PrintPreviewDialog Control](#)

[PrintDialog Component](#)

[Windows Forms Print Support](#)

[Windows Forms Controls](#)

PrintDocument Component (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `PrintDocument` component is used to set the properties that describe what to print and then to print the document within Windows-based applications. It can be used in conjunction with the `PrintDialog` component to be in command of all aspects of document printing.

In This Section

[PrintDocument Component Overview](#)

Introduces the general concepts of the `PrintDocument` component, which allows you to set properties describing what to print and launches printing in a Windows-based application.

Reference

[PrintDocument](#)

Provides reference information on the class and its members.

Related Sections

[Windows Forms Print Support](#)

Presents a list of printing topics related to Windows Forms.

[PrintDialog Component](#)

Introduces the general concepts of the `PrintDialog` component, which allows you to display a pre-configured dialog box that users can use to select a printer, choose pages to print, and determine print-related settings.

[PrintPreviewControl Control](#)

Introduces the general concepts of the `PrintPreviewControl`, which you can use to design your own print preview dialog box or component.

[PrintPreviewDialog Control](#)

Introduces the general concepts of the `PrintPreviewDialog` control, which allows you to display a pre-configured dialog box that users can use to see a version of their document as it will look when it prints.

PrintDocument Component Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [PrintDocument](#) component is used to set the properties that describe what to print and the ability to print the document within Windows-based applications. It can be used in conjunction with the [PrintDialog](#) component to be in control of all aspects of document printing.

Working with the PrintDocument Component

Two of the main scenarios that involve the [PrintDocument](#) component are:

- Simple print jobs, such as printing an individual text file. In such a case you would add the [PrintDocument](#) component to a Windows Form, then add programming logic that prints a file in the [PrintPage](#) event handler. The programming logic should culminate with the [Print](#) method to print the document. This method sends a [Graphics](#) object, contained in the [Graphics](#) property of the [PrintPageEventArgs](#) class, to the printer. For an example that shows how to print a text document using the [PrintDocument](#) component, see [How to: Print a Multi-Page Text File in Windows Forms](#).
- More complex print jobs, such as a situation where you will want to reuse printing logic you have written. In such a case you would derive a new component from the [PrintDocument](#) component and override (see [Overrides](#) for Visual Basic or [override](#) for C#) the [PrintPage](#) event.

When it is added to a form, the [PrintDocument](#) component appears in the tray at the bottom of the Windows Forms Designer.

See Also

[Graphics](#)

[PrintDocument](#)

[Windows Forms Print Support](#)

[PrintDocument Component](#)

PrintPreviewControl Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `PrintPreviewControl` is used to display a document as it will appear when printed. This control has no buttons or other user interface elements, so typically you use the `PrintPreviewControl` only if you wish to write your own print-preview user interface. If you want the standard user interface, use a [PrintPreviewDialog](#) control.

In This Section

[PrintPreviewControl Control Overview](#)

Introduces the general concepts of the `PrintPreviewControl`, which you can use to design your own print preview dialog or component.

Reference

[PrintPreviewControl](#)

Provides reference information on the class and its members.

Related Sections

[PrintPreviewDialog Control](#)

Describes an alternate way to create print preview functionality.

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

PrintPreviewControl Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [PrintPreviewControl](#) is used to display a [PrintDocument](#) as it will appear when printed. This control has no buttons or other user interface elements, so typically you use the [PrintPreviewControl](#) only if you wish to write your own print-preview user interface. If you want the standard user interface, use a [PrintPreviewDialog](#) control; see [PrintPreviewDialog Control Overview](#) for an overview.

Key Properties

The control's key property is [Document](#), which sets the document to be previewed. The document must be a [PrintDocument](#) object. For an overview of creating documents for printing, see [PrintDocument Component Overview](#) and [Windows Forms Print Support](#). The [Columns](#) and [Rows](#) properties determine the number of pages displayed horizontally and vertically on the control. Antialiasing can make the text appear smoother, but it can also make the display slower; to use it, set the [UseAntiAlias](#) property to `true`.

See Also

[PrintPreviewControl](#)
[PrintPreviewDialog Control Overview](#)
[PrintPreviewControl Control](#)
[Dialog-Box Controls and Components](#)

PrintPreviewDialog Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `PrintPreviewDialog` control is a pre-configured dialog box used to display how a document will appear when printed. Use it within your Windows-based application as a simple solution in lieu of configuring your own dialog box. The control contains buttons for printing, zooming in, displaying one or multiple pages, and closing the dialog box.

In This Section

[PrintPreviewDialog Control Overview](#)

Introduces the general concepts of the `PrintPreviewDialog` control, which allows you to display a pre-configured dialog box that users can use to see a version of their document as it will look when it prints.

[How to: Display Print Preview in Windows Forms Applications](#)

Explains how to view a page that is to be printed by using an instance of the `PrintPreviewDialog` control at run time.

Reference

[PrintPreviewDialog](#)

Provides reference information on the class and its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

[Essential Code for Windows Forms Dialog Boxes](#)

Discusses the Windows Forms dialog box controls and components and the code necessary for executing their basic functions. (MSDN Online Library technical article)

[Dialog-Box Controls and Components](#)

Lists the different dialog-box controls for Windows Forms.

[Dialog Boxes in Windows Forms](#)

Describes how to create a dialog box for a Windows Form.

PrintPreviewDialog control overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [PrintPreviewDialog](#) control is a pre-configured dialog box used to display how a [PrintDocument](#) will appear when printed. Use it within your Windows-based application as a simple solution instead of configuring your own dialog box. The control contains buttons for printing, zooming in, displaying one or multiple pages, and closing the dialog box.

Key properties and methods

The control's key property is [Document](#), which sets the document to be previewed. The document must be a [PrintDocument](#) object. In order to display the dialog box, you must call its [ShowDialog](#) method. Anti-aliasing can make the text appear smoother, but it can also make the display slower; to use it, set the [UseAntiAlias](#) property to `true`.

Certain properties are available through the [PrintPreviewControl](#) that the [PrintPreviewDialog](#) contains. (You do not have to add this [PrintPreviewControl](#) to the form; it is automatically contained within the [PrintPreviewDialog](#) when you add the dialog to your form.) Examples of properties available through the [PrintPreviewControl](#) are the [Columns](#) and [Rows](#) properties, which determine the number of pages displayed horizontally and vertically on the control. You can access the [Columns](#) property as `PrintPreviewDialog1.PrintPreviewControl.Columns` in Visual Basic, `printPreviewDialog1.PrintPreviewControl.Columns` in Visual C#, or `printPreviewDialog1->PrintPreviewControl->Columns` in Visual C++.

PrintPreviewDialog performance

Under the following conditions, the [PrintPreviewDialog](#) control initializes very slowly:

- A network printer is used.
- User preferences for this printer, such as duplex settings, are modified.

For apps running on the .NET Framework 4.5.2, you can add the following key to the `<appSettings>` section of your configuration file to improve the performance of [PrintPreviewDialog](#) control initialization:

```
<appSettings>
  <add key="EnablePrintPreviewOptimization" value="true" />
</appSettings>
```

If the `EnablePrintPreviewOptimization` key is set to any other value, or if the key is not present, the optimization is not applied.

For apps running on the .NET Framework 4.6 or later versions, you can add the following switch to the `<AppContextSwitchOverrides>` element in the `<runtime>` section of your app config file:

```
<runtime>
  <!-- AppContextSwitchOverrides values are in the form of 'key1=true|false;key2=true|false -->
  <AppContextSwitchOverrides value = "Switch.System.Drawing.Printing.OptimizePrintPreview=true" />
</runtime>
```

If the switch is not present or if it is set to any other value, the optimization is not applied.

If you use the [QueryPageSettings](#) event to modify printer settings, the performance of the [PrintPreviewDialog](#) control will not improve even if an optimization configuration switch is set.

See also

[PrintPreviewDialog](#)

[PrintPreviewControl Control Overview](#)

[PrintPreviewDialog Control](#)

[Dialog-Box Controls and Components](#)

How to: Display Print Preview in Windows Forms Applications

5/4/2018 • 1 min to read • [Edit Online](#)

You can use the [PrintPreviewDialog](#) control to enable users to display a document, often before it is to be printed.

To do this, you need to specify an instance of the [PrintDocument](#) class; this is the document to be printed. For more information about using print preview with the [PrintDocument](#) component, see [How to: Print in Windows Forms Using Print Preview](#).

NOTE

To use the [PrintPreviewDialog](#) control at run time, users must have a printer installed on their computer, either locally or through a network, as this is partly how the [PrintPreviewDialog](#) component determines how a document will look when printed.

The [PrintPreviewDialog](#) control uses the [PrinterSettings](#) class. Additionally, the [PrintPreviewDialog](#) control uses the [PageSettings](#) class, just as the [PrintPreviewDialog](#) component does. The print document specified in the [PrintPreviewDialog](#) control's [Document](#) property refers to instances of both the [PrinterSettings](#) and [PageSettings](#) classes, and these are used to render the document in the preview window.

To view pages using the PrintPreviewDialog control

- Use the [ShowDialog](#) method to display the dialog box, specifying the [PrintDocument](#) to use.

In the following code example, the [Button](#) control's [Click](#) event handler opens an instance of the [PrintPreviewDialog](#) control. The print document is specified in the [Document](#) property. In the example below, no print document is specified.

The example requires that your form has a [Button](#) control, a [PrintDocument](#) component named `myDocument`, and a [PrintPreviewDialog](#) control.

```
Private Sub Button1_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles Button1.Click  
    ' The print document 'myDocument' used below  
    ' is merely for an example.  
    ' You will have to specify your own print document.  
    PrintPreviewDialog1.Document = myDocument  
    PrintPreviewDialog1.ShowDialog()  
End Sub
```

```
private void button1_Click(object sender, System.EventArgs e)  
{  
    // The print document 'myDocument' used below  
    // is merely for an example.  
    // You will have to specify your own print document.  
    printPreviewDialog1.Document = myDocument;  
    printPreviewDialog1.ShowDialog();  
}
```

```
private:  
    void button1_Click(System::Object ^ sender,  
                      System::EventArgs ^ e)  
    {  
        // The print document 'myDocument' used below  
        // is merely for an example.  
        // You will have to specify your own print document.  
        printPreviewDialog1->Document = myDocument;  
        printPreviewDialog1->>ShowDialog();  
    }
```

(Visual C#, Visual C++) Place the following code in the form's constructor to register the event handler.

```
this.button1.Click += new System.EventHandler(this.button1_Click);
```

```
this->button1->Click += gcnew  
    System::EventHandler(this, &Form1::button1_Click);
```

See Also

[PrintDocument Component](#)

[PrintPreviewDialog Control](#)

[Windows Forms Print Support](#)

[Windows Forms](#)

ProgressBar Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

IMPORTANT

The [ToolStripProgressBar](#) control replaces and adds functionality to the [ProgressBar](#) control; however, the [ProgressBar](#) control is retained for both backward compatibility and future use, if you choose.

The Windows Forms [ProgressBar](#) control indicates the progress of an action by displaying an appropriate number of rectangles arranged in a horizontal bar. When the action is complete, the bar is filled. Progress bars are commonly used to give the user an indication of how long to wait for a protracted action to complete—for instance, when a large file is being loaded.

In This Section

[ProgressBar Control Overview](#)

Introduces the general concepts of the [ProgressBar](#) control, which enables you to graphically display the progress of an operation.

[How to: Set the Value Displayed by the Windows Forms ProgressBar Control](#)

Discusses a number of different ways to increase the value displayed by the [ProgressBar](#) control.

Reference

[ProgressBar](#)

Provides reference information on the class and its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

ProgressBar Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

IMPORTANT

The [ToolStripProgressBar](#) control replaces and adds functionality to the [ProgressBar](#) control; however, the [ProgressBar](#) control is retained for both backward compatibility and future use, if you choose.

The Windows Forms [ProgressBar](#) control indicates the progress of a process by displaying an appropriate number of rectangles arranged in a horizontal bar. When the process is complete, the bar is filled. Progress bars are commonly used to give the user an idea of how long to wait for a process to complete; for instance, when a large file is being loaded.

NOTE

The [ProgressBar](#) control can only be oriented horizontally on the form.

Key Properties and Methods

The key properties of the [ProgressBar](#) control are [Value](#), [Minimum](#), and [Maximum](#). The [Minimum](#) and [Maximum](#) properties set the maximum and minimum values the progress bar can display. The [Value](#) property represents the progress that has been made toward completing the operation. Because the bar displayed in the control is composed of blocks, the value displayed by the [ProgressBar](#) control only approximates the [Value](#) property's current value. Based on the size of the [ProgressBar](#) control, the [Value](#) property determines when to display the next block.

The most common way to update the current progress value is to write code to set the [Value](#) property. In the example of loading a large file, you might set the maximum to the size of the file in kilobytes. For example, if the [Maximum](#) property is set to 100, the [Minimum](#) property is set to 10, and the [Value](#) property is set to 50, 5 rectangles will be displayed. This is half of the number that can be displayed.

However, there are other ways to modify the value displayed by the [ProgressBar](#) control, aside from setting the [Value](#) property directly. The [Step](#) property can be used to specify a value to increment the [Value](#) property by. Then, calling the [PerformStep](#) method will increment the value. To vary the increment value, you can use the [Increment](#) method and specify a value with which to increment the [Value](#) property.

Another control that graphically informs the user about a current action is the [StatusBar](#) control.

IMPORTANT

The [StatusStrip](#) and [ToolStripStatusLabel](#) controls replace and add functionality to the [StatusBar](#) and [StatusBarPanel](#) controls; however, the [StatusBar](#) and [StatusBarPanel](#) controls are retained for both backward compatibility and future use, if you choose.

See Also

[ProgressBar](#)
[ProgressBar Control](#)

How to: Set the Value Displayed by the Windows Forms ProgressBar Control

5/4/2018 • 7 min to read • [Edit Online](#)

IMPORTANT

The [ToolStripProgressBar](#) control replaces and adds functionality to the [ProgressBar](#) control; however, the [ProgressBar](#) control is retained for both backward compatibility and future use, if you choose.

The .NET Framework gives you several different ways to display a given value within the [ProgressBar](#) control. Which approach you choose will depend on the task at hand or the problem you are solving. The following table shows the approaches you can choose.

APPROACH	DESCRIPTION
Set the value of the ProgressBar control directly.	This approach is useful for tasks where you know the total of the item measured that will be involved, such as reading records from a data source. Additionally, if you only need to set the value once or twice, this is an easy way to do it. Finally, use this process if you need to decrease the value displayed by the progress bar.
Increase the ProgressBar display by a fixed value.	This approach is useful when you are displaying a simple count between the minimum and maximum, such as elapsed time or the number of files that have been processed out of a known total.
Increase the ProgressBar display by a value that varies.	This approach is useful when you need to change the displayed value a number of times in different amounts. An example would be showing the amount of hard-disk space being consumed while writing a series of files to the disk.

The most direct way to set the value displayed by a progress bar is by setting the [Value](#) property. This can be done either at design time or at run time.

To set the [ProgressBar](#) value directly

1. Set the [ProgressBar](#) control's [Minimum](#) and [Maximum](#) values.
2. In code, set the control's [Value](#) property to an integer value between the minimum and maximum values you have established.

NOTE

If you set the [Value](#) property outside the boundaries established by the [Minimum](#) and [Maximum](#) properties, the control throws an [ArgumentException](#) exception.

The following code example illustrates how to set the [ProgressBar](#) value directly. The code reads records from a data source and updates the progress bar and label every time a data record is read. This example requires that your form has a [Label](#) control, a [ProgressBar](#) control, and a data table with a row called `CustomerRow` with `FirstName` and `LastName` fields.

```

Public Sub CreateNewRecords()
    ' Sets the progress bar's Maximum property to
    ' the total number of records to be created.
    ProgressBar1.Maximum = 20

    ' Creates a new record in the dataset.
    ' NOTE: The code below will not compile, it merely
    ' illustrates how the progress bar would be used.
    Dim anyRow As CustomerRow = DatasetName.ExistingTable.NewRow
    anyRow.FirstName = "Stephen"
    anyRow.LastName = "James"
    ExistingTable.Rows.Add(anyRow)

    ' Increases the value displayed by the progress bar.
    ProgressBar1.Value += 1
    ' Updates the label to show that a record was read.
    Label1.Text = "Records Read = " & ProgressBar1.Value.ToString()
End Sub

```

```

public void createNewRecords()
{
    // Sets the progress bar's Maximum property to
    // the total number of records to be created.
    progressBar1.Maximum = 20;

    // Creates a new record in the dataset.
    // NOTE: The code below will not compile, it merely
    // illustrates how the progress bar would be used.
    CustomerRow anyRow = DatasetName.ExistingTable.NewRow();
    anyRow.FirstName = "Stephen";
    anyRow.LastName = "James";
    ExistingTable.Rows.Add(anyRow);

    // Increases the value displayed by the progress bar.
    progressBar1.Value += 1;
    // Updates the label to show that a record was read.
    label1.Text = "Records Read = " + progressBar1.Value.ToString();
}

```

If you are displaying progress that proceeds by a fixed interval, you can set the value and then call a method that increases the [ProgressBar](#) control's value by that interval. This is useful for timers and other scenarios where you are not measuring progress as a percentage of the whole.

To increase the progress bar by a fixed value

1. Set the [ProgressBar](#) control's [Minimum](#) and [Maximum](#) values.
2. Set the control's [Step](#) property to an integer representing the amount to increase the progress bar's displayed value.
3. Call the [PerformStep](#) method to change the value displayed by the amount set in the [Step](#) property.

The following code example illustrates how a progress bar can maintain a count of the files in a copy operation.

In the following example, as each file is read into memory, the progress bar and label are updated to reflect the total files read. This example requires that your form has a [Label](#) control and a [ProgressBar](#) control.

```

Public Sub LoadFiles()
    ' Sets the progress bar's minimum value to a number representing
    ' no operations complete -- in this case, no files read.
    ProgressBar1.Minimum = 0
    ' Sets the progress bar's maximum value to a number representing
    ' all operations complete -- in this case, all five files read.
    ProgressBar1.Maximum = 5
    ' Sets the Step property to amount to increase with each iteration.
    ' In this case, it will increase by one with every file read.
    ProgressBar1.Step = 1

    ' Dimensions a counter variable.
    Dim i As Integer
    ' Uses a For...Next loop to iterate through the operations to be
    ' completed. In this case, five files are to be copied into memory,
    ' so the loop will execute 5 times.
    For i = 0 To 4
        ' Insert code to copy a file
        ProgressBar1.PerformStep()
        ' Update the label to show that a file was read.
        Label1.Text = "# of Files Read = " & ProgressBar1.Value.ToString
    Next i
End Sub

```

```

public void loadFiles()
{
    // Sets the progress bar's minimum value to a number representing
    // no operations complete -- in this case, no files read.
    progressBar1.Minimum = 0;
    // Sets the progress bar's maximum value to a number representing
    // all operations complete -- in this case, all five files read.
    progressBar1.Maximum = 5;
    // Sets the Step property to amount to increase with each iteration.
    // In this case, it will increase by one with every file read.
    progressBar1.Step = 1;

    // Uses a for loop to iterate through the operations to be
    // completed. In this case, five files are to be copied into memory,
    // so the loop will execute 5 times.
    for (int i = 0; i <= 4; i++)
    {
        // Inserts code to copy a file
        progressBar1.PerformStep();
        // Updates the label to show that a file was read.
        label1.Text = "# of Files Read = " + progressBar1.Value.ToString();
    }
}

```

Finally, you can increase the value displayed by a progress bar so that each increase is a unique amount. This is useful when you are keeping track of a series of unique operations, such as writing files of different sizes to a hard disk, or measuring progress as a percentage of the whole.

To increase the progress bar by a dynamic value

1. Set the [ProgressBar](#) control's [Minimum](#) and [Maximum](#) values.
2. Call the [Increment](#) method to change the value displayed by an integer you specify.

The following code example illustrates how a progress bar can calculate how much disk space has been used during a copy operation.

In the following example, as each file is written to the hard disk, the progress bar and label are updated to reflect the amount of hard-disk space available. This example requires that your form has a [Label](#) control

and a [ProgressBar](#) control.

```
Public Sub ReadFiles()
    ' Sets the progress bar's minimum value to a number
    ' representing the hard disk space before the files are read in.
    ' You will most likely have to set this using a system call.
    ' NOTE: The code below is meant to be an example and
    ' will not compile.
    ProgressBar1.Minimum = AvailableDiskSpace()
    ' Sets the progress bar's maximum value to a number
    ' representing the total hard disk space.
    ' You will most likely have to set this using a system call.
    ' NOTE: The code below is meant to be an example
    ' and will not compile.
    ProgressBar1.Maximum = TotalDiskSpace()

    ' Dimension a counter variable.
    Dim i As Integer
    ' Uses a For...Next loop to iterate through the operations to be
    ' completed. In this case, five files are to be written to the disk,
    ' so it will execute the loop 5 times.
    For i = 1 To 5
        ' Insert code to read a file into memory and update file size.
        ' Increases the progress bar's value based on the size of
        ' the file currently being written.
        ProgressBar1.Increment(FileSize)
        ' Updates the label to show available drive space.
        Label1.Text = "Current Disk Space Used = " &
            ProgressBar1.Value.ToString()
    Next i
End Sub
```

```
public void readFiles()
{
    // Sets the progress bar's minimum value to a number
    // representing the hard disk space before the files are read in.
    // You will most likely have to set this using a system call.
    // NOTE: The code below is meant to be an example and
    // will not compile.
    progressBar1.Minimum = AvailableDiskSpace();
    // Sets the progress bar's maximum value to a number
    // representing the total hard disk space.
    // You will most likely have to set this using a system call.
    // NOTE: The code below is meant to be an example
    // and will not compile.
    progressBar1.Maximum = TotalDiskSpace();

    // Uses a for loop to iterate through the operations to be
    // completed. In this case, five files are to be written
    // to the disk, so it will execute the loop 5 times.
    for (int i = 1; i<= 5; i++)
    {
        // Insert code to read a file into memory and update file size.
        // Increases the progress bar's value based on the size of
        // the file currently being written.
        progressBar1.Increment(FileSize);
        // Updates the label to show available drive space.
        label1.Text = "Current Disk Space Used = " + progressBar1.Value.ToString();
    }
}
```

See Also

[ProgressBar](#)

[ToolStripProgressBar](#)

[ProgressBar Control Overview](#)

[ProgressBar Control](#)

RadioButton Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

Windows Forms `RadioButton` controls present a set of two or more mutually exclusive choices to the user. While radio buttons and check boxes may appear to function similarly, there is an important difference: when a user selects a radio button, the other radio buttons in the same group cannot be selected as well.

In This Section

[RadioButton Control Overview](#)

Explains what this control is and its key features and properties.

[How to: Group Windows Forms RadioButton Controls to Function as a Set](#)

Explains how to group radio buttons as a set, of which only one may be selected.

Reference

[RadioButton class](#)

Describes this class and has links to all its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

RadioButton Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

Windows Forms [RadioButton](#) controls present a set of two or more mutually exclusive choices to the user. While radio buttons and check boxes may appear to function similarly, there is an important difference: when a user selects a radio button, the other radio buttons in the same group cannot be selected as well. In contrast, any number of check boxes can be selected. Defining a radio button group tells the user, "Here is a set of choices from which you can choose one and only one."

Using the Control

When a [RadioButton](#) control is clicked, its [Checked](#) property is set to `true` and the [Click](#) event handler is called. The [CheckedChanged](#) event is raised when the value of the [Checked](#) property changes. If the [AutoCheck](#) property is set to `true` (the default), when the radio button is selected all others in the group are automatically cleared. This property is usually only set to `false` when validation code is used to make sure the radio button selected is an allowable option. The text displayed within the control is set with the [Text](#) property, which can contain access key shortcuts. An access key enables a user to "click" the control by pressing the ALT key with the access key. For more information, see [How to: Create Access Keys for Windows Forms Controls](#) and [How to: Set the Text Displayed by a Windows Forms Control](#).

The [RadioButton](#) control can appear like a command button, which appears to have been depressed if selected, if the [Appearance](#) property is set to [Button](#). Radio buttons can also display images using the [Image](#) and [ImageList](#) properties. For more information, see [How to: Set the Image Displayed by a Windows Forms Control](#).

See Also

[RadioButton](#)
[Panel Control Overview](#)
[GroupBox Control Overview](#)
[CheckBox Control Overview](#)
[How to: Create Access Keys for Windows Forms Controls](#)
[How to: Set the Text Displayed by a Windows Forms Control](#)
[How to: Group Windows Forms RadioButton Controls to Function as a Set](#)
[RadioButton Control](#)

How to: Group Windows Forms RadioButton Controls to Function as a Set

5/4/2018 • 1 min to read • [Edit Online](#)

Windows Forms [RadioButton](#) controls are designed to give users a choice among two or more settings, of which only one can be assigned to a procedure or object. For example, a group of [RadioButton](#) controls may display a choice of package carriers for an order, but only one of the carriers will be used. Therefore only one [RadioButton](#) at a time can be selected, even if it is a part of a functional group.

You group radio buttons by drawing them inside a container such as a [Panel](#) control, a [GroupBox](#) control, or a form. All radio buttons that are added directly to a form become one group. To add separate groups, you must place them inside panels or group boxes. For more information about panels or group boxes, see [Panel Control Overview](#) or [GroupBox Control Overview](#).

To group RadioButton controls as a set to function independently of other sets

1. Drag a [GroupBox](#) or [Panel](#) control from the **Windows Forms** tab on the **Toolbox** onto the form.
2. Draw [RadioButton](#) controls on the [GroupBox](#) or [Panel](#) control.

See Also

- [RadioButton](#)
- [RadioButton Control Overview](#)
- [Panel Control Overview](#)
- [GroupBox Control Overview](#)
- [CheckBox Control Overview](#)
- [RadioButton Control](#)

RichTextBox Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `RichTextBox` control is used for displaying, entering, and manipulating text with formatting. The `RichTextBox` control does everything the `TextBox` control does, but it can also display fonts, colors, and links; load text and embedded images from a file; undo and redo editing operations; and find specified characters. The `RichTextBox` control is typically used to provide text manipulation and display features similar to word processing applications such as Microsoft Word. Like the `TextBox` control, the `RichTextBox` control can display scroll bars; but unlike the `TextBox` control, it displays both horizontal and vertical scrollbars by default and has additional scrollbar settings.

In This Section

[RichTextBox Control Overview](#)

Introduces the general concepts of the `RichTextBox` control, which allows users to enter, display, and manipulate text with formatting options.

[How to: Determine When Formatting Attributes Change in the Windows Forms RichTextBox Control](#)

Explains how to keep track of changes in font and paragraph formatting in the `RichTextBox` control.

[How to: Display Scroll Bars in the Windows Forms RichTextBox Control](#)

Describes the many choices available for scroll bars in the `RichTextBox` control.

[How to: Display Web-Style Links with the Windows Forms RichTextBox Control](#)

Explains how to link to Web sites from the `RichTextBox` control.

[How to: Enable Drag-and-Drop Operations with the Windows Forms RichTextBox Control](#)

Provides instructions for dragging data into the `RichTextBox` control.

[How to: Load Files into the Windows Forms RichTextBox Control](#)

Provides instructions for loading an existing file into the `RichTextBox` control.

[How to: Save Files with the Windows Forms RichTextBox Control](#)

Provides instructions for saving the contents of the `RichTextBox` control to a file.

[How to: Set Font Attributes for the Windows Forms RichTextBox Control](#)

Describes how to set the font family, size, style, and color of text in the `RichTextBox` control.

[How to: Set Indents, Hanging Indents, and Bulleted Paragraphs with the Windows Forms RichTextBox Control](#)

Describes how to format paragraphs in the `RichTextBox` control.

Reference

[RichTextBox class](#)

Describes this class and has links to all its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

[TextBox Control](#)

Introduces the general concepts of the [TextBox](#) control, which allows editable, multiline input from the user.

RichTextBox Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [RichTextBox](#) control is used for displaying, entering, and manipulating text with formatting. The [RichTextBox](#) control does everything the [TextBox](#) control does, but it can also display fonts, colors, and links; load text and embedded images from a file; and find specified characters. The [RichTextBox](#) control is typically used to provide text manipulation and display features similar to word processing applications such as Microsoft Word. Like the [TextBox](#) control, the [RichTextBox](#) control can display scroll bars; but unlike the [TextBox](#) control, its default setting is to display both horizontal and vertical scrollbars as needed, and it has additional scrollbar settings.

Working with the RichTextBox Control

As with the [TextBox](#) control, the text displayed is set by the [Text](#) property. The [RichTextBox](#) control has numerous properties to format text. For details on these properties, see [How to: Set Font Attributes for the Windows Forms RichTextBox Control](#) and [How to: Set Indents, Hanging Indents, and Bulleted Paragraphs with the Windows Forms RichTextBox Control](#). To manipulate files, the [LoadFile](#) and [SaveFile](#) methods can display and write multiple file formats including plain text, Unicode plain text, and Rich Text Format (RTF). The possible file formats are listed in [RichTextBoxStreamType](#). You can use the [Find](#) method to find strings of text or specific characters.

You can also use a [RichTextBox](#) control for Web-style links by setting the [DetectUrls](#) property to `true` and writing code to handle the [LinkClicked](#) event. For more information, see [How to: Display Web-Style Links with the Windows Forms RichTextBox Control](#). You can prevent the user from manipulating some or all of the text in the control by setting the [SelectionProtected](#) property to `true`.

You can undo and redo most edit operations in a [RichTextBox](#) control by calling the [Undo](#) and [Redo](#) methods. The [CanRedo](#) method enables you to determine whether the last operation the user has undone can be reapplied to the control.

See Also

[RichTextBox](#)

[RichTextBox Control](#)

[TextBox Control Overview](#)

How to: Determine When Formatting Attributes Change in the Windows Forms RichTextBox Control

5/4/2018 • 1 min to read • [Edit Online](#)

A common use of the Windows Forms [RichTextBox](#) control is formatting text with attributes such as font options or paragraph styles. Your application may need to keep track of any changes in text formatting for the purpose of displaying a toolbar, as in many word-processing applications.

To respond to changes in formatting attributes

1. Write code in the [SelectionChanged](#) event handler to perform an appropriate action depending on the value of the attribute. The following example changes the appearance of a toolbar button depending on the value of the [SelectionBullet](#) property. The toolbar button will only be updated when the insertion point is moved in the control.

The example below assumes a form with a [RichTextBox](#) control and a [ToolBar](#) control that contains a toolbar button. For more information about toolbars and toolbar buttons, see [How to: Add Buttons to a ToolBar Control](#).

```
' The following code assumes the existence of a toolbar control
' with at least one toolbar button.

Private Sub RichTextBox1_SelectionChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles
RichTextBox1.SelectionChanged
    If RichTextBox1.SelectionBullet = True Then
        ' Bullet button on toolbar should appear pressed
        ToolBarButton1.Pushed = True
    Else
        ' Bullet button on toolbar should appear unpressed
        ToolBarButton1.Pushed = False
    End If
End Sub
```

```
// The following code assumes the existence of a toolbar control
// with at least one toolbar button.

private void richTextBox1_SelectionChanged(object sender,
System.EventArgs e)
{
    if (richTextBox1.SelectionBullet == true)
    {
        // Bullet button on toolbar should appear pressed
        toolBarButton1.Pushed = true;
    }
    else
    {
        // Bullet button on toolbar should appear unpressed
        toolBarButton1.Pushed = false;
    }
}
```

```
// The following code assumes the existence of a toolbar control
// with at least one toolbar button.
private:
    System::Void richTextBox1_SelectionChanged(
        System::Object ^ sender, System::EventArgs ^ e)
{
    if (richTextBox1->SelectionBullet == true)
    {
        // Bullet button on toolbar should appear pressed
        toolBarButton1->Pushed = true;
    }
    else
    {
        // Bullet button on toolbar should appear unpressed
        toolBarButton1->Pushed = false;
    }
}
```

See Also

[SelectionChanged](#)

[RichTextBox](#)

[RichTextBox Control](#)

[Controls to Use on Windows Forms](#)

How to: Display Scroll Bars in the Windows Forms RichTextBox Control

5/4/2018 • 1 min to read • [Edit Online](#)

By default, the Windows Forms [RichTextBox](#) control displays horizontal and vertical scroll bars as necessary. There are seven possible values for the [ScrollBars](#) property of the [RichTextBox](#) control, which are described in the table below.

To display scroll bars in a RichTextBox control

1. Set the [Multiline](#) property to `true`. No type of scroll bar, including horizontal, will display if the [Multiline](#) property is set to `false`.
2. Set the [ScrollBars](#) property to an appropriate value of the [RichTextBoxScrollBars](#) enumeration.

VALUE	DESCRIPTION
Both (default)	Displays horizontal or vertical scroll bars, or both, only when text exceeds the width or length of the control.
None	Never displays any type of scroll bar.
Horizontal	Displays a horizontal scroll bar only when the text exceeds the width of the control. (For this to occur, the WordWrap property must be set to <code>false</code> .)
Vertical	Displays a vertical scroll bar only when the text exceeds the height of the control.
ForcedHorizontal	Displays a horizontal scroll bar when the WordWrap property is set to <code>false</code> . The scroll bar appears dimmed when text does not exceed the width of the control.
ForcedVertical	Always displays a vertical scroll bar. The scroll bar appears dimmed when text does not exceed the length of the control.
ForcedBoth	Always displays a vertical scrollbar. Displays a horizontal scroll bar when the WordWrap property is set to <code>false</code> . The scroll bars appear grayed when text does not exceed the width or length of the control.

3. Set the [WordWrap](#) property to an appropriate value.

VALUE	DESCRIPTION
<code>false</code>	Text in the control is not automatically adjusted to fit the width of the control, so it will scroll to the right until a line break is reached. Use this value if you chose horizontal scroll bars or both, above.

VALUE	DESCRIPTION
<code>true</code> (default)	Text in the control is automatically adjusted to fit the width of the control. The horizontal scrollbar will not appear. Use this value if you chose vertical scroll bars or none, above, to display one or more paragraphs.

See Also

[RichTextBoxScrollBars](#)

[RichTextBox](#)

[RichTextBox Control](#)

[Controls to Use on Windows Forms](#)

How to: Display Web-Style Links with the Windows Forms RichTextBox Control

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [RichTextBox](#) control can display Web links as colored and underlined. You can write code that opens a browser window showing the Web site specified in the link text when the link is clicked.

To link to a Web page with the RichTextBox control

1. Set the [Text](#) property to a string that includes a valid URL (for example, "<http://www.microsoft.com/>").
2. Make sure the [DetectUrls](#) property is set to `true` (the default).
3. Create a new global instance of the [Process](#) object.
4. Write an event handler for the [LinkClicked](#) event that sends the browser the desired text.

In the example below, the [LinkClicked](#) event opens an instance of Internet Explorer to the URL specified in the [Text](#) property of the [RichTextBox](#) control. This example assumes a form with a [RichTextBox](#) control.

IMPORTANT

In calling the [Process.Start](#) method, you will encounter a [SecurityException](#) exception if you are running the code in a partial-trust context because of insufficient privileges. For more information, see [Code Access Security Basics](#).

```
Public p As New System.Diagnostics.Process
Private Sub RichTextBox1_LinkClicked _
    (ByVal sender As Object, ByVal e As _
     System.Windows.Forms.LinkClickedEventArgs) _
    Handles RichTextBox1.LinkClicked
    ' Call Process.Start method to open a browser
    ' with link text as URL.
    p = System.Diagnostics.Process.Start("IExplore.exe", e.LinkText)
End Sub
```

```
public System.Diagnostics.Process p = new System.Diagnostics.Process();

private void richTextBox1_LinkClicked(object sender,
System.Windows.Forms.LinkClickedEventArgs e)
{
    // Call Process.Start method to open a browser
    // with link text as URL.
    p = System.Diagnostics.Process.Start("IExplore.exe", e.LinkText);
}
```

```

public:
    System::Diagnostics::Process ^ p;

private:
    void richTextBox1_LinkClicked(System::Object ^ sender,
        System::Windows::Forms::LinkClickedEventArgs ^ e)
    {
        // Call Process.Start method to open a browser
        // with link text as URL.
        p = System::Diagnostics::Process::Start("IExplore.exe",
            e->LinkText);
    }

```

(Visual C++) You must initialize process `p`, which you can do by including the following statement in the constructor of your form:

```
p = gcnew System::Diagnostics::Process();
```

(Visual C#, Visual C++) Place the following code in the form's constructor to register the event handler.

```
this.richTextBox1.LinkClicked += new
    System.Windows.Forms.LinkClickedEventHandler
    (this.richTextBox1_LinkClicked);
```

```
this->richTextBox1->LinkClicked += gcnew
    System::Windows::Forms::LinkClickedEventHandler
    (this, &Form1::richTextBox1_LinkClicked);
```

It is important to immediately stop the process you have created once you have finished working with it. Referring to the code presented above, your code to stop the process might look like this:

```
Public Sub StopWebProcess()
    p.Kill()
End Sub
```

```
public void StopWebProcess()
{
    p.Kill();
}
```

```
public: void StopWebProcess()
{
    p->Kill();
}
```

See Also

[DetectUrls](#)

[LinkClicked](#)

[RichTextBox](#)

[RichTextBox Control](#)

[Controls to Use on Windows Forms](#)

How to: Enable Drag-and-Drop Operations with the Windows Forms RichTextBox Control

5/4/2018 • 2 min to read • [Edit Online](#)

Drag-and-drop operations with the Windows Forms [RichTextBox](#) control are done by handling the [DragEnter](#) and [DragDrop](#) events. Thus, drag-and-drop operations are extremely simple with the [RichTextBox](#) control.

To enable drag operations in a RichTextBox control

1. Set the [AllowDrop](#) property of the [RichTextBox](#) control to `true`.
2. Write code in the event handler of the [DragEnter](#) event. Use an `if` statement to ensure that the data being dragged is of an acceptable type (in this case, text). The [DragEventArgs.Effect](#) property can be set to any value of the [DragDropEffects](#) enumeration.

```
Private Sub RichTextBox1_DragEnter(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DragEventArgs) _
Handles RichTextBox1.DragEnter
    If (e.Data.GetDataPresent(DataFormats.Text)) Then
        e.Effect = DragDropEffects.Copy
    Else
        e.Effect = DragDropEffects.None
    End If
End Sub
```

```
private void richTextBox1_DragEnter(object sender,
System.Windows.Forms.DragEventArgs e)
{
    if (e.Data.GetDataPresent(DataFormats.Text))
        e.Effect = DragDropEffects.Copy;
    else
        e.Effect = DragDropEffects.None;
}
```

```
private:
void richTextBox1_DragEnter(System::Object ^ sender,
    System::Windows::Forms::DragEventArgs ^ e)
{
    if (e->Data->GetDataPresent(DataFormats::Text))
        e->Effect = DragDropEffects::Copy;
    else
        e->Effect = DragDropEffects::None;
}
```

(Visual C# and Visual C++) Place the following code in the form's constructor to register the event handler.

```
this.richTextBox1.DragEnter += new
    System.Windows.Forms.DragEventHandler
    (this.richTextBox1_DragEnter);
```

```
this->richTextBox1->DragEnter += gcnew  
System::Windows::Forms::DragEventHandler  
(this, &Form1::richTextBox1_DragEnter);
```

3. Write code to handle the [DragDrop](#) event. Use the [DataObject.GetData](#) method to retrieve the data being dragged.

In the example below, the code sets the [Text](#) property of the [RichTextBox](#) control equal to the data being dragged. If there is already text in the [RichTextBox](#) control, the dragged text is inserted at the insertion point.

```
Private Sub RichTextBox1_DragDrop(ByVal sender As Object, _  
    ByVal e As System.Windows.Forms.DragEventArgs) _  
Handles RichTextBox1.DragDrop  
Dim i As Int16  
Dim s As String  
  
' Get start position to drop the text.  
i = RichTextBox1.SelectionStart  
s = RichTextBox1.Text.Substring(i)  
RichTextBox1.Text = RichTextBox1.Text.Substring(0, i)  
  
' Drop the text on to the RichTextBox.  
RichTextBox1.Text = RichTextBox1.Text + _  
    e.Data.GetData(DataFormats.Text).ToString()  
RichTextBox1.Text = RichTextBox1.Text + s  
End Sub
```

```
private void richTextBox1_DragDrop(object sender,  
System.Windows.Forms.DragEventArgs e)  
{  
    int i;  
    String s;  
  
    // Get start position to drop the text.  
    i = richTextBox1.SelectionStart;  
    s = richTextBox1.Text.Substring(i);  
    richTextBox1.Text = richTextBox1.Text.Substring(0,i);  
  
    // Drop the text on to the RichTextBox.  
    richTextBox1.Text = richTextBox1.Text +  
        e.Data.GetData(DataFormats.Text).ToString();  
    richTextBox1.Text = richTextBox1.Text + s;  
}
```

```

private:
    System::Void richTextBox1_DragDrop(System::Object ^ sender,
        System::Windows::Forms::DragEventArgs ^ e)
{
    int i;
    String ^s;

    // Get start position to drop the text.
    i = richTextBox1->SelectionStart;
    s = richTextBox1->Text->Substring(i);
    richTextBox1->Text = richTextBox1->Text->Substring(0,i);

    // Drop the text on to the RichTextBox.
    String ^str = String::Concat(richTextBox1->Text, e->Data
        ->GetData(DataFormats->Text)->ToString());
    richTextBox1->Text = String::Concat(str, s);
}

```

(Visual C# and Visual C++) Place the following code in the form's constructor to register the event handler.

```

this.richTextBox1.DragDrop += new
    System.Windows.Forms.DragEventHandler
    (this.richTextBox1_DragDrop);

```

```

this->richTextBox1->DragDrop += gcnew
    System::Windows::Forms::DragEventHandler
    (this, &Form1::richTextBox1_DragDrop);

```

To test the drag-and-drop functionality in your application

1. Save and build your application. While it is running, run WordPad.

WordPad is a text editor installed by Windows that allows drag-and-drop operations. It is accessible by clicking the **Start** button, selecting **Run**, typing **WordPad** in the text box of the **Run** dialog box, and then clicking **OK**.

2. Once WordPad is open, type a string of text in it. Using the mouse, select the text, and then drag the selected text over to the **RichTextBox** control in your Windows application.

Notice that when you point the mouse at the **RichTextBox** control (and, consequently, raise the **DragEnter** event), the mouse pointer changes and you can drop the selected text into the **RichTextBox** control.

When you release the mouse button, the selected text is dropped (that is, the **DragDrop** event is raised) and is inserted within the **RichTextBox** control.

See Also

[RichTextBox](#)

[How to: Perform Drag-and-Drop Operations Between Applications](#)

[RichTextBox Control](#)

[Controls to Use on Windows Forms](#)

How to: Load Files into the Windows Forms RichTextBox Control

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [RichTextBox](#) control can display a plain-text, Unicode plain-text, or Rich-Text-Format (RTF) file. To do so, call the [LoadFile](#) method. You can also use the [LoadFile](#) method to load data from a stream. For more information, see [LoadFile\(Stream, RichTextBoxStreamType\)](#).

To load a file into the RichTextBox control

1. Determine the path of the file to be opened using the [OpenFileDialog](#) component. For an overview, see [OpenFileDialog Component Overview](#).
2. Call the [LoadFile](#) method of the [RichTextBox](#) control, specifying the file to load and optionally a file type. In the example below, the file to load is taken from the [OpenFileDialog](#) component's [FileName](#) property. If you call the method with a file name as its only argument, the file type will be assumed to be RTF. To specify another file type, call the method with a value of the [RichTextBoxStreamType](#) enumeration as its second argument.

In the example below, the [OpenFileDialog](#) component is shown when a button is clicked. The file selected is then opened and displayed in the [RichTextBox](#) control. This example assumes a form has a button,

`btnOpenFile` .

```
Private Sub btnOpenFile_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles btnOpenFile.Click  
    If OpenFileDialog1.ShowDialog() = DialogResult.OK Then  
        RichTextBox1.LoadFile(OpenFileDialog1.FileName, _  
            RichTextBoxStreamType.RichText)  
    End If  
End Sub
```

```
private void btnOpenFile_Click(object sender, System.EventArgs e)  
{  
    if(openFileDialog1.ShowDialog() == DialogResult.OK)  
    {  
        richTextBox1.LoadFile(openFileDialog1.FileName, RichTextBoxStreamType.RichText);  
    }  
}
```

```
private:  
    void btnOpenFile_Click(System::Object ^  sender,  
        System::EventArgs ^  e)  
    {  
        if(openFileDialog1->ShowDialog() == DialogResult::OK)  
        {  
            richTextBox1->LoadFile(openFileDialog1->FileName,  
                RichTextBoxStreamType::RichText);  
        }  
    }  
}
```

(Visual C#, Visual C++) Place the following code in the form's constructor to register the event handler.

```
this.btnOpenFile.Click += new System.EventHandler(this.btnOpenFile_Click);
```

```
this->btnOpenFile->Click += gcnew  
System::EventHandler(this, &Form1::btnOpenFile_Click);
```

IMPORTANT

To run this process, your assembly may require a privilege level granted by the [System.Security.Permissions.FileIOPermission](#) class. If you are running in a partial-trust context, the process might throw an exception because of insufficient privileges. For more information, see [Code Access Security Basics](#).

See Also

[RichTextBox.LoadFile](#)

[RichTextBox](#)

[RichTextBox Control](#)

[Controls to Use on Windows Forms](#)

How to: Save Files with the Windows Forms RichTextBox Control

5/4/2018 • 2 min to read • [Edit Online](#)

The Windows Forms [RichTextBox](#) control can write the information it displays in one of several formats:

- Plain text
- Unicode plain text
- Rich-Text Format (RTF)
- RTF with spaces in place of OLE objects
- Plain text with a textual representation of OLE objects

To save a file, call the [SaveFile](#) method. You can also use the [SaveFile](#) method to save data to a stream. For more information, see [SaveFile\(Stream, RichTextBoxStreamType\)](#).

To save the contents of the control to a file

1. Determine the path of the file to be saved.

To do this in a real-world application, you would typically use the [SaveFileDialog](#) component. For an overview, see [SaveFileDialog Component Overview](#).

2. Call the [SaveFile](#) method of the [RichTextBox](#) control, specifying the file to save and optionally a file type. If you call the method with a file name as its only argument, the file will be saved as RTF. To specify another file type, call the method with a value of the [RichTextBoxStreamType](#) enumeration as its second argument.

In the example below, the path set for the location of the rich-text file is the **My Documents** folder. This location is used because you can assume that most computers running the Windows operating system will include this folder. Choosing this location also allows users with minimal system access levels to safely run the application. The example below assumes a form with a [RichTextBox](#) control already added.

```
Public Sub SaveFile()
    ' You should replace the bold file name in the
    ' sample below with a file name of your own choosing.
    RichTextBox1.SaveFile(System.Environment.GetFolderPath _
        (System.Environment.SpecialFolder.Personal) _
        & "\Testdoc.rtf", _
        RichTextBoxStreamType.RichNoOleObjs)
End Sub
```

```
public void SaveFile()
{
    // You should replace the bold file name in the
    // sample below with a file name of your own choosing.
    // Note the escape character used (@) when specifying the path.
    richTextBox1.SaveFile(System.Environment.GetFolderPath
        (System.Environment.SpecialFolder.Personal)
        + @"\Testdoc.rtf",
        RichTextBoxStreamType.RichNoOleObjs);
}
```

```
public:  
    void SaveFile()  
    {  
        // You should replace the bold file name in the  
        // sample below with a file name of your own choosing.  
        richTextBox1->SaveFile(String::Concat  
            (System::Environment::GetFolderPath  
            (System::Environment::SpecialFolder::Personal),  
            "\\\Testdoc.rtf"), RichTextBoxStreamType::RichNoOleObjs);  
    }
```

IMPORTANT

This example creates a new file, if the file does not already exist. If an application needs to create a file, that application needs Create access for the folder. Permissions are set using access control lists. If the file already exists, the application needs only Write access, a lesser privilege. Where possible, it is more secure to create the file during deployment, and only grant Read access to a single file, rather than Create access for a folder. Also, it is more secure to write data to user folders than to the root folder or the Program Files folder.

See Also

[RichTextBox.SaveFile](#)

[RichTextBox](#)

[RichTextBox Control](#)

[Controls to Use on Windows Forms](#)

How to: Set Font Attributes for the Windows Forms RichTextBox Control

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [RichTextBox](#) control has numerous options for formatting the text it displays. You can make the selected characters bold, underlined, or italic, using the [SelectionFont](#) property. You can also use this property to change the size and typeface of the selected characters. The [SelectionColor](#) property enables you to change the selected characters' color.

To change the appearance of characters

1. Set the [SelectionFont](#) property to an appropriate font.

To enable users to set the font family, size, and typeface in an application, you would typically use the [FontDialog](#) component. For an overview, see [FontDialog Component Overview](#).

2. Set the [SelectionColor](#) property to an appropriate color.

To enable users to set the color in an application, you would typically use the [ColorDialog](#) component. For an overview, see [ColorDialog Component Overview](#).

```
RichTextBox1.SelectionFont = New Font("Tahoma", 12, FontStyle.Bold)
RichTextBox1.SelectionColor = System.Drawing.Color.Red
```

```
richTextBox1.SelectionFont = new Font("Tahoma", 12, FontStyle.Bold);
richTextBox1.SelectionColor = System.Drawing.Color.Red;
```

```
richTextBox1->SelectionFont =
    gcnew System::Drawing::Font("Tahoma", 12, FontStyle::Bold);
richTextBox1->SelectionColor = System::Drawing::Color::Red;
```

NOTE

These properties only affect selected text, or, if no text is selected, the text that is typed at the current location of the insertion point. For information on selecting text programmatically, see [Select](#).

See Also

[RichTextBox](#)

[RichTextBox Control](#)

[Controls to Use on Windows Forms](#)

How to: Set Indents, Hanging Indents, and Bulleted Paragraphs with the Windows Forms RichTextBox Control

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [RichTextBox](#) control has numerous options for formatting the text it displays. You can format selected paragraphs as bulleted lists by setting the [SelectionBullet](#) property. You can also use the [SelectionIndent](#), [SelectionRightIndent](#), and [SelectionHangingIndent](#) properties to set the indentation of paragraphs relative to the left and right edges of the control, and the left edge of other lines of text.

To format a paragraph as a bulleted list

1. Set the [SelectionBullet](#) property to `true`.

```
RichTextBox1.SelectionBullet = True
```

```
richTextBox1.SelectionBullet = true;
```

```
richTextBox1->SelectionBullet = true;
```

To indent a paragraph

1. Set the [SelectionIndent](#) property to an integer representing the distance in pixels between the left edge of the control and the left edge of the text.
2. Set the [SelectionHangingIndent](#) property to an integer representing the distance in pixels between the left edge of the first line of text in the paragraph and the left edge of subsequent lines in the same paragraph. The value of the [SelectionHangingIndent](#) property only applies to lines in a paragraph that have wrapped below the first line.
3. Set the [SelectionRightIndent](#) property to an integer representing the distance in pixels between the right edge of the control and the right edge of the text.

```
RichTextBox1.SelectionIndent = 8  
RichTextBox1.SelectionHangingIndent = 3  
RichTextBox1.SelectionRightIndent = 12
```

```
richTextBox1.SelectionIndent = 8;  
richTextBox1.SelectionHangingIndent = 3;  
richTextBox1.SelectionRightIndent = 12;
```

```
richTextBox1->SelectionIndent = 8;  
richTextBox1->SelectionHangingIndent = 3;  
richTextBox1->SelectionRightIndent = 12;
```

NOTE

All these properties affect any paragraphs that contain selected text, and also the text that is typed after the current insertion point. For example, when a user selects a word within a paragraph and then adjusts the indentation, the new settings will apply to the entire paragraph that contains that word, and also to any paragraphs subsequently entered after the selected paragraph. For information about selecting text programmatically, see [Select](#).

See Also

[RichTextBox](#)

[RichTextBox Control](#)

[Controls to Use on Windows Forms](#)

SaveFileDialog Component (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [SaveFileDialog](#) component is a pre-configured dialog box. It is the same as the standard Save File dialog box used by Windows. It inherits from the [CommonDialog](#) class.

In This Section

[SaveFileDialog Component Overview](#)

Introduces the general concepts of the [SaveFileDialog](#) component, which allows you to display a pre-configured dialog that users can use to save a file to a specified location.

[How to: Save Files Using the SaveFileDialog Component](#)

Explains how to save a file via an instance of the [SaveFileDialog](#) component at run time.

Reference

[SaveFileDialog](#) class

Provides reference information on the class and its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

[Dialog-Box Controls and Components](#)

Describes a set of controls and components that allow users to perform standard interactions with the application or system.

[Essential Code for Windows Forms Dialog Boxes](#)

Discusses the Windows Forms dialog box controls and components and the code necessary for executing their basic functions. (MSDN Online Library technical article)

SaveFileDialog Component Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [SaveFileDialog](#) component is a pre-configured dialog box. It is the same as the standard **Save File** dialog box used by Windows. It inherits from the [CommonDialog](#) class.

Working with the SaveFileDialog Component

Use it as a simple solution for enabling users to save files instead of configuring your own dialog box. By relying on standard Windows dialog boxes, the basic functionality of applications you create is immediately familiar to users. Be aware, however, that when using the [SaveFileDialog](#) component, you must write your own file-saving logic.

You can use the [ShowDialog](#) method to display the dialog box at run time. You can open a file in read/write mode using the [OpenFile](#) method.

When it is added to a form, the [SaveFileDialog](#) component appears in the tray at the bottom of the Windows Forms Designer.

See Also

[SaveFileDialog](#)
[SaveFileDialog Component](#)

How to: Save Files Using the SaveFileDialog Component

5/4/2018 • 3 min to read • [Edit Online](#)

The [SaveFileDialog](#) component allows users to browse the file system and select files to be saved. The dialog box returns the path and name of the file the user has selected in the dialog box. However, you must write the code to actually write the files to disk.

To save a file using the SaveFileDialog component

- Display the **Save File** dialog box and call a method to save the file selected by the user.

Use the [SaveFileDialog](#) component's [OpenFile](#) method to save the file. This method gives you a [Stream](#) object you can write to.

The example below uses the [DialogResult](#) property to get the name of the file, and the [OpenFile](#) method to save the file. The [OpenFile](#) method gives you a stream to write the file to.

In the example below, there is a [Button](#) control with an image assigned to it. When you click the button, a [SaveFileDialog](#) component is instantiated with a filter that allows files of type .gif, jpeg, and .bmp. If a file of this type is selected in the Save File dialog box, the button's image is saved.

IMPORTANT

To get or set the [FileName](#) property, your assembly requires a privilege level granted by the [System.Security.Permissions.FileIOPermission](#) class. If you are running in a partial-trust context, the process might throw an exception due to insufficient privileges. For more information, see [Code Access Security Basics](#).

The example assumes your form has a [Button](#) control with its [Image](#) property set to a file of type .gif, jpeg, or .bmp.

NOTE

The [FileDialog](#) class's [FilterIndex](#) property (which, due to inheritance, is part of the [SaveFileDialog](#) class) uses a one-based index. This is important if you are writing code to save data in a specific format (for example, saving a file in plain text versus binary format). This property is featured in the example below.

```
Private Sub Button2_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button2.Click
    ' Displays a SaveFileDialog so the user can save the Image
    ' assigned to Button2.
    Dim saveFileDialog1 As New SaveFileDialog()
    saveFileDialog1.Filter = "Jpeg Image|*.jpg|Bitmap Image|*.bmp|Gif Image|*.gif"
    saveFileDialog1.Title = "Save an Image File"
    saveFileDialog1.ShowDialog()

    ' If the file name is not an empty string open it for saving.
    If saveFileDialog1.FileName <> "" Then
        ' Saves the Image via a FileStream created by the OpenFile method.
        Dim fs As System.IO.FileStream = CType _
            (saveFileDialog1.OpenFile(), System.IO.FileStream)
        ' Saves the Image in the appropriate ImageFormat based upon the
        ' file type selected in the dialog box.
        ' NOTE that the FilterIndex property is one-based.
        Select Case saveFileDialog1.FilterIndex
            Case 1
                Me.button2.Image.Save(fs, _
                    System.Drawing.Imaging.ImageFormat.Jpeg)

            Case 2
                Me.button2.Image.Save(fs, _
                    System.Drawing.Imaging.ImageFormat.Bmp)

            Case 3
                Me.button2.Image.Save(fs, _
                    System.Drawing.Imaging.ImageFormat.Gif)
        End Select

        fs.Close()
    End If
End Sub
```

```
private void button2_Click(object sender, System.EventArgs e)
{
    // Displays a SaveFileDialog so the user can save the Image
    // assigned to Button2.
    SaveFileDialog saveFileDialog1 = new SaveFileDialog();
    saveFileDialog1.Filter = "Jpeg Image|*.jpg|Bitmap Image|*.bmp|Gif Image|*.gif";
    saveFileDialog1.Title = "Save an Image File";
    saveFileDialog1.ShowDialog();

    // If the file name is not an empty string open it for saving.
    if(saveFileDialog1.FileName != "")
    {
        // Saves the Image via a FileStream created by the OpenFile method.
        System.IO.FileStream fs =
            (System.IO.FileStream)saveFileDialog1.OpenFile();
        // Saves the Image in the appropriate ImageFormat based upon the
        // File type selected in the dialog box.
        // NOTE that the FilterIndex property is one-based.
        switch(saveFileDialog1.FilterIndex)
        {
            case 1 :
                this.button2.Image.Save(fs,
                    System.Drawing.Imaging.ImageFormat.Jpeg);
                break;

            case 2 :
                this.button2.Image.Save(fs,
                    System.Drawing.Imaging.ImageFormat.Bmp);
                break;

            case 3 :
                this.button2.Image.Save(fs,
                    System.Drawing.Imaging.ImageFormat.Gif);
                break;
        }

        fs.Close();
    }
}
```

```

private:
    System::Void button2_Click(System::Object ^ sender,
        System::EventArgs ^ e)
    {
        // Displays a SaveFileDialog so the user can save the Image
        // assigned to Button2.
        SaveFileDialog ^ saveFileDialog1 = new SaveFileDialog();
        saveFileDialog1->Filter =
            "Jpeg Image|*.jpg|Bitmap Image|*.bmp|Gif Image|*.gif";
        saveFileDialog1->Title = "Save an Image File";
        saveFileDialog1->>ShowDialog();
        // If the file name is not an empty string, open it for saving.
        if(saveFileDialog1->FileName != "")
        {
            // Saves the Image through a FileStream created by
            // the OpenFile method.
            System::IO::FileStream ^ fs =
                safe_cast<System::IO::FileStream*>(
                    saveFileDialog1->OpenFile());
            // Saves the Image in the appropriate ImageFormat based on
            // the file type selected in the dialog box.
            // Note that the FilterIndex property is one based.
            switch(saveFileDialog1->FilterIndex)
            {
                case 1 :
                    this->button2->Image->Save(fs,
                        System::Drawing::Imaging::ImageFormat::Jpeg);
                    break;
                case 2 :
                    this->button2->Image->Save(fs,
                        System::Drawing::Imaging::ImageFormat::Bmp);
                    break;
                case 3 :
                    this->button2->Image->Save(fs,
                        System::Drawing::Imaging::ImageFormat::Gif);
                    break;
            }
            fs->Close();
        }
    }
}

```

(Visual C# and Visual C++) Place the following code in the form's constructor to register the event handler.

```
this.button2.Click += new System.EventHandler(this.button2_Click);
```

```
this->button2->Click += gcnew
    System::EventHandler(this, &Form1::button2_Click);
```

For more information about writing file streams, see [BeginWrite](#) and [Write](#).

NOTE

Certain controls, such as the [RichTextBox](#) control, have the ability to save files. For more information, see the "SaveFileDialog Component" section of the MSDN Online Library technical article, [Essential Code for Windows Forms Dialog Boxes](#).

See Also

[SaveFileDialog](#)

SaveFileDialog Component

SoundPlayer Class

5/4/2018 • 1 min to read • [Edit Online](#)

The `SoundPlayer` class enables you to easily include sounds in your applications.

You can also use the `SystemSounds` class to play common system sounds, including a beep.

In This Section

[SoundPlayer Class Overview](#)

Introduces the class and its commonly used properties, methods, and events.

[How to: Play a Sound from a Windows Form](#)

Provides code to play a sound specified via a file path, UNC path, or HTTP path.

[How to: Play a Beep from a Windows Form](#)

Provides code to play the computer's beep sound.

[How to: Play a Sound Embedded in a Resource from a Windows Form](#)

Provides code to play a sound from a resource.

[How to: Play a System Sound from a Windows Form](#)

Provides code to play the one of the system sounds.

[How to: Load a Sound Asynchronously within a Windows Form](#)

Provides code to load a sound asynchronously from a URL and play it.

[How to: Loop a Sound Playing on a Windows Form](#)

Provides code that plays a sound repeatedly.

Reference

[SoundPlayer](#)

Describes this class and has links to all its members.

Related Sections

[Windows Forms Controls](#)

Provides links to topics about the controls designed specifically to work with Windows Forms.

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

Also see [HYPERLINK "http://msdn.microsoft.com/library/11bxex12\(v=vs.110\)"](http://msdn.microsoft.com/library/11bxex12(v=vs.110)) `My.Computer Object` or `My.Computer Object`.

SoundPlayer Class Overview

5/4/2018 • 1 min to read • [Edit Online](#)

The [SoundPlayer](#) class enables you to easily include sounds in your applications.

The [SoundPlayer](#) class can play sound files in the .wav format, either from a resource or from UNC or HTTP locations. Additionally, the [SoundPlayer](#) class enables you to load or play sounds asynchronously.

You can also use the [SystemSounds](#) class to play common system sounds, including a beep.

Commonly Used Properties, Methods, and Events

NAME	DESCRIPTION
SoundLocation property	The file path or Web address of the sound. Acceptable values can be UNC or HTTP.
LoadTimeout property	The number of milliseconds your program will wait to load a sound before it throws an exception. The default is 10 seconds.
IsLoadCompleted property	A Boolean value indicating whether the sound has finished loading.
Load method	Loads a sound synchronously.
LoadAsync method	Begins to load a sound asynchronously. When loading is complete, it raises the OnLoadCompleted event.
Play method	Plays the sound specified in the SoundLocation or Stream property in a new thread.
PlaySync method	Plays the sound specified in the SoundLocation or Stream property in the current thread.
Stop method	Stops any sound currently playing.
LoadCompleted event	Raised after the load of a sound is attempted.

See Also

[SoundPlayer](#)

[SystemSounds](#)

How to: Play a Sound from a Windows Form

5/4/2018 • 1 min to read • [Edit Online](#)

This example plays a sound at a given path at run time.

Example

```
Sub PlaySimpleSound()
    My.Computer.Audio.Play("c:\Windows\Media\chimes.wav")
End Sub
```

```
private void playSimpleSound()
{
    SoundPlayer simpleSound = new SoundPlayer(@"c:\Windows\Media\chimes.wav");
    simpleSound.Play();
}
```

Compiling the Code

This example requires:

- That you replace the file name `"c:\Windows\Media\chimes.wav"` with a valid file name.
- (C#) A reference to the [System.Media](#) namespace.

Robust Programming

File operations should be enclosed within appropriate structured exception handling blocks.

The following conditions may cause an exception:

- The path name is malformed. For example, it contains illegal characters or is only white space ([ArgumentException](#) class).
- The path is read-only ([IOException](#) class).
- The path name is `null` ([ArgumentNullException](#) class).
- The path name is too long ([PathTooLongException](#) class).
- The path is invalid ([DirectoryNotFoundException](#) class).
- The path is only a colon, ":" ([NotSupportedException](#) class).

.NET Framework Security

Do not make decisions about the contents of the file based on the name of the file. For example, the file `Form1.vb` may not be a Visual Basic source file. Verify all inputs before using the data in your application.

See Also

[SoundPlayer](#)

How to: Load a Sound Asynchronously within a Windows Form

How to: Play a Beep from a Windows Form

5/4/2018 • 1 min to read • [Edit Online](#)

This example plays a beep at run time.

Example

```
Public Sub OnePing()
    Beep()
End Sub
```

```
public void onePing()
{
    SystemSounds.Beep.Play();
}
```

NOTE

The sound played in the C# code sample is determined by the [Beep](#) system sound setting. For more information, see [SystemSounds](#).

Compiling the Code

For C#, this example requires a reference to the [System.Media](#) namespace.

See Also

[Beep](#)

[SoundPlayer](#)

[How to: Play a System Sound from a Windows Form](#)

[How to: Play a Sound from a Windows Form](#)

How to: Play a Sound Embedded in a Resource from a Windows Form

5/4/2018 • 1 min to read • [Edit Online](#)

You can use the [SoundPlayer](#) class to play a sound from an embedded resource.

Example

```
private void playSoundFromResource(object sender, EventArgs e)
{
    System.Reflection.Assembly a = System.Reflection.Assembly.GetExecutingAssembly();
    System.IO.Stream s = a.GetManifestResourceStream("<AssemblyName>.chimes.wav");
    SoundPlayer player = new SoundPlayer(s);
    player.Play();
}
```

```
Private Sub playSoundFromResource(ByVal sender As Object, _
ByVal e As EventArgs)
    Dim a As System.Reflection.Assembly = System.Reflection.Assembly.GetExecutingAssembly()
    Dim s As System.IO.Stream = a.GetManifestResourceStream("<AssemblyName>.chimes.wav")
    Dim player As SoundPlayer = New SoundPlayer(s)
    player.Play()
End Sub
```

Compiling the Code

This example requires:

Importing the [System.Media](#) namespace.

Including the sound file as an embedded resource in your project.

Replacing "<AssemblyName>" with the name of the assembly in which the sound file is embedded. Do not include the ".dll" suffix.

See Also

[SoundPlayer](#)

[How to: Play a Sound from a Windows Form](#)

[How to: Loop a Sound Playing on a Windows Form](#)

How to: Play a System Sound from a Windows Form

5/4/2018 • 1 min to read • [Edit Online](#)

The following code example plays the `Exclamation` system sound at run time. For more information about system sounds, see [SystemSounds](#).

Example

```
Public Sub PlayExclamation()
    SystemSounds.Exclamation.Play()
End Sub
```

```
public void playExclamation()
{
    SystemSounds.Exclamation.Play();
}
```

Compiling the Code

This example requires:

- A reference to the [System.Media](#) namespace.

See Also

[SoundPlayer](#)

[SystemSounds](#)

[How to: Play a Beep from a Windows Form](#)

[How to: Play a Sound from a Windows Form](#)

How to: Load a Sound Asynchronously within a Windows Form

5/4/2018 • 3 min to read • [Edit Online](#)

The following code example asynchronously loads a sound from an URL and then plays it on a new thread.

Example

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Media;
using System.Windows.Forms;

namespace SoundPlayerLoadAsyncExample
{
    public class Form1 : Form
    {
        private SoundPlayer Player = new SoundPlayer();

        public Form1()
        {
            InitializeComponent();

            this.Player.LoadCompleted += new AsyncCompletedEventHandler(Player_LoadCompleted);
        }

        private void playSoundButton_Click(object sender, EventArgs e)
        {
            this.LoadAsyncSound();
        }

        public void LoadAsyncSound()
        {
            try
            {
                // Replace this file name with a valid file name.
                this.Player.SoundLocation = "http://www.tailspintoyos.com/sounds/stop.wav";
                this.Player.LoadAsync();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Error loading sound");
            }
        }

        // This is the event handler for the LoadCompleted event.
        void Player_LoadCompleted(object sender, AsyncCompletedEventArgs e)
        {
            if (Player.IsLoadCompleted)
            {
                try
                {
                    this.Player.Play();
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message, "Error playing sound");
                }
            }
        }
    }
}
```

```

        }

    }

    private Button playSoundButton;

    /// <summary>
    /// Required designer variable.
    /// </summary>
    private System.ComponentModel.IContainer components = null;

    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
    protected override void Dispose(bool disposing)
    {
        if (disposing && (components != null))
        {
            components.Dispose();
        }
        base.Dispose(disposing);
    }

    #region Windows Form Designer generated code

    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
        this.playSoundButton = new System.Windows.Forms.Button();
        this.SuspendLayout();
        //
        // playSoundButton
        //
        this.playSoundButton.Location = new System.Drawing.Point(106, 112);
        this.playSoundButton.Name = "playSoundButton";
        this.playSoundButton.Size = new System.Drawing.Size(75, 23);
        this.playSoundButton.TabIndex = 0;
        this.playSoundButton.Text = "Play Sound";
        this.playSoundButton.Click += new System.EventHandler(this.playSoundButton_Click);
        //
        // Form1
        //
        this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.ClientSize = new System.Drawing.Size(292, 273);
        this.Controls.Add(this.playSoundButton);
        this.Name = "Form1";
        this.Text = "Form1";
        this.ResumeLayout(false);

    }

    #endregion

}

static class Program
{
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.Run(new Form1());
    }
}

```

```
        }
    }
}
```

```
Imports System
Imports System.Media
Imports System.Windows.Forms

Public Class Form1
    Inherits System.Windows.Forms.Form

    Friend WithEvents playSoundButton As System.Windows.Forms.Button
    Private WithEvents Player As New SoundPlayer

    Sub New()

        Me.InitializeComponent()

    End Sub

    Private Sub playSoundButton_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles playSoundButton.Click

        ' Replace this file name with a valid file name.
        Me.Player.SoundLocation = "http://www.tailspintoyos.com/sounds/stop.wav"
        Me.Player.LoadAsync()

    End Sub

    Private Sub Player_LoadCompleted( _
        ByVal sender As Object, _
        ByVal e As _
        System.ComponentModel.AsyncCompletedEventArgs) _
        Handles Player.LoadCompleted

        If Me.Player.IsLoadCompleted Then
            Me.Player.Play()
        End If

    End Sub

    'Form overrides dispose to clean up the component list.
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing AndAlso components IsNot Nothing Then
            components.Dispose()
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        Me.playSoundButton = New System.Windows.Forms.Button
        Me.SuspendLayout()
        '
        'playSoundButton
        '
        Me.playSoundButton.Location = New System.Drawing.Point(105, 107)
        Me.playSoundButton.Name = "playSoundButton"
    End Sub
```

```

        Me.playSoundButton.Size = New System.Drawing.Size(75, 23)
        Me.playSoundButton.TabIndex = 0
        Me.playSoundButton.Text = "Play Sound"
        Me.playSoundButton.UseVisualStyleBackColor = True

        '
        'Form1

        Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)
        Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
        Me.ClientSize = New System.Drawing.Size(292, 273)
        Me.Controls.Add(Me.playSoundButton)
        Me.Name = "Form1"
        Me.Text = "Form1"
        Me.ResumeLayout(False)

    End Sub

    <STAThread()> _
    Shared Sub Main()
        Application.EnableVisualStyles()
        Application.Run(New Form1())
    End Sub

End Class

```

Compiling the Code

This example requires:

- References to the System and System.Windows.Forms assemblies.
- That you replace the file name `"http://www.tailspintoyos.com/sounds/stop.wav"` with a valid file name.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

Robust Programming

File operations should be enclosed within appropriate exception-handling blocks.

The following conditions may cause an exception:

- The path name is malformed. For example, it contains characters that are not valid or is only white space ([ArgumentException](#) class).
- The path is read-only ([IOException](#) class).
- The path name is `Nothing` ([ArgumentNullException](#) class).
- The path name is too long ([PathTooLongException](#) class).
- The path is not valid ([DirectoryNotFoundException](#) class).
- The path is only a colon ":" ([NotSupportedException](#) class).

.NET Framework Security

Do not make decisions about the contents of the file based on the name of the file. For example, the file `Form1.vb` may not be a Visual Basic source file. Verify all inputs before using the data in your application.

See Also

[LoadAsync](#)

[LoadCompleted](#)

[Play](#)

[How to: Play a Sound from a Windows Form](#)

How to: Loop a Sound Playing on a Windows Form

5/4/2018 • 4 min to read • [Edit Online](#)

The following code example plays a sound repeatedly. When the code in the `stopPlayingButton_Click` event handler runs, any sound currently playing stops. If no sound is playing, nothing happens.

Example

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Media;
using System.Windows.Forms;

namespace SoundPlayerPlayLoopingExample
{
    public class Form1 : Form
    {
        private SoundPlayer Player = new SoundPlayer();

        public Form1()
        {
            InitializeComponent();
        }

        private void playLoopingButton_Click(object sender, EventArgs e)
        {
            try
            {
                // Note: You may need to change the location specified based on
                // the sounds loaded on your computer.
                this.Player.SoundLocation = @"C:\Windows\Media\chimes.wav";
                this.Player.PlayLooping();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Error playing sound");
            }
        }

        private void stopPlayingButton_Click(object sender, EventArgs e)
        {
            this.Player.Stop();
        }

        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
        }
    }
}
```

```

        base.Dispose(disposing);
    }

#region Windows Form Designer generated code

    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
        this.playLoopingButton = new System.Windows.Forms.Button();
        this.stopPlayingButton = new System.Windows.Forms.Button();
        this.SuspendLayout();
        //
        // playLoopingButton
        //
        this.playLoopingButton.Location = new System.Drawing.Point(12, 12);
        this.playLoopingButton.Name = "playLoopingButton";
        this.playLoopingButton.Size = new System.Drawing.Size(87, 23);
        this.playLoopingButton.TabIndex = 0;
        this.playLoopingButton.Text = "Play Looping";
        this.playLoopingButton.UseVisualStyleBackColor = true;
        this.playLoopingButton.Click += new System.EventHandler(this.playLoopingButton_Click);
        //
        // stopPlayingButton
        //
        this.stopPlayingButton.Location = new System.Drawing.Point(105, 12);
        this.stopPlayingButton.Name = "stopPlayingButton";
        this.stopPlayingButton.Size = new System.Drawing.Size(75, 23);
        this.stopPlayingButton.TabIndex = 1;
        this.stopPlayingButton.Text = "Stop";
        this.stopPlayingButton.UseVisualStyleBackColor = true;
        this.stopPlayingButton.Click += new System.EventHandler(this.stopPlayingButton_Click);
        //
        // Form1
        //
        this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.ClientSize = new System.Drawing.Size(195, 51);
        this.Controls.Add(this.stopPlayingButton);
        this.Controls.Add(this.playLoopingButton);
        this.Name = "Form1";
        this.Text = "Form1";
        this.ResumeLayout(false);

    }

#endregion

    private System.Windows.Forms.Button playLoopingButton;
    private System.Windows.Forms.Button stopPlayingButton;
}

static class Program
{
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.Run(new Form1());
    }
}
}

```

```

Imports System
Imports System.Media
Imports System.Windows.Forms

Public Class Form1
    Inherits System.Windows.Forms.Form

    Private Player As New SoundPlayer

    Sub New()
        Me.InitializeComponent()
    End Sub

    Private Sub playLoopingButton_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles playLoopingButton.Click

        Try
            ' Note: You may need to change the location specified based on
            ' the sounds loaded on your computer.
            Me.Player.SoundLocation = "C:\Windows\Media\chimes.wav"
            Me.Player.PlayLooping()
        Catch ex As Exception
            MessageBox.Show(ex.Message, "Error playing sound")
        End Try
    End Sub

    Private Sub stopPlayingButton_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles stopPlayingButton.Click

        Me.Player.Stop()
    End Sub

    'Form overrides dispose to clean up the component list.
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing AndAlso components IsNot Nothing Then
            components.Dispose()
        End If
        MyBase.Dispose(disposing)
    End Sub

    Friend WithEvents playLoopingButton As System.Windows.Forms.Button
    Friend WithEvents stopPlayingButton As System.Windows.Forms.Button

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        Me.playLoopingButton = New System.Windows.Forms.Button
        Me.stopPlayingButton = New System.Windows.Forms.Button
        Me.SuspendLayout()
        '
        'playLoopingButton
        '
        Me.playLoopingButton.Location = New System.Drawing.Point(12, 12)
        Me.playLoopingButton.Name = "playLoopingButton"

```

```

        Me.playLoopingButton.Size = New System.Drawing.Size(89, 23)
        Me.playLoopingButton.TabIndex = 0
        Me.playLoopingButton.Text = "Play Looping"
        Me.playLoopingButton.UseVisualStyleBackColor = True
        '
        'stopPlayingButton
        '

        Me.stopPlayingButton.Location = New System.Drawing.Point(107, 12)
        Me.stopPlayingButton.Name = "stopPlayingButton"
        Me.stopPlayingButton.Size = New System.Drawing.Size(75, 23)
        Me.stopPlayingButton.TabIndex = 1
        Me.stopPlayingButton.Text = "Stop"
        Me.stopPlayingButton.UseVisualStyleBackColor = True
        '
        'Form1
        '

        Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)
        Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
        Me.ClientSize = New System.Drawing.Size(197, 52)
        Me.Controls.Add(Me.stopPlayingButton)
        Me.Controls.Add(Me.playLoopingButton)
        Me.Name = "Form1"
        Me.Text = "Form1"
        Me.ResumeLayout(False)

    End Sub

    <STAThread()> _
    Shared Sub Main()
        Application.EnableVisualStyles()
        Application.Run(New Form1())
    End Sub

End Class

```

Compiling the Code

This example requires:

- References to the System and System.Windows.Forms assemblies.
- That you replace the file name `"c:\Windows\Media\chimes.wav"` with a valid file name.

For information about building this example from the command line for visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

Robust Programming

File operations should be enclosed within appropriate exception-handling blocks.

The following conditions may cause an exception:

- The path name is malformed. For example, it contains characters that are not valid or it is only white space ([ArgumentException](#) class).
- The path is read-only ([IOException](#) class).
- The path name is `Nothing` ([ArgumentNullException](#) class).
- The path name is too long ([PathTooLongException](#) class).

- The path is invalid ([DirectoryNotFoundException](#) class).
- The path is only a colon ":" ([NotSupportedException](#) class).

.NET Framework Security

Do not make decisions about the contents of the file based on the name of the file. For example, the file Form1.vb may not be a Visual Basic source file. Verify all inputs before using the data in your application.

See Also

[PlayLooping](#)

[How to: Play a Sound from a Windows Form](#)

[SoundPlayer Class Overview](#)

SplitContainer Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `SplitContainer` control can be thought of as a composite; it is two panels separated by a movable bar. When the mouse pointer is over the bar, the pointer changes shape to show that the bar is movable.

NOTE

In the **Toolbox**, this control replaces the `Splitter` control that was there in the previous version of Visual Studio. The `SplitContainer` control is much preferred over the `Splitter` control. The `Splitter` class is still included in the .NET Framework for compatibility with existing applications, but we strongly encourage you to use the `SplitContainer` control for new projects.

The `SplitContainer` control lets you create complex user interfaces; often, a selection in one panel determines what objects are shown in the other panel. This arrangement is very effective for displaying and browsing information. Having two panels allow you to aggregate information in areas, and the bar, or "splitter," makes it easy for users to resize the panels.

In This Section

[SplitContainer Control Overview](#)

Introduces the `SplitContainer` control and describes the commonly used properties, methods, and events.

[How to: Define Resize and Positioning Behavior in a Split Window](#)

Describes how to control the splitter within the `SplitContainer` control.

[How to: Split a Window Horizontally](#)

Describes how to control the orientation of the splitter within the `SplitContainer` control.

[How to: Create a Multipane User Interface with Windows Forms](#)

Creates a multi-pane user interface that is similar to the one used in Microsoft Outlook.

Also see [How to: Split a Window Horizontally Using the Designer](#), [How to: Create a Windows Explorer–Style Interface on a Windows Form](#), [How to: Create a Multipane User Interface with Windows Forms Using the Designer](#).

Reference

[SplitContainer class](#)

Describes this class and has links to all its members.

Related Sections

[Windows Forms Controls](#)

Provides links to topics about the controls designed specifically to work with Windows Forms.

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

SplitContainer Control Overview (Windows Forms)

5/4/2018 • 2 min to read • [Edit Online](#)

The Windows Forms [SplitContainer](#) control can be thought of as a composite; it is two panels separated by a movable bar. When the mouse pointer is over the bar, the pointer changes shape to show that the bar is movable.

IMPORTANT

In the [Toolbox](#), [SplitContainer](#) control replaces the [Splitter](#) control that was there in the previous version of Visual Studio. The [SplitContainer](#) control is much preferred over the [Splitter](#) control. The [Splitter](#) class is still included in the .NET Framework for compatibility with existing applications, but we strongly encourage you to use the [SplitContainer](#) control for new projects.

With the [SplitContainer](#) control, you can create complex user interfaces; often, a selection in one panel determines what objects are shown in the other panel. This arrangement is very effective for displaying and browsing information. Having two panels lets you aggregate information in areas, and the bar, or "splitter," makes it easy for users to resize the panels.

More than one [SplitContainer](#) control can also be nested, with the second [SplitContainer](#) control oriented horizontally, to create top and bottom panels.

Be aware that the [SplitContainer](#) control is keyboard-accessible by default; users can press the ARROW keys to move the splitter if the [IsSplitterFixed](#) property is set to `false`.

The [Orientation](#) property of the [SplitContainer](#) control determines the direction of the splitter, not of the control itself. Hence, when this property is set to [Vertical](#), the splitter runs from top to bottom, creating left and right panels.

Additionally, be aware that the value of the [SplitterRectangle](#) property varies depending on the value of the [Orientation](#) property. For more information, see [SplitterRectangle](#) property.

You can also restrict the size and movement of the [SplitContainer](#) control. The [FixedPanel](#) property determines which panel will remain the same size after the [SplitContainer](#) control is resized, and the [IsSplitterFixed](#) property determines if the splitter is movable by the keyboard or mouse.

NOTE

Even if the [IsSplitterFixed](#) property is set to `true`, the splitter may still be moved programmatically; for example, by using the [SplitterDistance](#) property.

Finally, each panel of the [SplitContainer](#) control has properties to determine its individual size.

Commonly Used Properties, Methods, and Events

NAME	DESCRIPTION
FixedPanel property	Determines which panel will remain the same size after the SplitContainer control is resized.
IsSplitterFixed property	Determines if the splitter can be moved with the keyboard or mouse.

NAME	DESCRIPTION
Orientation property	Determines if the splitter is arranged vertically or horizontally.
SplitterDistance property	Determines the distance in pixels from the left or upper edge to the movable splitter bar.
SplitterIncrement property	Determines the minimum distance, in pixels, that the splitter can be moved by the user.
SplitterWidth property	Determines the thickness, in pixels, of the splitter.
SplitterMoving event	Occurs when the splitter is moving.
SplitterMoved event	Occurs when the splitter has moved.

See Also

[SplitContainer](#)
[SplitContainer Control](#)
[SplitContainer Control Sample](#)

How to: Create a Multipane User Interface with Windows Forms

5/4/2018 • 3 min to read • [Edit Online](#)

In the following procedure, you will create a multipane user interface that is similar to the one used in Microsoft Outlook, with a **Folder** list, a **Messages** pane, and a **Preview** pane. This arrangement is achieved chiefly through docking controls with the form.

When you dock a control, you determine which edge of the parent container a control is fastened to. Thus, if you set the **Dock** property to **Right**, the right edge of the control will be docked to the right edge of its parent control. Additionally, the docked edge of the control is resized to match that of its container control. For more information about how the **Dock** property works, see [How to: Dock Controls on Windows Forms](#).

This procedure focuses on arranging the **SplitContainer** and the other controls on the form, not on adding functionality to make the application mimic Microsoft Outlook.

To create this user interface, you place all the controls within a **SplitContainer** control, which contains a **TreeView** control in the left-hand panel. The right-hand panel of the **SplitContainer** control contains a second **SplitContainer** control with a **ListView** control above a **RichTextBox** control. These **SplitContainer** controls enable independent resizing of the other controls on the form. You can adapt the techniques in this procedure to craft custom user interfaces of your own.

To create an Outlook-style user interface programmatically

1. Within a form, declare each control that comprises your user interface. For this example, use the **TreeView**, **ListView**, **SplitContainer**, and **RichTextBox** controls to mimic the Microsoft Outlook user interface.

```
Private WithEvents treeView1 As System.Windows.Forms.TreeView
Private WithEvents listView1 As System.Windows.Forms.ListView
Private WithEvents richTextBox1 As System.Windows.Forms.RichTextBox
Private WithEvents splitContainer1 As _
    System.Windows.Forms.SplitContainer
Private WithEvents splitContainer2 As _
    System.Windows.Forms.SplitContainer
```

```
private System.Windows.Forms.TreeView treeView1;
private System.Windows.Forms.ListView listView1;
private System.Windows.Forms.RichTextBox richTextBox1;
private System.Windows.Forms.SplitContainer splitContainer2;
private System.Windows.Forms.SplitContainer splitContainer1;
```

2. Create a procedure that defines your user interface. The following code sets the properties so that the form will resemble the user interface in Microsoft Outlook. However, by using other controls or docking them differently, it is just as easy to create other user interfaces that are equally flexible.

```

Public Sub CreateOutlookUI()
    ' Create an instance of each control being used.
    Me.components = New System.ComponentModel.Container()
    Me.treeView1 = New System.Windows.Forms.TreeView()
    Me.listView1 = New System.Windows.Forms.ListView()
    Me.richTextBox1 = New System.Windows.Forms.RichTextBox()
    Me.splitContainer1 = New System.Windows.Forms.SplitContainer()
    Me.splitContainer2= New System.Windows.Forms.SplitContainer()

    ' Should you develop this into a working application,
    ' use the AddHandler method to hook up event procedures here.

    ' Set properties of TreeView control.
    ' Use the With statement to avoid repetitive code.
    With Me.treeView1
        .Dock = System.Windows.Forms.DockStyle.Fill
        .TabIndex = 0
        .Nodes.Add("treeView")
    End With

    ' Set properties of ListView control.
    With Me.listView1
        .Dock = System.Windows.Forms.DockStyle.Top
        .TabIndex = 2
        .Items.Add("listView")
    End With

    ' Set properties of RichTextBox control.
    With Me.richTextBox1
        .Dock = System.Windows.Forms.DockStyle.Fill
        .TabIndex = 3
        .Text = "richTextBox1"
    End With

    ' Set properties of the first SplitContainer control.
    With Me.splitContainer1
        .Dock = System.Windows.Forms.DockStyle.Fill
        .TabIndex = 1
        .SplitterWidth = 4
        .SplitterDistance = 150
        .Orientation = Orientation.Horizontal
        .Panel1.Controls.Add(Me.listView1)
        .Panel2.Controls.Add(Me.richTextBox1)
    End With

    ' Set properties of the second SplitContainer control.
    With Me.splitContainer2
        .Dock = System.Windows.Forms.DockStyle.Fill
        .TabIndex = 4
        .SplitterWidth = 4
        .SplitterDistance = 100
        .Panel1.Controls.Add(Me.treeView1)
        .Panel2.Controls.Add(Me.splitContainer1)
    End With

    ' Add the main SplitContainer control to the form.
    Me.Controls.Add(Me.splitContainer2)
    Me.Text = "Intricate UI Example"
End Sub

```

```

public void createOutlookUI()
{
    // Create an instance of each control being used.
    treeView1 = new System.Windows.Forms.TreeView();
    listView1 = new System.Windows.Forms.ListView();
    richTextBox1 = new System.Windows.Forms.RichTextBox();
    splitContainer2 = new System.Windows.Forms.SplitContainer();
    splitContainer1 = new System.Windows.Forms.SplitContainer();

    // Insert code here to hook up event methods.

    // Set properties of TreeView control.
    treeView1.Dock = System.Windows.Forms.DockStyle.Fill;
    treeView1.TabIndex = 0;
    treeView1.Nodes.Add("treeView");

    // Set properties of ListView control.
    listView1.Dock = System.Windows.Forms.DockStyle.Top;
    listView1.TabIndex = 2;
    listView1.Items.Add("listView");

    // Set properties of RichTextBox control.
    richTextBox1.Dock = System.Windows.Forms.DockStyle.Fill;
    richTextBox1.TabIndex = 3;
    richTextBox1.Text = "richTextBox1";

    // Set properties of first SplitContainer control.
    splitContainer1.Dock = System.Windows.Forms.DockStyle.Fill;
    splitContainer2.TabIndex = 1;
    splitContainer2.SplitterWidth = 4;
    splitContainer2.SplitterDistance = 150;
    splitContainer2.Orientation = Orientation.Horizontal;
    splitContainer2.Panel1.Controls.Add(this.listView1);
    splitContainer2.Panel1.Controls.Add(this.richTextBox1);

    // Set properties of second SplitContainer control.
    splitContainer2.Dock = System.Windows.Forms.DockStyle.Fill;
    splitContainer2.TabIndex = 4;
    splitContainer2.SplitterWidth = 4;
    splitContainer2.SplitterDistance = 100;
    splitContainer2.Panel1.Controls.Add(this.treeView1);
    splitContainer2.Panel1.Controls.Add(this.splitContainer1);

    // Add the main SplitContainer control to the form.
    this.Controls.Add(this.splitContainer2);
    this.Text = "Intricate UI Example";
}

```

3. In Visual Basic, add a call to the procedure you just created in the `New()` procedure. In Visual C#, add this line of code to the constructor for the form class.

```

' Add this to the New procedure.
CreateOutlookUI()

```

```

// Add this to the form class's constructor.
createOutlookUI();

```

See Also

[SplitContainer](#)

[SplitContainer Control](#)

How to: Create a Multipane User Interface with Windows Forms Using the Designer

How to: Create a Multipane User Interface with Windows Forms Using the Designer

5/4/2018 • 2 min to read • [Edit Online](#)

In the following procedure, you will create a multipane user interface that is similar to the one used in Microsoft Outlook, with a **Folder** list, a **Messages** pane, and a **Preview** pane. This arrangement is achieved chiefly through docking controls with the form.

When you dock a control, you determine which edge of the parent container a control is fastened to. Thus, if you set the **Dock** property to **Right**, the right edge of the control will be docked to the right edge of its parent control. Additionally, the docked edge of the control is resized to match that of its container control. For more information about how the **Dock** property works, see [How to: Dock Controls on Windows Forms](#).

This procedure focuses on arranging the **SplitContainer** and the other controls on the form, not on adding functionality to make the application mimic Microsoft Outlook.

To create this user interface, you place all the controls within a **SplitContainer** control, which contains a **TreeView** control in the left-hand panel. The right-hand panel of the **SplitContainer** control contains a second **SplitContainer** control with a **ListView** control above a **RichTextBox** control. These **SplitContainer** controls enable independent resizing of the other controls on the form. You can adapt the techniques in this procedure to craft custom user interfaces of your own.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To create an Outlook-style user interface at design time

1. Create a new Windows Application project. For details, see [How to: Create a Windows Application Project](#).
2. Drag a **SplitContainer** control from the **Toolbox** to the form. In the **Properties** window, set the **Dock** property to **Fill**.
3. Drag a **TreeView** control from the **Toolbox** to the left-hand panel of the **SplitContainer** control. In the **Properties** window, set the **Dock** property to **Left** by clicking the left hand panel in the value editor shown when the down arrow is clicked.
4. Drag another **SplitContainer** control from the **Toolbox**; place it in the right-hand panel of the **SplitContainer** control you added to your form. In the **Properties** window, set the **Dock** property to **Fill** and the **Orientation** property to **Horizontal**.
5. Drag a **ListView** control from the **Toolbox** to the upper panel of the second **SplitContainer** control you added to your form. Set the **Dock** property of the **ListView** control to **Fill**.
6. Drag a **RichTextBox** control from the **Toolbox** to the lower panel of the second **SplitContainer** control. Set the **Dock** property of the **RichTextBox** control to **Fill**.

At this point, if you press F5 to run the application, the form displays a three-part user interface, similar to that of Microsoft Outlook.

NOTE

When you put the mouse pointer over either of the splitters within the [SplitContainer](#) controls, you can resize the internal dimensions.

At this point in application development, you have crafted a sophisticated user interface. The next step is proceeding with the programming of the application itself, perhaps by connecting the [TreeView](#) control and [ListView](#) controls to some kind of data source. For more information about connecting controls to data, see [Data Binding and Windows Forms](#).

See Also

[SplitContainer](#)

[SplitContainer Control](#)

How to: Create a Windows Explorer–Style Interface on a Windows Form

5/4/2018 • 1 min to read • [Edit Online](#)

Windows Explorer is a common user-interface choice for applications because of its ready familiarity.

Windows Explorer is, essentially, a [TreeView](#) control and a [ListView](#) control on separate panels. The panels are made resizable by a splitter. This arrangement of controls is very effective for displaying and browsing information.

The following steps show how to arrange controls in a Windows Explorer-like form. They do not show how to add the file-browsing functionality of the Windows Explorer application.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To create a Windows Explorer-style Windows Form

1. Create a new Windows Application project. For details, see [How to: Create a Windows Application Project](#).
2. From the **Toolbox**:
 - a. Drag a [SplitContainer](#) control onto your form.
 - b. Drag a [TreeView](#) control into **SplitterPanel1** (the panel of the [SplitContainer](#) control marked **Panel1**).
 - c. Drag a [ListView](#) control into **SplitterPanel2** (the panel of the [SplitContainer](#) control marked **Panel2**).
3. Select all three controls by pressing the CTRL key and clicking them in turn. When you select the [SplitContainer](#) control, click the splitter bar, rather than the panels.

NOTE

Do not use the **Select All** command on the **Edit** menu. If you do so, the property needed in the next step will not appear in the **Properties** window.

4. In the **Properties** window, set the [Dock](#) property to [Fill](#).
5. Press F5 to run the application.

The form displays a two-part user interface, similar to that of the Windows Explorer.

NOTE

When you drag the splitter, the panels resize themselves.

See Also

[SplitContainer](#)

[How to: Create a Multipane User Interface with Windows Forms](#)

[How to: Define Resize and Positioning Behavior in a Split Window](#)

[How to: Split a Window Horizontally](#)

[SplitContainer Control](#)

How to: Define Resize and Positioning Behavior in a Split Window

5/4/2018 • 1 min to read • [Edit Online](#)

The panels of the [SplitContainer](#) control lend themselves well to being resized and manipulated by users. However, there will be times when you will want to programmatically control the splitter—where it is positioned and to what degree it can be moved.

The [SplitterIncrement](#) property and the other properties on the [SplitContainer](#) control give you precise control over the behavior of your user interface to suit your needs. These properties are listed in the following table.

NAME	DESCRIPTION
IsSplitterFixed property	Determines if the splitter is movable by means of the keyboard or mouse.
SplitterDistance property	Determines the distance in pixels from the left or upper edge to the movable splitter bar.
SplitterIncrement property	Determines the minimum distance, in pixels, that the splitter can be moved by the user.

The example below modifies the [SplitterIncrement](#) property to create a "snapping splitter" effect; when the user drags the splitter, it increments in units of 10 pixels rather than the default 1.

To define SplitContainer resize behavior

1. In a procedure, set the [SplitterIncrement](#) property to the desired size, so that the 'snapping' behavior of the splitter is achieved.

In the following code example, within the form's [Load](#) event, the splitter within the [SplitContainer](#) control is set to jump 10 pixels when dragged.

```
Private Sub Form1_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load
    Dim splitSnapper As New SplitContainer()
    splitSnapper.SplitterIncrement = 10
    splitSnapper.Dock = DockStyle.Fill
    splitSnapper.Parent = Me
End Sub
```

```
private void Form1_Load(System.Object sender, System.EventArgs e)
{
    SplitContainer splitSnapper = new SplitContainer();
    splitSnapper.SplitterIncrement = 10;
    splitSnapper.Dock = DockStyle.Fill;
    splitSnapper.Parent = this;
}
```

(Visual C#) Place the following code in the form's constructor to register the event handler.

```
this.Load += new System.EventHandler(this.Form1_Load);
```

Moving the splitter slightly to the left or right will have no discernible effect; however, when the mouse pointer goes 10 pixels in either direction, the splitter will snap to the new position.

See Also

[SplitContainer](#)

[SplitterIncrement](#)

How to: Split a Window Horizontally

5/4/2018 • 1 min to read • [Edit Online](#)

The following code example makes the splitter that divides the [SplitContainer](#) control horizontal.

NOTE

The [Orientation](#) property of the [SplitContainer](#) control determines the direction of the splitter, not of the control itself.

To split a window horizontally

1. Within a procedure, set the [Orientation](#) property of the [SplitContainer](#) control to [Horizontal](#).

```
Sub ShowSplitContainer()
    Dim splitContainer1 As New SplitContainer()
    splitContainer1.BorderStyle = BorderStyle.FixedSingle
    splitContainer1.Location = New System.Drawing.Point(74, 20)
    splitContainer1.Name = "DemoSplitContainer"
    splitContainer1.Size = New System.Drawing.Size(212, 435)
    splitContainer1.TabIndex = 0
    splitContainer1.Orientation = Orientation.Horizontal
    Controls.Add(splitContainer1)
End Sub
```

```
public void showSplitContainer()
{
    SplitContainer splitContainer1 = new SplitContainer();
    splitContainer1.BorderStyle = BorderStyle.FixedSingle;
    splitContainer1.Location = new System.Drawing.Point(74, 20);
    splitContainer1.Name = "DemoSplitContainer";
    splitContainer1.Size = new System.Drawing.Size(212, 435);
    splitContainer1.TabIndex = 0;
    splitContainer1.Orientation = Orientation.Horizontal;
    this.Controls.Add (splitContainer1);

}
```

See Also

[SplitContainer](#)

[SplitContainer Control](#)

How to: Split a Window Horizontally Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

This example makes the splitter that divides the [SplitContainer](#) control horizontal.

NOTE

The [Orientation](#) property of the [SplitContainer](#) control determines the direction of the splitter, not of the control itself. The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To split a window horizontally

1. In the **Properties** window, set the [Orientation](#) property of the [SplitContainer](#) control to [Horizontal](#).

See Also

[SplitContainer](#)

[SplitContainer Control](#)

Splitter Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

Windows Forms `Splitter` controls are used to resize docked controls at run time. The `Splitter` control is often used on forms with controls that have varying lengths of data to present, like Windows Explorer, whose data panes contain information of varying widths at different times.

NOTE

Although `SplitContainer` replaces and adds functionality to the `Splitter` control of previous versions, `Splitter` is retained for both backward compatibility and future use if you choose.

In This Section

[Splitter Control Overview](#)

Explains what this control is and its key features and properties.

Reference

[Splitter class](#)

Describes this class and has links to all its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

Splitter Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

IMPORTANT

Although [SplitContainer](#) replaces and adds functionality to the [Splitter](#) control of previous versions, [Splitter](#) is retained for both backward compatibility and future use if you choose.

Windows Forms [Splitter](#) controls are used to resize docked controls at run time. The [Splitter](#) control is often used on forms with controls that have varying lengths of data to present, like Windows Explorer, whose data panes contain information of varying widths at different times.

Working with the Splitter Control

When the user points the mouse pointer at the undocked edge of a control that can be resized by a splitter control, the pointer changes its appearance to indicate that the control can be resized. With the splitter control, the user can resize the docked control that is immediately before it. Therefore, to enable the user to resize a docked control at run time, dock the control to be resized to an edge of a container, and then dock a splitter control to the same side of that container.

See Also

[SplitContainer](#)

[How to: Dock Controls on Windows Forms](#)

[Controls to Use on Windows Forms](#)

StatusBar Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

NOTE

The [ToolStripStatusLabel](#) control replaces and adds functionality to the [StatusBar](#) control; however, the [StatusBar](#) control is retained for both backward compatibility and future use, if you choose.

The Windows Forms [StatusBar](#) control is used on forms as an area, usually displayed at the bottom of a window, in which an application can display various kinds of status information. [StatusBar](#) controls can have status-bar panels on them that display icons to indicate state, or a series of icons in an animation that indicate a process is working; for example, Microsoft Word indicating that the document is being saved.

In This Section

[StatusBar Control Overview](#)

Introduces the general concepts of the [StatusBar](#) control, which enables users to see relevant information for the control that has focus.

[How to: Add Panels to a StatusBar Control](#)

Explains how to add programmable panels to the [StatusBar](#) control.

[How to: Determine Which Panel in the Windows Forms StatusBar Control Was Clicked](#)

Explains how to handle [Click](#) events raised from the [StatusBar](#) control.

[How to: Set the Size of Status-Bar Panels](#)

Gives details on the properties that control the width and resize behavior of status-bar panels at run time.

[Walkthrough: Updating Status Bar Information at Run Time](#)

Explains how to programmatically control the data within status-bar panels.

Reference

[StatusBar](#)

Provides reference information on the class and its members.

[ToolStripStatusLabel](#)

Replaces and adds functionality to the [StatusBar](#) control.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

StatusBar Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

IMPORTANT

The [StatusStrip](#) and [ToolStripStatusLabel](#) controls replace and add functionality to the [StatusBar](#) and [StatusBarPanel](#) controls; however, the [StatusBar](#) and [StatusBarPanel](#) controls are retained for both backward compatibility and future use, if you choose.

The Windows Forms [StatusBar Control](#) is used on forms as an area, usually displayed at the bottom of a window, in which an application can display various kinds of status information. [StatusBar](#) controls can have status bar panels on them that display text or icons to indicate state, or a series of icons in an animation that indicate a process is working; for example, Microsoft Word indicating that the document is being saved.

Using the StatusBar Control

Internet Explorer uses a status bar to indicate the URL of a page when the mouse rolls over the hyperlink; Microsoft Word gives you information on page location, section location, and editing modes such as overtype and revision tracking; and Visual Studio uses the status bar to give context-sensitive information, such as telling you how to manipulate dockable windows as either docked or floating.

You can display a single message on the status bar by setting the [ShowPanels](#) property to `false` (the default) and setting the [Text](#) property of the status bar to the text you want to appear in the status bar. You can divide the status bar into panels to display more than one type of information by setting the [ShowPanels](#) property to `true` and using the [Add](#) method of [StatusBar.StatusBarPanelCollection](#).

See Also

[StatusBar](#)

[ToolStripStatusLabel](#)

[How to: Determine Which Panel in the Windows Forms StatusBar Control Was Clicked](#)

How to: Add Panels to a StatusBar Control

5/4/2018 • 2 min to read • [Edit Online](#)

IMPORTANT

The [StatusStrip](#) and [ToolStripStatusLabel](#) controls replace and add functionality to the [StatusBar](#) and [StatusBarPanel](#) controls; however, the [StatusBar](#) and [StatusBarPanel](#) controls are retained for both backward compatibility and future use, if you choose.

The programmable area within a [StatusBar Control](#) control consists of instances of the [StatusBarPanel](#) class. These are added through additions to the [StatusBar.StatusBarPanelCollection](#) class.

To add panels to a status bar

1. In a procedure, create status-bar panels by adding them to the [StatusBar.StatusBarPanelCollection](#). Specify property settings for individual panels by using its index passed through the [Panels](#) property.

In the following code example, the path set for the location of the icon is the **My Documents** folder. This location is used because you can assume that most computers running the Windows operating system will include this folder. Choosing this location also allows users with minimal system access levels to safely run the application. The following example requires a form with a [StatusBar](#) control already added.

NOTE

The [StatusBar.StatusBarPanelCollection](#) is a zero-based collection, so code should proceed accordingly.

```
Public Sub CreateStatusBarPanels()
    ' Create panels and set text property.
    StatusBar1.Panels.Add("One")
    StatusBar1.Panels.Add("Two")
    StatusBar1.Panels.Add("Three")
    ' Set properties of StatusBar panels.
    ' Set AutoSize property of panels.
    StatusBar1.Panels(0).AutoSize = StatusBarPanelAutoSize.Spring
    StatusBar1.Panels(1).AutoSize = StatusBarPanelAutoSize.Contents
    StatusBar1.Panels(2).AutoSize = StatusBarPanelAutoSize.Contents
    ' Set BorderStyle property of panels.
    StatusBar1.Panels(0).BorderStyle = StatusBarPanelBorderStyle.Raised
    StatusBar1.Panels(1).BorderStyle = StatusBarPanelBorderStyle.Sunken
    StatusBar1.Panels(2).BorderStyle = StatusBarPanelBorderStyle.Raised
    ' Set Icon property of third panel. You should replace the bolded
    ' icon in the sample below with an icon of your own choosing.
    StatusBar1.Panels(2).Icon = New _
        System.Drawing.Icon(System.Environment.GetFolderPath _
        (System.Environment.SpecialFolder.Personal) _
        & "\Icon.ico")
    StatusBar1.ShowPanels = True
End Sub
```

```

public void CreateStatusBarPanels()
{
    // Create panels and set text property.
    statusBar1.Panels.Add("One");
    statusBar1.Panels.Add("Two");
    statusBar1.Panels.Add("Three");
    // Set properties of StatusBar panels.
    // Set AutoSize property of panels.
    statusBar1.Panels[0].AutoSize = StatusBarPanelAutoSize.Spring;
    statusBar1.Panels[1].AutoSize = StatusBarPanelAutoSize.Contents;
    statusBar1.Panels[2].AutoSize = StatusBarPanelAutoSize.Contents;
    // Set BorderStyle property of panels.
    statusBar1.Panels[0].BorderStyle =
        StatusBarPanelBorderStyle.Raised;
    statusBar1.Panels[1].BorderStyle = StatusBarPanelBorderStyle.Sunken;
    statusBar1.Panels[2].BorderStyle = StatusBarPanelBorderStyle.Raised;
    // Set Icon property of third panel. You should replace the bolded
    // icon in the sample below with an icon of your own choosing.
    // Note the escape character used (@) when specifying the path.
    statusBar1.Panels[2].Icon =
        new System.Drawing.Icon (System.Environment.GetFolderPath _
        (System.Environment.SpecialFolder.Personal) _
        +"\\Icon.ico");
    statusBar1.ShowPanels = true;
}

```

```

public:
void CreateStatusBarPanels()
{
    // Create panels and set text property.
    statusBar1->Panels->Add("One");
    statusBar1->Panels->Add("Two");
    statusBar1->Panels->Add("Three");
    // Set properties of StatusBar panels.
    // Set AutoSize property of panels.
    statusBar1->Panels[0]->AutoSize =
        StatusBarPanelAutoSize::Spring;
    statusBar1->Panels[1]->AutoSize =
        StatusBarPanelAutoSize::Contents;
    statusBar1->Panels[2]->AutoSize =
        StatusBarPanelAutoSize::Contents;
    // Set BorderStyle property of panels.
    statusBar1->Panels[0]->BorderStyle =
        StatusBarPanelBorderStyle::Raised;
    statusBar1->Panels[1]->BorderStyle =
        StatusBarPanelBorderStyle::Sunken;
    statusBar1->Panels[2]->BorderStyle =
        StatusBarPanelBorderStyle::Raised;
    // Set Icon property of third panel.
    // You should replace the bolded image
    // in the sample below with an icon of your own choosing.
    statusBar1->Panels[2]->Icon =
        gcnew System::Drawing::Icon(String::Concat(
        System::Environment::GetFolderPath(
        System::Environment::SpecialFolder::Personal),
        "\\Icon.ico"));
    statusBar1->ShowPanels = true;
}

```

See Also

[StatusBar](#)

[ToolStripStatusLabel](#)

[Collection Editor Dialog Box](#)

[How to: Set the Size of Status-Bar Panels](#)

[Walkthrough: Updating Status Bar Information at Run Time](#)

[How to: Determine Which Panel in the Windows Forms StatusBar Control Was Clicked](#)

[StatusBar Control Overview](#)

How to: Determine Which Panel in the Windows Forms StatusBar Control Was Clicked

5/4/2018 • 1 min to read • [Edit Online](#)

IMPORTANT

The [StatusStrip](#) and [ToolStripStatusLabel](#) controls replace and add functionality to the [StatusBar](#) and [StatusBarPanel](#) controls; however, the [StatusBar](#) and [StatusBarPanel](#) controls are retained for both backward compatibility and future use, if you choose.

To program the [StatusBar Control](#) control to respond to user clicks, use a case statement within the [PanelClick](#) event. The event contains an argument (the panel argument), which contains a reference to the clicked [StatusBarPanel](#). Using this reference, you can determine the index of the clicked panel, and program accordingly.

NOTE

Ensure that the [StatusBar](#) control's [ShowPanels](#) property is set to `true`.

To determine which panel was clicked

1. In the [PanelClick](#) event handler, use a `Select Case` (in Visual Basic) or `switch case` (Visual C# or Visual C++) statement to determine which panel was clicked by examining the index of the clicked panel in the event arguments.

The following code example requires the presence, on the form, of a [StatusBar](#) control, `StatusBar1`, and two [StatusBarPanel](#) objects, `StatusBarPanel1` and `StatusBarPanel2`.

```
Private Sub StatusBar1_PanelClick(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.StatusBarPanelEventArgs) Handles StatusBar1.PanelClick
    Select Case StatusBar1.Panels.IndexOf(e.StatusBarPanel)
        Case 0
            MessageBox.Show("You have clicked Panel One.")
        Case 1
            MessageBox.Show("You have clicked Panel Two.")
    End Select
End Sub
```

```
private void statusBar1_PanelClick(object sender,
System.Windows.Forms.StatusBarPanelEventArgs e)
{
    switch (statusBar1.Panels.IndexOf(e.StatusBarPanel))
    {
        case 0 :
            MessageBox.Show("You have clicked Panel One.");
            break;
        case 1 :
            MessageBox.Show("You have clicked Panel Two.");
            break;
    }
}
```

```
private:  
    void statusBar1_PanelClick(System::Object ^ sender,  
        System::Windows::Forms::StatusBarPanelEventArgs ^ e)  
{  
    switch (statusBar1->Panels->IndexOf(e->StatusBarPanel))  
    {  
        case 0 :  
            MessageBox::Show("You have clicked Panel One.");  
            break;  
        case 1 :  
            MessageBox::Show("You have clicked Panel Two.");  
            break;  
    }  
}
```

(Visual C#, Visual C++) Place the following code in the form's constructor to register the event handler.

```
this->statusBar1->PanelClick += gcnew  
    System::Windows::Forms::StatusBarPanelClickEventHandler  
    (this, &Form1::statusBar1_PanelClick);
```

```
this->statusBar1->PanelClick += new  
    System::Windows::Forms::StatusBarPanelClickEventHandler  
    (this, &Form1::statusBar1_PanelClick);
```

See Also

[StatusBar](#)

[ToolStripStatusLabel](#)

[How to: Set the Size of Status-Bar Panels](#)

[Walkthrough: Updating Status Bar Information at Run Time](#)

[StatusBar Control Overview](#)

How to: Set the Size of Status-Bar Panels

5/4/2018 • 1 min to read • [Edit Online](#)

NOTE

The [ToolStripStatusLabel](#) control replaces and adds functionality to the [StatusBar](#) control; however, the [StatusBar](#) control is retained for both backward compatibility and future use, if you choose.

Each instance of the [StatusBarPanel](#) class within a [StatusBar Control](#) has a number of dynamic properties that determine its width and resize behavior at run time.

To set the size of a panel

1. In a procedure, set the [AutoSize](#), [MinWidth](#), and [Width](#) properties (or any subset therein) for the status-bar panels using their index passed through the [Panels](#) property of the [StatusBarPanel](#) collection.

```
Public Sub SetStatusBarPanelSize()
    ' Create panel and set text property.
    StatusBar1.Panels.Add("One")
    ' Set properties of panels.
    StatusBar1.Panels(0).AutoSize = StatusBarPanelAutoSize.Spring
    StatusBar1.Panels(0).Width = 200
    ' Enable the StatusBar control to display panels.
    StatusBar1.ShowPanels = True
End Sub
```

```
public void SetStatusBarPanelSize()
{
    // Create panel and set text property.
    statusBar1.Panels.Add("One");
    // Set properties of panels.
    statusBar1.Panels[0].AutoSize = StatusBarPanelAutoSize.Spring;
    statusBar1.Panels[0].Width = 200;
    statusBar1.ShowPanels = true;
}
```

```
public:
void SetStatusBarPanelSize()
{
    // Create panel and set text property.
    statusBar1->Panels->Add("One");
    // Set properties of panels.
    statusBar1->Panels[0]->AutoSize =
        StatusBarPanelAutoSize::Spring;
    statusBar1->Panels[0]->Width = 200;
    statusBar1->ShowPanels = true;
}
```

See Also

[StatusBar](#)

[ToolStripStatusLabel](#)

[Walkthrough: Updating Status Bar Information at Run Time](#)

[How to: Determine Which Panel in the Windows Forms StatusBar Control Was Clicked](#)
[StatusBar Control Overview](#)

Walkthrough: Updating Status Bar Information at Run Time

5/4/2018 • 2 min to read • [Edit Online](#)

IMPORTANT

The [StatusStrip](#) and [ToolStripStatusLabel](#) controls replace and add functionality to the [StatusBar](#) and [StatusBarPanel](#) controls; however, the [StatusBar](#) and [StatusBarPanel](#) controls are retained for both backward compatibility and future use, if you choose.

Often, a program will call for you to update the contents of status bar panels dynamically at run time, based on changes to application state or other user interaction. This is a common way to signal users that keys such as the CAPS LOCK, NUM LOCK, or SCROLL LOCK are enabled, or to provide the date or a clock as a convenient reference.

In the following example, you will use an instance of the [StatusBarPanel](#) class to host a clock.

To get the status bar ready for updating

1. Create a new Windows form.
2. Add a [StatusBar](#) control to your form. For details, see [How to: Add Controls to Windows Forms](#).
3. Add a status bar panel to your [StatusBar](#) control. For details, see [How to: Add Panels to a StatusBar Control](#).
4. For the [StatusBar](#) control you added to your form, set the [ShowPanels](#) property to `true`.
5. Add a Windows Forms [Timer](#) component to the form.

NOTE

The Windows Forms [System.Windows.Forms.Timer](#) component is designed for a Windows Forms environment. If you need a timer that is suitable for a server environment, see [Introduction to Server-Based Timers](#).

6. Set the [Enabled](#) property to `true`.
7. Set the [Interval](#) property of the [Timer](#) to 30000.

NOTE

The [Interval](#) property of the [Timer](#) component is set to 30 seconds (30,000 milliseconds) to ensure that an accurate time is reflected in the time displayed.

To implement the timer to update the status bar

1. Insert the following code into the event handler of the [Timer](#) component to update the panel of the [StatusBar](#) control.

```
Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Timer1.Tick
    StatusBar1.Panels(0).Text = Now.ToShortTimeString()
End Sub
```

```
private void timer1_Tick(object sender, System.EventArgs e)
{
    statusBar1.Panels[0].Text = DateTime.Now.ToShortTimeString();
}
```

```
private:
System::Void timer1_Tick(System::Object ^ sender,
    System::EventArgs ^ e)
{
    statusBar1->Panels[0]->Text =
    DateTime::Now.ToShortTimeString();
}
```

At this point, you are ready to run the application and observe the clock running in the status bar panel.

To test the application

1. Debug the application and press F5 to run it. For details about debugging, see [Debugging in Visual Studio](#).

NOTE

It will take approximately 30 seconds for the clock to appear in the status bar. This is to get the most accurate time possible. Conversely, to make the clock appear sooner, you can reduce the value of the [Interval](#) property you set in step 7 in the previous procedure.

See Also

[StatusBar](#)

[ToolStripStatusLabel](#)

[How to: Add Panels to a StatusBar Control](#)

[How to: Determine Which Panel in the Windows Forms StatusBar Control Was Clicked](#)

[StatusBar Control Overview](#)

StatusStrip Control

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `StatusStrip` control is used on forms as an area, usually displayed at the bottom of a window, in which an application can display various kinds of status information. `StatusStrip` controls typically have `ToolStripStatusLabel` controls on them that display text or icons to indicate state, or a `ToolStripProgressBar` that graphically displays the completion state of a process.

In This Section

[StatusStrip Control Overview](#)

Explains what this control is and its key features and properties.

[How to: Use the Spring Property Interactively in a StatusStrip](#)

Demonstrates using the `Spring` property to interactively center a `ToolStripStatusLabel` in a `StatusStrip`.

Also see [StatusStrip Items Collection Editor](#), [StatusStrip Items Collection Editor](#).

Reference

[StatusStrip](#)

Provides reference information on the class and its members.

[ToolStripStatusLabel](#)

Provides reference information on the class and its members.

See Also

[Controls to Use on Windows Forms](#)

StatusStrip Control Overview

5/4/2018 • 1 min to read • [Edit Online](#)

A [StatusStrip](#) control displays information about an object being viewed on a [Form](#), the object's components, or contextual information that relates to that object's operation within your application. Typically, a [StatusStrip](#) control consists of [ToolStripStatusLabel](#) objects, each of which displays text, an icon, or both. The [StatusStrip](#) can also contain [ToolStripDropDownButton](#), [ToolStripSplitButton](#), and [ToolStripProgressBar](#) controls.

The default [StatusStrip](#) has no panels. To add panels to a [StatusStrip](#), use the [ToolStripItemCollection.AddRange](#) method.

There is extensive support for handling [StatusStrip](#) items and common commands in Visual Studio.

Also see [StatusStrip Items Collection Editor](#), [StatusStrip Tasks Dialog Box](#).

Although [StatusStrip](#) replaces and extends the [StatusBar](#) control of previous versions, [StatusBar](#) is retained for both backward compatibility and future use if you choose.

Important StatusStrip Members

NAME	DESCRIPTION
CanOverflow	Gets or sets a value indicating whether the StatusStrip supports overflow functionality.
Stretch	Gets or sets a value indicating whether the StatusStrip stretches from end to end in its ToolStripContainer .

Important StatusStrip Companion Classes

NAME	DESCRIPTION
ToolStripStatusLabel	Represents a panel in a StatusStrip control.
ToolStripDropDownButton	Displays an associated ToolStripDropDown from which the user can select a single item.
ToolStripSplitButton	Represents a two-part control that is a standard button and a drop-down menu.
ToolStripProgressBar	Displays the completion state of a process.

See Also

[StatusStrip](#)

[ToolStripStatusLabel](#)

[Spring](#)

How to: Use the Spring Property Interactively in a StatusStrip

5/4/2018 • 2 min to read • [Edit Online](#)

You can use the [Spring](#) property to position a [ToolStripStatusLabel](#) control in a [StatusStrip](#) control. The [Spring](#) property determines whether the [ToolStripStatusLabel](#) control automatically fills the available space on the [StatusStrip](#) control.

Example

The following code example demonstrates how to use the [Spring](#) property to position a [ToolStripStatusLabel](#) control in a [StatusStrip](#) control. The [Click](#) event handler performs an exclusive-or (XOR) operation to switch the value of the [Spring](#) property.

To use this code example, compile and run the application, and then click **Middle (Spring)** on the [StatusStrip](#) control to switch the value of the [Spring](#) property.

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Drawing;
```

```
Imports System
Imports System.Collections.Generic
Imports System.Windows.Forms
Imports System.Drawing
```

```
// This code example demonstrates using the Spring property
// to interactively center a ToolStripStatusLabel in a StatusStrip.
class Form4 : Form
{
    // Declare the ToolStripStatusLabel.
    ToolStripStatusLabel middleLabel;

    public Form4()
    {
        // Create a new StatusStrip control.
        StatusStrip ss = new StatusStrip();

        // Add the leftmost label.
        ss.Items.Add("Left");

        // Handle middle label separately -- action will occur
        // when the label is clicked.
        middleLabel = new ToolStripStatusLabel("Middle (Spring)");
        middleLabel.Click += new EventHandler(middleLabel_Click);
        ss.Items.Add(middleLabel);

        // Add the rightmost label
        ss.Items.Add("Right");

        // Add the StatusStrip control to the controls collection.
        this.Controls.Add(ss);
    }

    // This event hadler is invoked when the
    // middleLabel control is clicked. It toggles
    // the value of the Spring property.
    void middleLabel_Click(object sender, EventArgs e)
    {
        // Toggle the value of the Spring property.
        middleLabel.Spring ^= true;

        // Set the Text property according to the
        // value of the Spring property.
        middleLabel.Text =
            middleLabel.Spring ? "Middle (Spring - True)" : "Middle (Spring - False)";
    }
}
```

```

' This code example demonstrates using the Spring property
' to interactively center a ToolStripStatusLabel in a StatusStrip.
Class Form4
    Inherits Form

    ' Declare the ToolStripStatusLabel.
    Private middleLabel As ToolStripStatusLabel

    Public Sub New()
        ' Create a new StatusStrip control.
        Dim ss As New StatusStrip()

        ' Add the leftmost label.
        ss.Items.Add("Left")

        ' Handle middle label separately -- action will occur
        ' when the label is clicked.
        middleLabel = New ToolStripStatusLabel("Middle (Spring)")
        AddHandler middleLabel.Click, AddressOf middleLabel_Click
        ss.Items.Add(middleLabel)

        ' Add the rightmost label
        ss.Items.Add("Right")

        ' Add the StatusStrip control to the controls collection.
        Me.Controls.Add(ss)
    End Sub

    ' This event hadler is invoked when the
    ' middleLabel control is clicked. It toggles
    ' the value of the Spring property.
    Sub middleLabel_Click(ByVal sender As Object, ByVal e As EventArgs)

        ' Toggle the value of the Spring property.
        middleLabel.Spring = middleLabel.Spring Xor True

        ' Set the Text property according to the
        ' value of the Spring property.
        middleLabel.Text = IIf(middleLabel.Spring, _
            "Middle (Spring - True)", "Middle (Spring - False)")
    End Sub
End Class

```

Compiling the Code

This example requires:

- References to the System.Design, System.Drawing, and System.Windows.Forms assemblies.

For information about building this example from the command line for visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[ToolStripStatusLabel](#)

[StatusStrip](#)

[ToolStrip](#)

[ToolStripItem](#)

[ToolStripMenuItem](#)

ToolStrip Control

TabControl Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `TabControl` displays multiple tabs, like dividers in a notebook or labels in a set of folders in a filing cabinet. The tabs can contain pictures and other controls. Use the `TabControl` to create property pages.

In This Section

[TabControl Control Overview](#)

Explains what this control is and its key features and properties.

[How to: Add a Control to a Tab Page](#)

Gives directions for displaying controls on tab pages.

[How to: Add and Remove Tabs with the Windows Forms TabControl](#)

Gives directions for adding and removing tabs in the designer or in code.

[How to: Change the Appearance of the Windows Forms TabControl](#)

Gives directions for adjusting properties that affect the appearance of individual tabs.

[How to: Disable Tab Pages](#)

Explains how to restrict access to a tab page, possibly based on user credentials.

Also see [How to: Add and Remove Tabs with the Windows Forms TabControl Using the Designer](#), [How to: Add a Control to a Tab Page Using the Designer](#)

Reference

[TabControl class](#)

Describes this class and has links to all its members.

Related Sections

[Dialog Boxes in Windows Forms](#)

Provides a list of tasks for dialog boxes, which often display tabs.

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

TabControl Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [TabControl](#) displays multiple tabs, like dividers in a notebook or labels in a set of folders in a filing cabinet. The tabs can contain pictures and other controls. You can use the tab control to produce the kind of multiple-page dialog box that appears many places in the Windows operating system, such as the Control Panel Display Properties. Additionally, the [TabControl](#) can be used to create property pages, which are used to set a group of related properties.

Working with TabControl

The most important property of the [TabControl](#) is [TabPages](#), which contains the individual tabs. Each individual tab is a [TabPage](#) object. When a tab is clicked, it raises the [Click](#) event for that [TabPage](#) object.

See Also

[TabControl](#)

[TabControl Control](#)

[How to: Change the Appearance of the Windows Forms TabControl](#)

[How to: Add a Control to a Tab Page](#)

[How to: Add and Remove Tabs with the Windows Forms TabControl](#)

[How to: Disable Tab Pages](#)

[Dialog Boxes in Windows Forms](#)

How to: Add a Control to a Tab Page

5/4/2018 • 1 min to read • [Edit Online](#)

You can use the Windows Forms [TabControl](#) to display other controls in an organized fashion. The following procedure shows how to add a button to the first tab. For information about adding an icon to the label part of a tab page, see [How to: Change the Appearance of the Windows Forms TabControl](#).

To add a control programmatically

1. Use the [Add](#) method of the collection returned by the [Controls](#) property of [TabPage](#):

```
tabPage1->Controls->Add(gcnew Button);
```

```
tabPage1.Controls.Add(new Button());
```

```
tabPage1.Controls.Add(New Button())
```

See Also

[TabControl Control](#)

[TabControl Control Overview](#)

[How to: Change the Appearance of the Windows Forms TabControl](#)

[How to: Disable Tab Pages](#)

[How to: Add and Remove Tabs with the Windows Forms TabControl](#)

How to: Add a Control to a Tab Page Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

The use of the Windows Forms [TabControl](#) is to display other controls in an organized fashion. You can use these instructions to display a picture on the main part of a tab page. For information about adding an icon to the label part of a tab page, see [How to: Change the Appearance of the Windows Forms TabControl](#).

The following procedure requires a **Windows Application** project with a form containing a [TabControl](#) control. For information about setting up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To add a control using the designer

1. Click the appropriate tab page so that it appears on top.
2. Draw the control on the tab page.

See Also

[TabControl Control](#)

[TabControl Control Overview](#)

[How to: Change the Appearance of the Windows Forms TabControl](#)

[How to: Disable Tab Pages](#)

[How to: Add and Remove Tabs with the Windows Forms TabControl](#)

How to: Add and Remove Tabs with the Windows Forms TabControl

5/4/2018 • 1 min to read • [Edit Online](#)

By default, a [TabControl](#) control contains two [TabPage](#) controls. You can access these tabs through the [TabPages](#) property.

To add a tab programmatically

- Use the [Add](#) method of the [TabPages](#) property.

```
Dim myTabPage As NewTabPage()
myTabPage.Text = "TabPage" & (TabControl1.TabPages.Count + 1)
TabControl1.TabPages.Add(myTabPage)
```

```
string title = "TabPage " + (tabControl1.TabCount + 1).ToString();
TabPage myTabPage = newTabPage(title);
tabControl1.TabPages.Add(myTabPage);
```

```
String^ title = String::Concat("TabPage ",
    (tabControl1->TabCount + 1).ToString());
TabPage^ myTabPage = gcnewTabPage(title);
tabControl1->TabPages->Add(myTabPage);
```

To remove a tab programmatically

- To remove selected tabs, use the [Remove](#) method of the [TabPages](#) property.

-or-

- To remove all tabs, use the [Clear](#) method of the [TabPages](#) property.

```
' Removes the selected tab:
TabControl1.TabPages.Remove(TabControl1.SelectedTab)
' Removes all the tabs:
TabControl1.TabPages.Clear()
```

```
// Removes the selected tab:
tabControl1.TabPages.Remove(tabControl1.SelectedTab);
// Removes all the tabs:
tabControl1.TabPages.Clear();
```

```
// Removes the selected tab:
tabControl1->TabPages->Remove(tabControl1->SelectedTab);
// Removes all the tabs:
tabControl1->TabPages->Clear();
```

See Also

[TabControl Control Overview](#)

[How to: Add a Control to a Tab Page](#)

[How to: Disable Tab Pages](#)

[How to: Change the Appearance of the Windows Forms TabControl](#)

How to: Add and Remove Tabs with the Windows Forms TabControl Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

When you place a [TabControl](#) control on your form, it contains two tabs by default. You can add or remove tabs using the designer.

The following procedure requires a **Windows Application** project with a form containing a [TabControl](#) control. For information about setting up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To add or remove a tab using the designer

- On the control's smart tag, click **Add Tab** or **Remove Tab**

-or-

In the **Properties** window, click the **Ellipsis** button (next to the **TabPage** property to open the **TabPage Collection Editor**. Click the **Add** or **Remove** button.

See Also

- [TabControl Control](#)
- [TabControl Control Overview](#)
- [How to: Add a Control to a Tab Page](#)
- [How to: Disable Tab Pages](#)
- [How to: Change the Appearance of the Windows Forms TabControl](#)

How to: Change the Appearance of the Windows Forms TabControl

5/4/2018 • 1 min to read • [Edit Online](#)

You can change the appearance of tabs in Windows Forms by using properties of the [TabControl](#) and the [TabPage](#) objects that make up the individual tabs on the control. By setting these properties, you can display images on tabs, display tabs vertically instead of horizontally, display multiple rows of tabs, and enable or disable tabs programmatically.

To display an icon on the label part of a tab

1. Add an [ImageList](#) control to the form.
2. Add images to the image list.

For more information about image lists, see [ImageList Component](#) and [How to: Add or Remove Images with the Windows Forms ImageList Component](#).

3. Set the [ImageList](#) property of the [TabControl](#) to the [ImageList](#) control.
4. Set the [ImageIndex](#) property of the [TabPage](#) to the index of an appropriate image in the list.

To create multiple rows of tabs

1. Add the number of tab pages you want.
2. Set the [Multiline](#) property of the [TabControl](#) to `true`.
3. If the tabs do not already appear in multiple rows, set the [Width](#) property of the [TabControl](#) to be narrower than all the tabs.

To arrange tabs on the side of the control

- Set the [Alignment](#) property of the [TabControl](#) to [Left](#) or [Right](#).

To programmatically enable or disable all controls on a tab

1. Set the [Enabled](#) property of the [TabPage](#) to `true` or `false`.

```
TabPage1.Enabled = False
```

```
tabPage1.Enabled = false;
```

```
tabPage1->Enabled = false;
```

To display tabs as buttons

- Set the [Appearance](#) property of the [TabControl](#) to [Buttons](#) or [FlatButtons](#).

See Also

[TabControl Control](#)

[TabControl Control Overview](#)

[How to: Add a Control to a Tab Page](#)

[How to: Disable Tab Pages](#)

[How to: Add and Remove Tabs with the Windows Forms TabControl](#)

How to: Disable Tab Pages

5/4/2018 • 2 min to read • [Edit Online](#)

On some occasions, you will want to restrict access to data that is available within your Windows Forms application. One example of this might be when you have data displayed in the tab pages of a tab control; administrators could have information on a tab page that you would want to restrict from guest or lower-level users.

To disable tab pages programmatically

1. Write code to handle the tab control's `SelectedIndexChanged` event. This is the event that is raised when the user switches from one tab to the next.
2. Check credentials. Depending upon the information presented, you may want to check the user name the user has logged in with or some other form of credentials before allowing the user to view the tab.
3. If the user has appropriate credentials, display the tab that was clicked. If the user does not have appropriate credentials, display a message box or some other user interface indicating that they do not have access, and return to the initial tab.

NOTE

When you implement this functionality in your production applications, you can perform this credential check during the form's `Load` event. This will allow you to hide the tab before any user interface is shown, which is a much cleaner approach to programming. The methodology used below (checking credentials and disabling the tab during the `SelectedIndexChanged` event) is for illustrative purposes.

4. Optionally, if you have more than two tab pages, display a tab page different from the original.

In the example below, a `CheckBox` control is used in lieu of checking the credentials, as the criteria for access to the tab will vary by application. When the `SelectedIndexChanged` event is raised, if the credential check is true (that is, the check box is checked) and the selected tab is `TabPage2` (the tab with the confidential information, in this example), then `TabPage2` is displayed. Otherwise, `TabPage3` is displayed and a message box is shown to the user, indicating they did not have appropriate access privileges. The code below assumes a form with a `CheckBox` control (`CredentialCheck`) and a `TabControl` control with three tab pages.

```
Private Sub TabControl1_SelectedIndexChanged(ByVal sender As Object, ByVal e As System.EventArgs)
Handles TabControl1.SelectedIndexChanged
    ' Check Credentials Here

    If CredentialCheck.Checked = True And _
    TabControl1.SelectedTab Is TabPage2 Then
        TabControl1.SelectedTab = TabPage2
    ElseIf CredentialCheck.Checked = False _
    And TabControl1.SelectedTab Is TabPage2 Then
        MessageBox.Show _
        ("Unable to load tab. You have insufficient access privileges.")
        TabControl1.SelectedTab = TabPage3
    End If
End Sub
```

```
private void tabControl1_SelectedIndexChanged(object sender, System.EventArgs e)
{
    // Check Credentials Here

    if ((CredentialCheck.Checked == true) && (tabControl1.SelectedTab == tabPage2))
    {
        tabControl1.SelectedTab = tabPage2;
    }
    else if ((CredentialCheck.Checked == false) && (tabControl1.SelectedTab == tabPage2))
    {
        MessageBox.Show("Unable to load tab. You have insufficient access privileges.");
        tabControl1.SelectedTab = tabPage3;
    }
}
```

```
private:
System::Void tabControl1_SelectedIndexChanged(
    System::Object ^ sender,
    System::EventArgs ^ e)
{
    // Check Credentials Here
    if ((CredentialCheck->Checked == true) &&
        (tabControl1->SelectedTab == tabPage2))
    {
        tabControl1->SelectedTab = tabPage2;
    }
    else if ((CredentialCheck->Checked == false) &&
              (tabControl1->SelectedTab == tabPage2))
    {
        MessageBox::Show(String::Concat("Unable to load tab. ",
                                       "You have insufficient access privileges."));
        tabControl1->SelectedTab = tabPage3;
    }
}
```

(Visual C#, Visual C++) Place the following code in the form's constructor to register the event handler.

```
this.tabControl1.SelectedIndexChanged +=
    new System.EventHandler(this.tabControl1_SelectedIndexChanged);
```

```
this->tabControl1->SelectedIndexChanged +=
    gcnew System::EventHandler(this, &Form1::tabControl1_SelectedIndexChanged);
```

See Also

[TabControl Control Overview](#)

[How to: Add a Control to a Tab Page](#)

[How to: Add and Remove Tabs with the Windows Forms TabControl](#)

[How to: Change the Appearance of the Windows Forms TabControl](#)

How to: Display Side-Aligned Tabs with TabControl

5/4/2018 • 2 min to read • [Edit Online](#)

The [Alignment](#) property of [TabControl](#) supports displaying tabs vertically (along the left or right edge of the control), as opposed to horizontally (across the top or bottom of the control). By default, this vertical display results in a poor user experience, because the [Text](#) property of the [TabPage](#) object does not display in the tab when visual styles are enabled. There is also no direct way to control the direction of the text within the tab. You can use owner draw on [TabControl](#) to improve this experience.

The following procedure shows how to render right-aligned tabs, with the tab text running from left to right, by using the "owner draw" feature.

To display right-aligned tabs

1. Add a [TabControl](#) to your form.
2. Set the [Alignment](#) property to [Right](#).
3. Set the [SizeMode](#) property to [Fixed](#), so that all tabs are the same width.
4. Set the [ItemSize](#) property to the preferred fixed size for the tabs. Keep in mind that the [ItemSize](#) property behaves as though the tabs were on top, although they are right-aligned. As a result, in order to make the tabs wider, you must change the [Height](#) property, and in order to make them taller, you must change the [Width](#) property.

For best result with the code example below, set the [Width](#) of the tabs to 25 and the [Height](#) to 100.

5. Set the [DrawMode](#) property to [OwnerDrawFixed](#).
6. Define a handler for the [DrawItem](#) event of [TabControl](#) that renders the text from left to right.

```
public Form1()
{
    // Remove this call if you do not program using Visual Studio.
    InitializeComponent();

    tabControl1.DrawItem += new DrawItemEventHandler(tabControl1_DrawItem);
}

private void tabControl1_DrawItem(Object sender, System.Windows.Forms.DrawItemEventArgs e)
{
    Graphics g = e.Graphics;
    Brush _textBrush;

    // Get the item from the collection.
    TabPage _tabPage = tabControl1.TabPages[e.Index];

    // Get the real bounds for the tab rectangle.
    Rectangle _tabBounds = tabControl1.GetTabRect(e.Index);

    if (e.State == DrawItemState.Selected)
    {

        // Draw a different background color, and don't paint a focus rectangle.
        _textBrush = new SolidBrush(Color.Red);
        g.FillRectangle(Brushes.Gray, e.Bounds);
    }
    else
    {
        _textBrush = new System.Drawing.SolidBrush(e.ForeColor);
        e.DrawBackground();
    }

    // Use our own font.
    Font _tabFont = new Font("Arial", (float)10.0, FontStyle.Bold, GraphicsUnit.Pixel);

    // Draw string. Center the text.
    StringFormat _stringFlags = new StringFormat();
    _stringFlags.Alignment = StringAlignment.Center;
    _stringFlags.LineAlignment = StringAlignment.Center;
    g.DrawString(_tabPage.Text, _tabFont, _textBrush, _tabBounds, new StringFormat(_stringFlags));
}
```

```
Private Sub TabControl1_DrawItem(ByVal sender As Object, ByVal e As
System.Windows.Forms.DrawItemEventArgs) Handles TabControl1.DrawItem
    Dim g As Graphics = e.Graphics
    Dim _TextBrush As Brush

    ' Get the item from the collection.
    Dim _TabPage As TabPage = TabControl1.TabPages(e.Index)

    ' Get the real bounds for the tab rectangle.
    Dim _TabBounds As Rectangle = TabControl1.GetTabRect(e.Index)

    If (e.State = DrawItemState.Selected) Then
        ' Draw a different background color, and don't paint a focus rectangle.
        _TextBrush = New SolidBrush(Color.Red)
        g.FillRectangle(Brushes.Gray, e.Bounds)
    Else
        _TextBrush = New System.Drawing.SolidBrush(e.ForeColor)
        e.DrawBackground()
    End If

    ' Use our own font.
    Dim _TabFont As New Font("Arial", 10.0, FontStyle.Bold, GraphicsUnit.Pixel)

    ' Draw string. Center the text.
    Dim _StringFlags As New StringFormat()
    _StringFlags.Alignment = StringAlignment.Center
    _StringFlags.LineAlignment = StringAlignment.Center
    g.DrawString(_TabPage.Text, _TabFont, _TextBrush, _TabBounds, New StringFormat(_StringFlags))
End Sub
```

See Also

[TabControl Control](#)

TableLayoutPanel Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The [TableLayoutPanel](#) control arranges its contents in a grid. Because the layout is performed both at design time and run time, it can change dynamically as the application environment changes. This gives the controls in the panel the ability to proportionally resize, so it can respond to changes such as the parent control resizing or text length changing due to localization.

In This Section

[TableLayoutPanel Control Overview](#)

Introduces the general concepts of the [TableLayoutPanel](#) control, which allows you to create a layout with rows and columns.

[Best Practices for the TableLayoutPanel Control](#)

Outlines recommendations to help you get the most out of the layout features of the [TableLayoutPanel](#) control.

[AutoSize Behavior in the TableLayoutPanel Control](#)

Explains the ways in which the [TableLayoutPanel](#) control supports automatic sizing behavior.

[How to: Anchor and Dock Child Controls in a TableLayoutPanel Control](#)

Shows how to anchor and dock child controls in a [TableLayoutPanel](#) control.

[How to: Design a Windows Forms Layout that Responds Well to Localization](#)

Demonstrates using a [TableLayoutPanel](#) control to build a form that responds well to localization.

[How to: Create a Resizable Windows Form for Data Entry](#)

Demonstrates using a [TableLayoutPanel](#) control to build a form that responds well to resizing.

1. [How to: Align and Stretch a Control in a TableLayoutPanel Control](#)
2. [How to: Span Rows and Columns in a TableLayoutPanel Control](#)
3. [How to: Edit Columns and Rows in a TableLayoutPanel Control](#)
4. [Walkthrough: Arranging Controls on Windows Forms Using a TableLayoutPanel](#)

Reference

[TableLayoutPanel](#)

Provides reference documentation for the [TableLayoutPanel](#) control.

[TableLayoutSettings](#)

Provides reference documentation for the [TableLayoutSettings](#) type.

[FlowLayoutPanel](#)

Provides reference documentation for the [FlowLayoutPanel](#) control.

Related Sections

[Localization](#)

Provides an overview of topics relating to localization.

Also see [Localizing Applications](#) or [Localizing Applications](#)

TableLayoutPanel Control Overview

5/4/2018 • 1 min to read • [Edit Online](#)

The [TableLayoutPanel](#) control arranges its contents in a grid. Because the layout is performed both at design time and run time, it can change dynamically as the application environment changes. This gives the controls in the panel the ability to proportionally resize, so they can respond to changes such as the parent control resizing or text length changing due to localization.

Any Windows Forms control can be a child of the [TableLayoutPanel](#) control, including other instances of [TableLayoutPanel](#). This allows you to construct sophisticated layouts that adapt to changes at run time.

The [TableLayoutPanel](#) control can expand to accommodate new controls when they are added, depending on the value of the [RowCount](#), [ColumnCount](#), and [GrowStyle](#) properties. Setting either the [RowCount](#) or [ColumnCount](#) property to a value of 0 specifies that the [TableLayoutPanel](#) will be unbound in the corresponding direction.

You can also control the direction of expansion (horizontal or vertical) after the [TableLayoutPanel](#) control is full of child controls. By default, the [TableLayoutPanel](#) control expands downward by adding rows.

If you want rows and columns that behave differently from the default behavior, you can control the properties of rows and columns by using the [RowStyles](#) and [ColumnStyles](#) properties. You can set the properties of rows or columns individually.

The [TableLayoutPanel](#) control adds the following properties to its child controls: [Cell](#), [Column](#), [Row](#), [ColumnSpan](#), and [RowSpan](#).

You can merge cells in the [TableLayoutPanel](#) control by setting the [ColumnSpan](#) or [RowSpan](#) properties on a child control.

1. [How to: Align and Stretch a Control in a TableLayoutPanel Control](#)
2. [How to: Span Rows and Columns in a TableLayoutPanel Control](#)
3. [How to: Edit Columns and Rows in a TableLayoutPanel Control](#)
4. [Walkthrough: Arranging Controls on Windows Forms Using a TableLayoutPanel](#)

See Also

[FlowLayoutPanel](#)

[TableLayoutSettings](#)

[How to: Design a Windows Forms Layout that Responds Well to Localization](#)

[How to: Create a Resizable Windows Form for Data Entry](#)

[Best Practices for the TableLayoutPanel Control](#)

AutoSize Behavior in the TableLayoutPanel Control

5/4/2018 • 1 min to read • [Edit Online](#)

Distinct AutoSize Behaviors

The [TableLayoutPanel](#) control supports automatic sizing behavior in the following ways:

- Through the [AutoSize](#) property;
- Through the [SizeType](#) property on the [TableLayoutPanel](#) control's column and row styles.

The AutoSize Property with Row and Column Styles

The following table describes the interaction between the [AutoSize](#) property and the [TableLayoutPanel](#) control's column and row styles.

AUTOSIZE SETTING	STYLE INTERACTION
<code>false</code>	<p>The TableLayoutPanel control proceeds from left to right, and allocates space for the column or row or in the following order.</p> <ol style="list-style-type: none">1. If the SizeType property is set to Absolute, the number of pixels specified by Width or Height is allocated.2. If the SizeType property is set to AutoSize, the number of pixels returned by the child control's GetPreferredSize method is allocated.3. After space for all Absolute and AutoSize columns or rows is allocated, any columns or rows with SizeType set to Percent are used to proportionally allocate the remaining free space
<code>true</code>	<p>Similar to the previous interaction, with the exception that Percent columns or rows acquire an automatic sizing aspect.</p> <p>The TableLayoutPanel control expands the column or row to create adequate free space, so that no column or row with Percent styling clips its contents. The TableLayoutPanel control allocates the new space proportionally according to the Width or Height property.</p>

See Also

[TableLayoutPanel](#)

[TableLayoutPanel Control Overview](#)

Best Practices for the TableLayoutPanel Control

5/4/2018 • 1 min to read • [Edit Online](#)

The [TableLayoutPanel](#) control provides powerful layout features that you should consider carefully before using on your Windows Forms.

Recommendations

The following recommendations will help you use the [TableLayoutPanel](#) control to its best advantage.

Targeted Use

Use the [TableLayoutPanel](#) control sparingly. You should not use it in all situations that require a resizable layout.

The following list describes layouts that benefit most from the use of the [TableLayoutPanel](#) control:

- Layouts in which there are multiple parts of the form that resize proportionally to each other.
- Layouts that will be modified or generated dynamically at run time, such as data entry forms that have user-customizable fields added or subtracted based on preferences.
- Layouts that should remain at an overall fixed size. For example, you may have a dialog box that should remain smaller than 800 x 600, but you need to support localized strings.

The following list describes layouts that do not benefit greatly from using the [TableLayoutPanel](#) control:

- Simple data entry forms with a single column of labels and a single column of text-entry areas.
- Forms with a single large display area that should fill all the available space when a resize occurs. An example of this is a form that displays a single [PropertyGrid](#) control. In this case, use anchoring, because nothing else should expand when the form is resized.

Choose carefully which controls need to be in a [TableLayoutPanel](#) control. If you have room for your text to grow by 30% using anchoring, consider using the [Anchor](#) property only. If you can estimate the space required by your layout, use of [Dock](#) and [Anchor](#) is easier than estimating the details of remaining space and [AutoSize](#) behavior.

In general, when designing your layout with the [TableLayoutPanel](#) control, keep the design as simple as possible.

Use the Document Outline Window

The Document Outline window gives you a tree view of your layout, which you can use to manipulate the z-order and parent-child relationships of your controls. From the **View menu**, select **Other Windows**, then select **Document Outline**.

Avoid Nesting

Avoid nesting other [TableLayoutPanel](#) controls within a [TableLayoutPanel](#) control. Debugging nested layouts can be difficult.

Avoid Visual Inheritance

The [TableLayoutPanel](#) control does not support visual inheritance in the Windows Forms Designer. A [TableLayoutPanel](#) control in a derived class appears as "locked" at design time.

See Also

[TableLayoutPanel](#)
[FlowLayoutPanel](#)

How to: Align and Stretch a Control in a TableLayoutPanel Control

5/4/2018 • 1 min to read • [Edit Online](#)

You can align and stretch controls in a **TableLayoutPanel** with the **Anchor** and **Dock** properties.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To align and stretch a control

1. Drag a **TableLayoutPanel** control from the **Toolbox** onto your form.
2. Drag a **Button** control from the **Toolbox** into the upper-left cell of the **TableLayoutPanel** control. The **Button** control is centered in the cell.
3. Set the value of the **Button** control's **Anchor** property to `Left,Right`. The **Button** control stretches to match the width of the cell.
4. Set the value of the **Button** control's **Anchor** property to `Top,Bottom`. The **Button** control stretches to match the height of the cell.
5. Set the value of the **Button** control's **Dock** property to **Fill**. The **Button** control expands to fill the cell.
6. Set the value of the **Button** control's **Dock** property to **None**. The **Button** control returns to its original size and moves to the upper-left corner of the cell. The **Windows Forms Designer** has set the **Anchor** property to `Top, Left`.
7. Set the value of the **Button** control's **Anchor** property to `Bottom,Right`. The **Button** control moves to the lower-right corner of the cell.
8. Set the value of the **Button** control's **Anchor** property to **None**. The **Button** control moves to the center of the cell.

See Also

[TableLayoutPanel Control](#)

How to: Anchor and Dock Child Controls in a TableLayoutPanel Control

5/4/2018 • 9 min to read • [Edit Online](#)

The [TableLayoutPanel](#) control supports the [Anchor](#) and [Dock](#) properties in its child controls.

To align a child control in a TableLayoutPanel cell

1. Create a [TableLayoutPanel](#) control on your form.
2. Set the value of the [TableLayoutPanel](#) control's [ColumnCount](#) and [RowCount](#) properties to **1**.
3. Create a [Button](#) control in the [TableLayoutPanel](#) control. The [Button](#) occupies the upper-left corner of the cell.
4. Change the value of the [Button](#) control's [Anchor](#) property to `Left`. The [Button](#) control moves to align with the left border of the cell.

NOTE

This behavior differs from the behavior of other container controls. In other container controls, the child control does not move when the [Anchor](#) property is set, and the distance between the anchored control and the parent container's boundary is fixed at the time the [Anchor](#) property is set.

5. Change the value of the [Button](#) control's [Anchor](#) property to `Top, Left`. The [Button](#) control moves to occupy the top-left corner of the cell.
6. Repeat step 5 with a value of `Top, Right` to move the [Button](#) control to the top-right corner of the cell.
Repeat with values of `Bottom, Left` and `Bottom, Right`.

To stretch a child control in a TableLayoutPanel cell

1. Change the value of the [Button](#) control's [Anchor](#) property to `Left, Right`. The [Button](#) control is resized to stretch across the cell.

NOTE

This behavior differs from the behavior of other container controls. In other container controls, the child control is not resized when the [Anchor](#) property is set to `Left, Right` or `Top, Bottom`.

2. Change the value of the [Button](#) control's [Anchor](#) property to `Top, Bottom`. The [Button](#) control is resized to stretch from the top to the bottom of the cell.
3. Change the value of the [Button](#) control's [Anchor](#) property to `Top, Bottom, Left, Right`. The [Button](#) control is resized to fill the cell.
4. Change the value of the [Button](#) control's [Anchor](#) property to `None`. The [Button](#) control is resized and centered in the cell.
5. Change the value of the [Button](#) control's [Dock](#) property to `Left`. The [Button](#) control moves to align with the left border of the cell. The [Button](#) control retains its width, but its height is resized to fill the cell vertically.

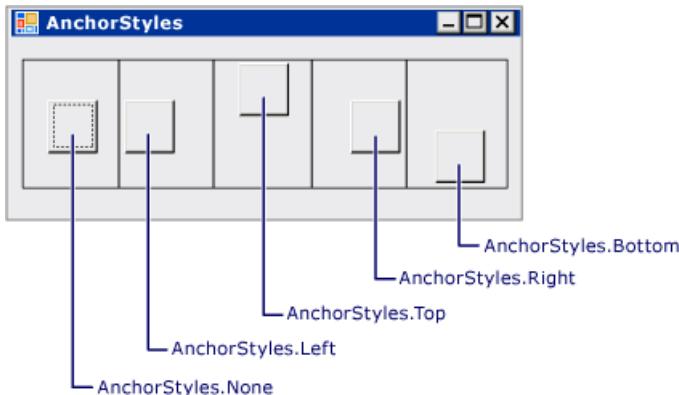
NOTE

This is the same behavior that occurs in other container controls.

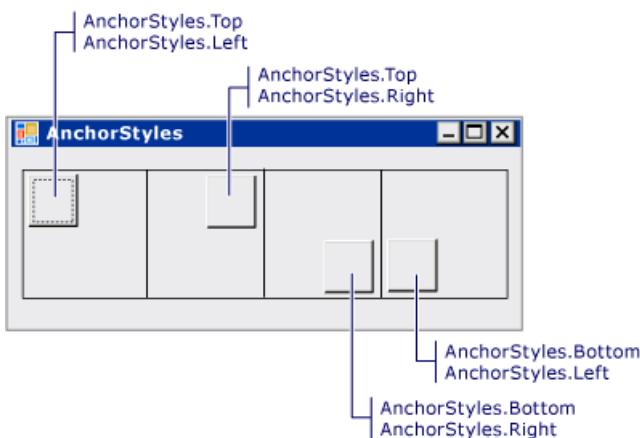
6. Change the value of the **Button** control's **Dock** property to **Fill**. The **Button** control is resized to fill the cell.

Example

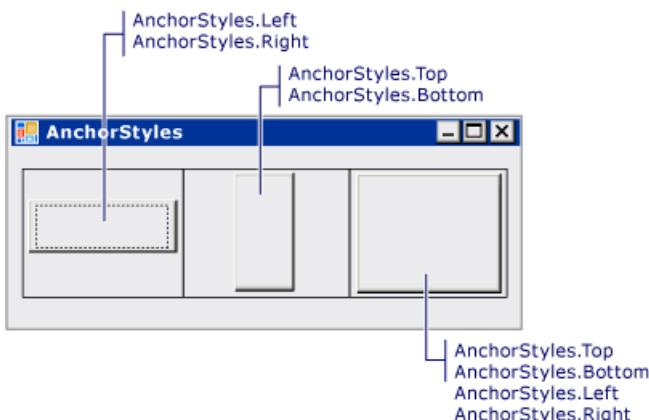
The following illustration shows five buttons anchored in five separate **TableLayoutPanel** cells.



The following illustration shows four buttons anchored in the corners of four separate **TableLayoutPanel** cells.



The following illustration shows three buttons stretched by anchoring in three separate **TableLayoutPanel** cells.



The following code example demonstrates all the combinations of **Anchor** property values for a **Button** control in a **TableLayoutPanel** control.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
```

```
using System.Data;
using System.Drawing;
using System.Windows.Forms;

public class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

    private System.ComponentModel.IContainer components = null;

    private System.Windows.Forms.TableLayoutPanel tableLayoutPanel1;
    private System.Windows.Forms.Button button1;
    private System.Windows.Forms.Button button2;
    private System.Windows.Forms.Button button3;
    private System.Windows.Forms.Button button4;
    private System.Windows.Forms.Button button5;
    private System.Windows.Forms.TableLayoutPanel tableLayoutPanel2;
    private System.Windows.Forms.Button button6;
    private System.Windows.Forms.Button button7;
    private System.Windows.Forms.Button button8;
    private System.Windows.Forms.Button button9;
    private System.Windows.Forms.TableLayoutPanel tableLayoutPanel3;
    private System.Windows.Forms.Button button10;
    private System.Windows.Forms.Button button11;
    private System.Windows.Forms.Button button12;

    protected override void Dispose(bool disposing)
    {
        if (disposing && (components != null))
        {
            components.Dispose();
        }
        base.Dispose(disposing);
    }

    private void InitializeComponent()
    {
        this.tableLayoutPanel1 = new System.Windows.Forms.TableLayoutPanel();
        this.button1 = new System.Windows.Forms.Button();
        this.button2 = new System.Windows.Forms.Button();
        this.button3 = new System.Windows.Forms.Button();
        this.button4 = new System.Windows.Forms.Button();
        this.button5 = new System.Windows.Forms.Button();
        this.tableLayoutPanel2 = new System.Windows.Forms.TableLayoutPanel();
        this.button6 = new System.Windows.Forms.Button();
        this.button7 = new System.Windows.Forms.Button();
        this.button8 = new System.Windows.Forms.Button();
        this.button9 = new System.Windows.Forms.Button();
        this.tableLayoutPanel3 = new System.Windows.Forms.TableLayoutPanel();
        this.button10 = new System.Windows.Forms.Button();
        this.button11 = new System.Windows.Forms.Button();
        this.button12 = new System.Windows.Forms.Button();
        this.tableLayoutPanel1.SuspendLayout();
        this.tableLayoutPanel2.SuspendLayout();
        this.tableLayoutPanel3.SuspendLayout();
        this.SuspendLayout();

        // 
        // tableLayoutPanel1
        // 
        this.tableLayoutPanel1.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Bottom) | System.Windows.Forms.AnchorStyles.Left) | System.Windows.Forms.AnchorStyles.Right));
        this.tableLayoutPanel1.CellBorderStyle = System.Windows.Forms.TableLayoutPanelCellBorderStyle.Single;
        this.tableLayoutPanel1.ColumnCount = 5;
```



```
this.button5.Text = "Anchor=Bottom";
//
// tableLayoutPanel2
//
this.tableLayoutPanel2.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Bottom)
| System.Windows.Forms.AnchorStyles.Left)
| System.Windows.Forms.AnchorStyles.Right));
this.tableLayoutPanel2.CellBorderStyle = System.Windows.Forms.TableLayoutPanelCellBorderStyle.Single;
this.tableLayoutPanel2.ColumnCount = 4;
this.tableLayoutPanel2.ColumnStyles.Add(new
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 25F));
this.tableLayoutPanel2.ColumnStyles.Add(new
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 25F));
this.tableLayoutPanel2.ColumnStyles.Add(new
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 25F));
this.tableLayoutPanel2.ColumnStyles.Add(new
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 25F));
this.tableLayoutPanel2.Controls.Add(this.button6, 0, 0);
this.tableLayoutPanel2.Controls.Add(this.button7, 1, 0);
this.tableLayoutPanel2.Controls.Add(this.button8, 2, 0);
this.tableLayoutPanel2.Controls.Add(this.button9, 3, 0);
this.tableLayoutPanel2.Location = new System.Drawing.Point(12, 118);
this.tableLayoutPanel2.Name = "tableLayoutPanel2";
this.tableLayoutPanel2.RowCount = 1;
this.tableLayoutPanel2.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 50F));
this.tableLayoutPanel2.Size = new System.Drawing.Size(731, 100);
this.tableLayoutPanel2.TabIndex = 1;
//
// button6
//
this.button6.AutoSize = true;
this.button6.Location = new System.Drawing.Point(4, 4);
this.button6.Name = "button6";
this.button6.TabIndex = 0;
this.button6.Text = "Top, Left";
//
// button7
//
this.button7.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Right)));
this.button7.AutoSize = true;
this.button7.Location = new System.Drawing.Point(286, 4);
this.button7.Name = "button7";
this.button7.TabIndex = 1;
this.button7.Text = "Top, Right";
//
// button8
//
this.button8.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Bottom | System.Windows.Forms.AnchorStyles.Right)));
this.button8.AutoSize = true;
this.button8.Location = new System.Drawing.Point(466, 73);
this.button8.Name = "button8";
this.button8.Size = new System.Drawing.Size(77, 23);
this.button8.TabIndex = 2;
this.button8.Text = "Bottom, Right";
//
// button9
//
this.button9.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Bottom | System.Windows.Forms.AnchorStyles.Left)));
this.button9.AutoSize = true;
this.button9.Location = new System.Drawing.Point(550, 73);
this.button9.Name = "button9";
this.button9.TabIndex = 3;
this.button9.Text = "Bottom, Left";
//
```

```
// tableLayoutPanel3
//
this.tableLayoutPanel3.Anchor = ((System.Windows.Forms.AnchorStyles)
((((System.Windows.Forms.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Bottom)
| System.Windows.Forms.AnchorStyles.Left)
| System.Windows.Forms.AnchorStyles.Right)));
this.tableLayoutPanel3.CellBorderStyle = System.Windows.Forms.TableLayoutPanelCellBorderStyle.Single;
this.tableLayoutPanel3.ColumnCount = 3;
this.tableLayoutPanel3.ColumnStyles.Add(new
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 33.3333F));
this.tableLayoutPanel3.ColumnStyles.Add(new
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 33.3333F));
this.tableLayoutPanel3.ColumnStyles.Add(new
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 33.3333F));
this.tableLayoutPanel3.Controls.Add(this.button10, 0, 0);
this.tableLayoutPanel3.Controls.Add(this.button11, 1, 0);
this.tableLayoutPanel3.Controls.Add(this.button12, 2, 0);
this.tableLayoutPanel3.Location = new System.Drawing.Point(12, 225);
this.tableLayoutPanel3.Name = "tableLayoutPanel3";
this.tableLayoutPanel3.RowCount = 1;
this.tableLayoutPanel3.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 50F));
this.tableLayoutPanel3.Size = new System.Drawing.Size(731, 100);
this.tableLayoutPanel3.TabIndex = 2;
//
// button10
//
this.button10.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Left |
System.Windows.Forms.AnchorStyles.Right)));
this.button10.Location = new System.Drawing.Point(4, 39);
this.button10.Name = "button10";
this.button10.Size = new System.Drawing.Size(236, 23);
this.button10.TabIndex = 0;
this.button10.Text = "Left, Right";
//
// button11
//
this.button11.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Bottom)));
this.button11.Location = new System.Drawing.Point(327, 4);
this.button11.Name = "button11";
this.button11.Size = new System.Drawing.Size(75, 93);
this.button11.TabIndex = 1;
this.button11.Text = "Top, Bottom";
//
// button12
//
this.button12.Anchor = ((System.Windows.Forms.AnchorStyles)((((System.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Bottom)
| System.Windows.Forms.AnchorStyles.Left)
| System.Windows.Forms.AnchorStyles.Right)));
this.button12.Location = new System.Drawing.Point(490, 4);
this.button12.Name = "button12";
this.button12.Size = new System.Drawing.Size(237, 93);
this.button12.TabIndex = 2;
this.button12.Text = "Top, Bottom, Left, Right";
//
// Form1
//
this.AutoSize = true;
this.ClientSize = new System.Drawing.Size(755, 338);
this.Controls.Add(this.tableLayoutPanel3);
this.Controls.Add(this.tableLayoutPanel2);
this.Controls.Add(this.tableLayoutPanel1);
this.Name = "Form1";
this.Text = "Form1";
this.tableLayoutPanel1.ResumeLayout(false);
this.tableLayoutPanel1.PerformLayout();
this.tableLayoutPanel2.ResumeLayout(false);
this.tableLayoutPanel2.PerformLayout();
this.tableLayoutPanel3.ResumeLayout(false);
this.tableLayoutPanel3.PerformLayout();
```

```

        childTableLayoutPanel1.ResumeLayout();
        this.tableLayoutPanel2.PerformLayout();
        this.tableLayoutPanel3.ResumeLayout(false);
        this.ResumeLayout(false);

    }

    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.Run(new Form1());
    }
}

```

```

Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Drawing
Imports System.Windows.Forms

Public Class Form1
    Inherits Form

    Public Sub New()
        InitializeComponent()
    End Sub

    Private components As System.ComponentModel.IContainer = Nothing

    Private tableLayoutPanel1 As System.Windows.Forms.TableLayoutPanel
    Private button1 As System.Windows.Forms.Button
    Private button2 As System.Windows.Forms.Button
    Private button3 As System.Windows.Forms.Button
    Private button4 As System.Windows.Forms.Button
    Private button5 As System.Windows.Forms.Button
    Private tableLayoutPanel2 As System.Windows.Forms.TableLayoutPanel
    Private button6 As System.Windows.Forms.Button
    Private button7 As System.Windows.Forms.Button
    Private button8 As System.Windows.Forms.Button
    Private button9 As System.Windows.Forms.Button
    Private tableLayoutPanel3 As System.Windows.Forms.TableLayoutPanel
    Private button10 As System.Windows.Forms.Button
    Private button11 As System.Windows.Forms.Button
    Private button12 As System.Windows.Forms.Button

    Protected Overrides Sub Dispose(disposing As Boolean)
        If disposing AndAlso (components IsNot Nothing) Then
            components.Dispose()
        End If
        MyBase.Dispose(disposing)
    End Sub

    Private Sub InitializeComponent()
        Me.tableLayoutPanel1 = New System.Windows.Forms.TableLayoutPanel()
        Me.button1 = New System.Windows.Forms.Button()
        Me.button2 = New System.Windows.Forms.Button()
        Me.button3 = New System.Windows.Forms.Button()
        Me.button4 = New System.Windows.Forms.Button()
        Me.button5 = New System.Windows.Forms.Button()
        Me.tableLayoutPanel2 = New System.Windows.Forms.TableLayoutPanel()
        Me.button6 = New System.Windows.Forms.Button()
        Me.button7 = New System.Windows.Forms.Button()
        Me.button8 = New System.Windows.Forms.Button()
        Me.button9 = New System.Windows.Forms.Button()

```

```
Me.tableLayoutPanel3 = New System.Windows.Forms.TableLayoutPanel()
Me.button10 = New System.Windows.Forms.Button()
Me.button11 = New System.Windows.Forms.Button()
Me.button12 = New System.Windows.Forms.Button()
Me.tableLayoutPanel1.SuspendLayout()
Me.tableLayoutPanel2.SuspendLayout()
Me.tableLayoutPanel3.SuspendLayout()
Me.SuspendLayout()
'
' tableLayoutPanel1
'

Me.tableLayoutPanel1.Anchor = CType(System.Windows.Forms.AnchorStyles.Top Or
System.Windows.Forms.AnchorStyles.Bottom Or System.Windows.Forms.AnchorStyles.Left Or
System.Windows.Forms.AnchorStyles.Right, System.Windows.Forms.AnchorStyles)
Me.tableLayoutPanel1.CellBorderStyle = System.Windows.Forms.TableLayoutPanelCellBorderStyle.Single
Me.tableLayoutPanel1.ColumnCount = 5
Me.tableLayoutPanel1.ColumnStyles.Add(New
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 20F))
Me.tableLayoutPanel1.Controls.Add(Me.button1, 0, 0)
Me.tableLayoutPanel1.Controls.Add(Me.button2, 1, 0)
Me.tableLayoutPanel1.Controls.Add(Me.button3, 2, 0)
Me.tableLayoutPanel1.Controls.Add(Me.button4, 3, 0)
Me.tableLayoutPanel1.Controls.Add(Me.button5, 4, 0)
Me.tableLayoutPanel1.Location = New System.Drawing.Point(12, 12)
Me.tableLayoutPanel1.Name = "tableLayoutPanel1"
Me.tableLayoutPanel1.RowCount = 1
Me.tableLayoutPanel1.RowStyles.Add(New
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 50F))
Me.tableLayoutPanel1.Size = New System.Drawing.Size(731, 100)
Me.tableLayoutPanel1.TabIndex = 0
'
' button1
'

Me.button1.Anchor = System.Windows.Forms.AnchorStyles.None
Me.button1.AutoSize = True
Me.button1.Location = New System.Drawing.Point(34, 38)
Me.button1.Name = "button1"
Me.button1.Size = New System.Drawing.Size(79, 23)
Me.button1.TabIndex = 0
Me.button1.Text = "Anchor=None"
'
' button2
'

Me.button2.Anchor = System.Windows.Forms.AnchorStyles.Left
Me.button2.AutoSize = True
Me.button2.Location = New System.Drawing.Point(150, 38)
Me.button2.Name = "button2"
Me.button2.TabIndex = 1
Me.button2.Text = "Anchor=Left"
'
' button3
'

Me.button3.Anchor = System.Windows.Forms.AnchorStyles.Top
Me.button3.AutoSize = True
Me.button3.Location = New System.Drawing.Point(328, 4)
Me.button3.Name = "button3"
Me.button3.TabIndex = 2
Me.button3.Text = "Anchor=Top"
'
' button4
'
```

```
Me.button4.Anchor = System.Windows.Forms.AnchorStyles.Right
Me.button4.AutoSize = True
Me.button4.Location = New System.Drawing.Point(503, 38)
Me.button4.Name = "button4"
Me.button4.Size = New System.Drawing.Size(78, 23)
Me.button4.TabIndex = 3
Me.button4.Text = "Anchor=Right"
'

' button5
'

Me.button5.Anchor = System.Windows.Forms.AnchorStyles.Bottom
Me.button5.AutoSize = True
Me.button5.Location = New System.Drawing.Point(614, 73)
Me.button5.Name = "button5"
Me.button5.Size = New System.Drawing.Size(86, 23)
Me.button5.TabIndex = 4
Me.button5.Text = "Anchor=Bottom"
'

' tableLayoutPanel2
'

Me.tableLayoutPanel2.Anchor = CType(System.Windows.Forms.AnchorStyles.Top Or
System.Windows.Forms.AnchorStyles.Bottom Or System.Windows.Forms.AnchorStyles.Left Or
System.Windows.Forms.AnchorStyles.Right, System.Windows.Forms.AnchorStyles)
Me.tableLayoutPanel2.CellBorderStyle = System.Windows.Forms.TableLayoutPanelCellBorderStyle.Single
Me.tableLayoutPanel2.ColumnCount = 4
Me.tableLayoutPanel2.ColumnStyles.Add(New
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 25F))
Me.tableLayoutPanel2.ColumnStyles.Add(New
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 25F))
Me.tableLayoutPanel2.ColumnStyles.Add(New
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 25F))
Me.tableLayoutPanel2.ColumnStyles.Add(New
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 25F))
Me.tableLayoutPanel2.Controls.Add(Me.button6, 0, 0)
Me.tableLayoutPanel2.Controls.Add(Me.button7, 1, 0)
Me.tableLayoutPanel2.Controls.Add(Me.button8, 2, 0)
Me.tableLayoutPanel2.Controls.Add(Me.button9, 3, 0)
Me.tableLayoutPanel2.Location = New System.Drawing.Point(12, 118)
Me.tableLayoutPanel2.Name = "tableLayoutPanel2"
Me.tableLayoutPanel2.RowCount = 1
Me.tableLayoutPanel2.RowStyles.Add(New
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 50F))
Me.tableLayoutPanel2.Size = New System.Drawing.Size(731, 100)
Me.tableLayoutPanel2.TabIndex = 1
'

' button6
'

Me.button6.AutoSize = True
Me.button6.Location = New System.Drawing.Point(4, 4)
Me.button6.Name = "button6"
Me.button6.TabIndex = 0
Me.button6.Text = "Top, Left"
'

' button7
'

Me.button7.Anchor = CType(System.Windows.Forms.AnchorStyles.Top Or
System.Windows.Forms.AnchorStyles.Right, System.Windows.Forms.AnchorStyles)
Me.button7.AutoSize = True
Me.button7.Location = New System.Drawing.Point(286, 4)
Me.button7.Name = "button7"
Me.button7.TabIndex = 1
Me.button7.Text = "Top, Right"
'

' button8
'

Me.button8.Anchor = CType(System.Windows.Forms.AnchorStyles.Bottom Or
System.Windows.Forms.AnchorStyles.Right, System.Windows.Forms.AnchorStyles)
Me.button8.AutoSize = True
Me.button8.Location = New System.Drawing.Point(503, 73)
```

```
Me.button8.Location = New System.Drawing.Point(466, 13)
Me.button8.Name = "button8"
Me.button8.Size = New System.Drawing.Size(77, 23)
Me.button8.TabIndex = 2
Me.button8.Text = "Bottom, Right"
'
' button9
'
Me.button9.Anchor = CType(System.Windows.Forms.AnchorStyles.Bottom Or
System.Windows.Forms.AnchorStyles.Left, System.Windows.Forms.AnchorStyles)
Me.button9.AutoSize = True
Me.button9.Location = New System.Drawing.Point(550, 73)
Me.button9.Name = "button9"
Me.button9.TabIndex = 3
Me.button9.Text = "Bottom, Left"
'
' tableLayoutPanel3
'
Me.tableLayoutPanel3.Anchor = CType(System.Windows.Forms.AnchorStyles.Top Or
System.Windows.Forms.AnchorStyles.Bottom Or System.Windows.Forms.AnchorStyles.Left Or
System.Windows.Forms.AnchorStyles.Right, System.Windows.Forms.AnchorStyles)
Me.tableLayoutPanel3.CellBorderStyle = System.Windows.Forms.TableLayoutPanelCellBorderStyle.Single
Me.tableLayoutPanel3.ColumnCount = 3
Me.tableLayoutPanel3.ColumnStyles.Add(New
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 33.33333F))
Me.tableLayoutPanel3.ColumnStyles.Add(New
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 33.33333F))
Me.tableLayoutPanel3.ColumnStyles.Add(New
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 33.33333F))
Me.tableLayoutPanel3.Controls.Add(Me.button10, 0, 0)
Me.tableLayoutPanel3.Controls.Add(Me.button11, 1, 0)
Me.tableLayoutPanel3.Controls.Add(Me.button12, 2, 0)
Me.tableLayoutPanel3.Location = New System.Drawing.Point(12, 225)
Me.tableLayoutPanel3.Name = "tableLayoutPanel3"
Me.tableLayoutPanel3.RowCount = 1
Me.tableLayoutPanel3.RowStyles.Add(New
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 50F))
Me.tableLayoutPanel3.Size = New System.Drawing.Size(731, 100)
Me.tableLayoutPanel3.TabIndex = 2
'
' button10
'
Me.button10.Anchor = CType(System.Windows.Forms.AnchorStyles.Left Or
System.Windows.Forms.AnchorStyles.Right, System.Windows.Forms.AnchorStyles)
Me.button10.Location = New System.Drawing.Point(4, 39)
Me.button10.Name = "button10"
Me.button10.Size = New System.Drawing.Size(236, 23)
Me.button10.TabIndex = 0
Me.button10.Text = "Left, Right"
'
' button11
'
Me.button11.Anchor = CType(System.Windows.Forms.AnchorStyles.Top Or
System.Windows.Forms.AnchorStyles.Bottom, System.Windows.Forms.AnchorStyles)
Me.button11.Location = New System.Drawing.Point(327, 4)
Me.button11.Name = "button11"
Me.button11.Size = New System.Drawing.Size(75, 93)
Me.button11.TabIndex = 1
Me.button11.Text = "Top, Bottom"
'
' button12
'
Me.button12.Anchor = CType(System.Windows.Forms.AnchorStyles.Top Or
System.Windows.Forms.AnchorStyles.Bottom Or System.Windows.Forms.AnchorStyles.Left Or
System.Windows.Forms.AnchorStyles.Right, System.Windows.Forms.AnchorStyles)
Me.button12.Location = New System.Drawing.Point(490, 4)
Me.button12.Name = "button12"
Me.button12.Size = New System.Drawing.Size(237, 93)
Me.button12.TabIndex = 2
```

```

Me.button12.Text = "Top, Bottom, Left, Right"
'
' Form1
'

Me.AutoSize = True
Me.ClientSize = New System.Drawing.Size(755, 338)
Me.Controls.Add(tableLayoutPanel3)
Me.Controls.Add(tableLayoutPanel2)
Me.Controls.Add(tableLayoutPanel1)
Me.Name = "Form1"
Me.Text = "Form1"
Me.tableLayoutPanel1.ResumeLayout(False)
Me.tableLayoutPanel1.PerformLayout()
Me.tableLayoutPanel2.ResumeLayout(False)
Me.tableLayoutPanel2.PerformLayout()
Me.tableLayoutPanel3.ResumeLayout(False)
Me.ResumeLayout(False)
End Sub

<STAThread()> _
Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())
End Sub
End Class

```

Compiling the Code

This example requires:

- References to the System, System.Data, System.Drawing and System.Windows.Forms assemblies.

For information about building this example from the command line for visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[TableLayoutPanel](#)
[TableLayoutPanel Control](#)

How to: Create a Resizable Windows Form for Data Entry

5/4/2018 • 15 min to read • [Edit Online](#)

A good layout responds well to changes in the dimensions of its parent form. You can use the [TableLayoutPanel](#) control to arrange the layout of your form to resize and position your controls in a consistent way as the form's dimensions change. The [TableLayoutPanel](#) control is also useful when changes in the contents of your controls cause changes in the layout. The process covered in this procedure can be done within the Visual Studio environment. Also see [Walkthrough: Creating a Resizable Windows Form for Data Entry](#).

Example

The following example demonstrates how to use a [TableLayoutPanel](#) control to build a layout that responds well when the user resizes the form. It also demonstrates a layout that responds well to localization.

```
#using <System.dll>
#using <System.Data.dll>
#using <System.Drawing.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Collections::Generic;
using namespace System::ComponentModel;
using namespace System::Data;
using namespace System::Drawing;
using namespace System::Text;
using namespace System::Windows::Forms;

// This form demonstrates how to build a form layout that adjusts well
// when the user resizes the form. It also demonstrates a layout that
// responds well to localization.
ref class BasicDataEntryForm : public System::Windows::Forms::Form
{
public:
    BasicDataEntryForm()
    {
        InitializeComponent();
        components = nullptr;
    }

private:
    System::ComponentModel::.IContainer^ components;

protected:
    ~BasicDataEntryForm()
    {
        if (components != nullptr)
        {
            delete components;
        }
    }

public:
    virtual String^ ToString() override
    {
        return "Basic Data Entry Form";
    }
}
```

```

private:
    void okBtn_Click(Object^ sender, EventArgs^ e)
    {
        this->Close();
    }

private:
    void cancelBtn_Click(Object^ sender, EventArgs^ e)
    {
        this->Close();
    }

private:
    void InitializeComponent()
    {
        this->tableLayoutPanel1 = gcnew
            System::Windows::Forms::TableLayoutPanel();
        this->lblFirstName = gcnew System::Windows::Forms::Label();
        this->lblLastName = gcnew System::Windows::Forms::Label();
        this->lblAddress1 = gcnew System::Windows::Forms::Label();
        this->lblAddress2 = gcnew System::Windows::Forms::Label();
        this->lblCity = gcnew System::Windows::Forms::Label();
        this->lblState = gcnew System::Windows::Forms::Label();
        this->lblPhoneH = gcnew System::Windows::Forms::Label();
        this->txtAddress1 = gcnew System::Windows::Forms::TextBox();
        this->txtAddress2 = gcnew System::Windows::Forms::TextBox();
        this->txtCity = gcnew System::Windows::Forms::TextBox();
        this->txtLastName = gcnew System::Windows::Forms::TextBox();
        this->maskedTxtPhoneW = gcnew System::Windows::Forms::MaskedTextBox();
        this->maskedTxtPhoneH = gcnew System::Windows::Forms::MaskedTextBox();
        this->cboState = gcnew System::Windows::Forms::ComboBox();
        this->txtFirstName = gcnew System::Windows::Forms::TextBox();
        this->lblNotes = gcnew System::Windows::Forms::Label();
        this->lblPhoneW = gcnew System::Windows::Forms::Label();
        this->richTxtNotes = gcnew System::Windows::Forms::RichTextBox();
        this->cancelBtn = gcnew System::Windows::Forms::Button();
        this->okBtn = gcnew System::Windows::Forms::Button();
        this->tableLayoutPanel1->SuspendLayout();
        this->SuspendLayout();
        //
        // tableLayoutPanel1
        //
        this->tableLayoutPanel1->Anchor =
            System::Windows::Forms::AnchorStyles::Top |
            System::Windows::Forms::AnchorStyles::Bottom |
            System::Windows::Forms::AnchorStyles::Left |
            System::Windows::Forms::AnchorStyles::Right;
        this->tableLayoutPanel1->ColumnCount = 4;
        this->tableLayoutPanel1->ColumnStyles->Add(gcnew
            System::Windows::Forms::ColumnStyle());
        this->tableLayoutPanel1->ColumnStyles->Add(gcnew
            System::Windows::Forms::ColumnStyle(
                System::Windows::Forms::SizeType::Percent, 50.0));
        this->tableLayoutPanel1->ColumnStyles->Add(gcnew
            System::Windows::Forms::ColumnStyle(
                System::Windows::Forms::SizeType::Percent, 50.0));
        this->tableLayoutPanel1->Controls->Add(this->lblFirstName, 0, 0);
        this->tableLayoutPanel1->Controls->Add(this->lblLastName, 2, 0);
        this->tableLayoutPanel1->Controls->Add(this->lblAddress1, 0, 1);
        this->tableLayoutPanel1->Controls->Add(this->lblAddress2, 0, 2);
        this->tableLayoutPanel1->Controls->Add(this->lblCity, 0, 3);
        this->tableLayoutPanel1->Controls->Add(this->lblState, 2, 3);
        this->tableLayoutPanel1->Controls->Add(this->lblPhoneH, 2, 4);
        this->tableLayoutPanel1->Controls->Add(this->txtAddress1, 1, 1);
        this->tableLayoutPanel1->Controls->Add(this->txtAddress2, 1, 2);
        this->tableLayoutPanel1->Controls->Add(this->txtCity, 1, 3);
        this->tableLayoutPanel1->Controls->Add(this->txtLastName, 3, 0);
    }

```

```
this->tableLayoutPanel1->Controls->Add(this->maskedTxtPhoneW, 1, 4);
this->tableLayoutPanel1->Controls->Add(this->maskedTxtPhoneH, 3, 4);
this->tableLayoutPanel1->Controls->Add(this-> cboState, 3, 3);
this->tableLayoutPanel1->Controls->Add(this->txtFirstName, 1, 0);
this->tableLayoutPanel1->Controls->Add(this->lblNotes, 0, 5);
this->tableLayoutPanel1->Controls->Add(this->lblPhoneW, 0, 4);
this->tableLayoutPanel1->Controls->Add(this->richTxtNotes, 1, 5);
this->tableLayoutPanel1->Location = System::Drawing::Point(13, 13);
this->tableLayoutPanel1->Name = "tableLayoutPanel1";
this->tableLayoutPanel1->RowCount = 6;
this->tableLayoutPanel1->RowStyles->Add(gcnew
    System::Windows::Forms::RowStyle(
        System::Windows::Forms::SizeType::Absolute, 28.0));
this->tableLayoutPanel1->RowStyles->Add(gcnew
    System::Windows::Forms::RowStyle(
        System::Windows::Forms::SizeType::Percent, 80.0));
this->tableLayoutPanel1->RowStyles->Add(gcnew
    System::Windows::Forms::RowStyle(
        System::Windows::Forms::SizeType::Absolute, 20.0));
this->tableLayoutPanel1->Size = System::Drawing::Size(623, 286);
this->tableLayoutPanel1->TabIndex = 0;
//
// lblFirstName
//
this->lblFirstName->Anchor =
    System::Windows::Forms::AnchorStyles::Right;
this->lblFirstName->AutoSize = true;
this->lblFirstName->Location = System::Drawing::Point(3, 7);
this->lblFirstName->Name = "lblFirstName";
this->lblFirstName->Size = System::Drawing::Size(59, 14);
this->lblFirstName->TabIndex = 20;
this->lblFirstName->Text = "First Name";
//
// lblLastName
//
this->lblLastName->Anchor =
    System::Windows::Forms::AnchorStyles::Right;
this->lblLastName->AutoSize = true;
this->lblLastName->Location = System::Drawing::Point(323, 7);
this->lblLastName->Name = "lblLastName";
this->lblLastName->Size = System::Drawing::Size(59, 14);
this->lblLastName->TabIndex = 21;
this->lblLastName->Text = "Last Name";
//
// lblAddress1
//
this->lblAddress1->Anchor =
    System::Windows::Forms::AnchorStyles::Right;
this->lblAddress1->AutoSize = true;
this->lblAddress1->Location = System::Drawing::Point(10, 35);
this->lblAddress1->Name = "lblAddress1";
this->lblAddress1->Size = System::Drawing::Size(52, 14);
this->lblAddress1->TabIndex = 22;
this->lblAddress1->Text = "Address1";
//
// lblAddress2
//
```

```
this->lblAddress2->Anchor =
    System::Windows::Forms::AnchorStyles::Right;
this->lblAddress2->AutoSize = true;
this->lblAddress2->Location = System::Drawing::Point(7, 63);
this->lblAddress2->Name = "lblAddress2";
this->lblAddress2->Size = System::Drawing::Size(55, 14);
this->lblAddress2->TabIndex = 23;
this->lblAddress2->Text = "Address 2";
//
// lblCity
//
this->lblCity->Anchor =
    System::Windows::Forms::AnchorStyles::Right;
this->lblCity->AutoSize = true;
this->lblCity->Location = System::Drawing::Point(38, 91);
this->lblCity->Name = "lblCity";
this->lblCity->Size = System::Drawing::Size(24, 14);
this->lblCity->TabIndex = 24;
this->lblCity->Text = "City";
//
// lblState
//
this->lblState->Anchor =
    System::Windows::Forms::AnchorStyles::Right;
this->lblState->AutoSize = true;
this->lblState->Location = System::Drawing::Point(351, 91);
this->lblState->Name = "lblState";
this->lblState->Size = System::Drawing::Size(31, 14);
this->lblState->TabIndex = 25;
this->lblState->Text = "State";
//
// lblPhoneH
//
this->lblPhoneH->Anchor =
    System::Windows::Forms::AnchorStyles::Right;
this->lblPhoneH->AutoSize = true;
this->lblPhoneH->Location = System::Drawing::Point(326, 119);
this->lblPhoneH->Name = "lblPhoneH";
this->lblPhoneH->Size = System::Drawing::Size(56, 14);
this->lblPhoneH->TabIndex = 33;
this->lblPhoneH->Text = "Phone (H)";
//
// txtAddress1
//
this->txtAddress1->Anchor =
    System::Windows::Forms::AnchorStyles::Left |
    System::Windows::Forms::AnchorStyles::Right;
this->tableLayoutPanel1->SetColumnSpan(this->txtAddress1, 3);
this->txtAddress1->Location = System::Drawing::Point(68, 32);
this->txtAddress1->Name = "txtAddress1";
this->txtAddress1->Size = System::Drawing::Size(552, 20);
this->txtAddress1->TabIndex = 2;
//
// txtAddress2
//
this->txtAddress2->Anchor =
    System::Windows::Forms::AnchorStyles::Left |
    System::Windows::Forms::AnchorStyles::Right;
this->tableLayoutPanel1->SetColumnSpan(this->txtAddress2, 3);
this->txtAddress2->Location = System::Drawing::Point(68, 60);
this->txtAddress2->Name = "txtAddress2";
this->txtAddress2->Size = System::Drawing::Size(552, 20);
this->txtAddress2->TabIndex = 3;
//
// txtCity
//
this->txtCity->Anchor =
    System::Windows::Forms::AnchorStyles::Left |
    System::Windows::Forms::AnchorStyles::Right;
```

```
this->txtCity->Location = System::Drawing::Point(68, 88);
this->txtCity->Name = "txtCity";
this->txtCity->Size = System::Drawing::Size(249, 20);
this->txtCity->TabIndex = 4;
//
// txtLastName
//
this->txtLastName->Anchor =
    System::Windows::Forms::AnchorStyles::Left |
    System::Windows::Forms::AnchorStyles::Right;
this->txtLastName->Location = System::Drawing::Point(388, 4);
this->txtLastName->Name = "txtLastName";
this->txtLastName->Size = System::Drawing::Size(232, 20);
this->txtLastName->TabIndex = 1;
//
// maskedTxtPhoneW
//
this->maskedTxtPhoneW->Anchor =
    System::Windows::Forms::AnchorStyles::Left;
this->maskedTxtPhoneW->Location = System::Drawing::Point(68, 116);
this->maskedTxtPhoneW->Mask = "(999)000-0000";
this->maskedTxtPhoneW->Name = "maskedTxtPhoneW";
this->maskedTxtPhoneW->TabIndex = 6;
//
// maskedTxtPhoneH
//
this->maskedTxtPhoneH->Anchor =
    System::Windows::Forms::AnchorStyles::Left;
this->maskedTxtPhoneH->Location = System::Drawing::Point(388, 116);
this->maskedTxtPhoneH->Mask = "(999)000-0000";
this->maskedTxtPhoneH->Name = "maskedTxtPhoneH";
this->maskedTxtPhoneH->TabIndex = 7;
//
// cboState
//
this->cboState->Anchor = System::Windows::Forms::AnchorStyles::Left;
this->cboState->FormattingEnabled = true;
this->cboState->Items->AddRange(gcnew array<Object^> {
    "AK - Alaska",
    "WA - Washington"});
this->cboState->Location = System::Drawing::Point(388, 87);
this->cboState->Name = "cboState";
this->cboState->Size = System::Drawing::Size(100, 21);
this->cboState->TabIndex = 5;
//
// txtFirstName
//
this->txtFirstName->Anchor =
    System::Windows::Forms::AnchorStyles::Left |
    System::Windows::Forms::AnchorStyles::Right;
this->txtFirstName->Location = System::Drawing::Point(68, 4);
this->txtFirstName->Name = "txtFirstName";
this->txtFirstName->Size = System::Drawing::Size(249, 20);
this->txtFirstName->TabIndex = 0;
//
// lblNotes
//
this->lblNotes->Anchor =
    System::Windows::Forms::AnchorStyles::Top |
    System::Windows::Forms::AnchorStyles::Right;
this->lblNotes->AutoSize = true;
this->lblNotes->Location = System::Drawing::Point(28, 143);
this->lblNotes->Name = "lblNotes";
this->lblNotes->Size = System::Drawing::Size(34, 14);
this->lblNotes->TabIndex = 26;
this->lblNotes->Text = "Notes";
//
// lblPhoneW
//
```

```

    //
    this->lblPhoneW->Anchor =
        System::Windows::Forms::AnchorStyles::Right;
    this->lblPhoneW->AutoSize = true;
    this->lblPhoneW->Location = System::Drawing::Point(4, 119);
    this->lblPhoneW->Name = "lblPhoneW";
    this->lblPhoneW->Size = System::Drawing::Size(58, 14);
    this->lblPhoneW->TabIndex = 32;
    this->lblPhoneW->Text = "Phone (W)";
    //
    // richTxtNotes
    //
    this->tableLayoutPanel1->SetColumnSpan(this->richTxtNotes, 3);
    this->richTxtNotes->Dock = System::Windows::Forms::DockStyle::Fill;
    this->richTxtNotes->Location = System::Drawing::Point(68, 143);
    this->richTxtNotes->Name = "richTxtNotes";
    this->richTxtNotes->Size = System::Drawing::Size(552, 140);
    this->richTxtNotes->TabIndex = 8;
    this->richTxtNotes->Text = "";
    //
    // cancelBtn
    //
    this->cancelBtn->Anchor =
        System::Windows::Forms::AnchorStyles::Bottom |
        System::Windows::Forms::AnchorStyles::Right;
    this->cancelBtn->DialogResult =
        System::Windows::Forms::DialogResult::Cancel;
    this->cancelBtn->Location = System::Drawing::Point(558, 306);
    this->cancelBtn->Name = "cancelBtn";
    this->cancelBtn->TabIndex = 1;
    this->cancelBtn->Text = "Cancel";
    this->cancelBtn->Click += gcnew System::EventHandler(
        this, &BasicDataEntryForm::cancelBtn_Click);
    //
    // okBtn
    //
    this->okBtn->Anchor =
        System::Windows::Forms::AnchorStyles::Bottom |
        System::Windows::Forms::AnchorStyles::Right;
    this->okBtn->DialogResult =
        System::Windows::Forms::DialogResult::OK;
    this->okBtn->Location = System::Drawing::Point(476, 306);
    this->okBtn->Name = "okBtn";
    this->okBtn->TabIndex = 0;
    this->okBtn->Text = "OK";
    this->okBtn->Click += gcnew System::EventHandler(
        this, &BasicDataEntryForm::okBtn_Click);
    //
    // BasicDataEntryForm
    //
    this->AutoScaleBaseSize = System::Drawing::Size(5, 13);
    this->ClientSize = System::Drawing::Size(642, 338);
    this->Controls->Add(this->okBtn);
    this->Controls->Add(this->cancelBtn);
    this->Controls->Add(this->tableLayoutPanel1);
    this->Name = "BasicDataEntryForm";
    this->Padding = System::Windows::Forms::Padding(9);
    this->StartPosition =
        System::Windows::Forms::FormStartPosition::Manual;
    this->Text = "Basic Data Entry";
    this->tableLayoutPanel1->ResumeLayout(false);
    this->tableLayoutPanel1->PerformLayout();
    this->ResumeLayout(false);

}

private:
    System::Windows::Forms::TableLayoutPanel^ tableLayoutPanel1;
    System::Windows::Forms::Label^ lblFirstName;

```

```

System::Windows::Forms::Label^ lblLastName;
System::Windows::Forms::Label^ lblAddress1;
System::Windows::Forms::Label^ lblAddress2;
System::Windows::Forms::Label^ lblCity;
System::Windows::Forms::Label^ lblState;
System::Windows::Forms::Label^ lblNotes;
System::Windows::Forms::Label^ lblPhoneW;
System::Windows::Forms::Label^ lblPhoneH;
System::Windows::Forms::Button^ cancelBtn;
System::Windows::Forms::Button^ okBtn;
System::Windows::Forms::TextBox^ txtFirstName;
System::Windows::Forms::TextBox^ txtAddress1;
System::Windows::Forms::TextBox^ txtAddress2;
System::Windows::Forms::TextBox^ txtCity;
System::Windows::Forms::TextBox^ txtLastName;
System::Windows::Forms::MaskedTextBox^ maskedTxtPhoneW;
System::Windows::Forms::MaskedTextBox^ maskedTxtPhoneH;
System::Windows::Forms::ComboBox^ cboState;
System::Windows::Forms::RichTextBox^ richTxtNotes;
};

[STAThread]
int main()
{
    Application::EnableVisualStyles();
    Application::Run(gcnew BasicDataEntryForm());
}

```

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

// This form demonstrates how to build a form layout that adjusts well
// when the user resizes the form. It also demonstrates a layout that
// responds well to localization.
class BasicDataEntryForm : System.Windows.Forms.Form
{
    public BasicDataEntryForm()
    {
        InitializeComponent();
    }

    private System.ComponentModel.IContainer components = null;

    protected override void Dispose(bool disposing)
    {
        if (disposing && (components != null))
        {
            components.Dispose();
        }
        base.Dispose(disposing);
    }

    public override string ToString()
    {
        return "Basic Data Entry Form";
    }

    private void okBtn_Click(object sender, EventArgs e)
    {
        this.Close();
    }
}

```

```

private void cancelBtn_Click(object sender, EventArgs e)
{
    this.Close();
}

private void InitializeComponent()
{
    this.tableLayoutPanel1 = new System.Windows.Forms.TableLayoutPanel();
    this.label1 = new System.Windows.Forms.Label();
    this.label2 = new System.Windows.Forms.Label();
    this.label3 = new System.Windows.Forms.Label();
    this.label4 = new System.Windows.Forms.Label();
    this.label5 = new System.Windows.Forms.Label();
    this.label6 = new System.Windows.Forms.Label();
    this.label9 = new System.Windows.Forms.Label();
    this.textBox2 = new System.Windows.Forms.TextBox();
    this.textBox3 = new System.Windows.Forms.TextBox();
    this.textBox4 = new System.Windows.Forms.TextBox();
    this.textBox5 = new System.Windows.Forms.TextBox();
    this.maskedTextBox1 = new System.Windows.Forms.MaskedTextBox();
    this.maskedTextBox2 = new System.Windows.Forms.MaskedTextBox();
    this.comboBox1 = new System.Windows.Forms.ComboBox();
    this.textBox1 = new System.Windows.Forms.TextBox();
    this.label7 = new System.Windows.Forms.Label();
    this.label8 = new System.Windows.Forms.Label();
    this.richTextBox1 = new System.Windows.Forms.RichTextBox();
    this.cancelBtn = new System.Windows.Forms.Button();
    this.okBtn = new System.Windows.Forms.Button();
    this.tableLayoutPanel1.SuspendLayout();
    this.SuspendLayout();

    // 
    // tableLayoutPanel1
    // 
    this.tableLayoutPanel1.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Bottom) | System.Windows.Forms.AnchorStyles.Left) | System.Windows.Forms.AnchorStyles.Right));
    this.tableLayoutPanel1.ColumnCount = 4;
    this.tableLayoutPanel1.ColumnStyles.Add(new System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 50F));
    this.tableLayoutPanel1.ColumnStyles.Add(new System.Windows.Forms.ColumnStyle());
    this.tableLayoutPanel1.ColumnStyles.Add(new System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 50F));
    this.tableLayoutPanel1.ColumnStyles.Add(new System.Windows.Forms.ColumnStyle());
    this.tableLayoutPanel1.Controls.Add(this.label1, 0, 0);
    this.tableLayoutPanel1.Controls.Add(this.label2, 2, 0);
    this.tableLayoutPanel1.Controls.Add(this.label3, 0, 1);
    this.tableLayoutPanel1.Controls.Add(this.label4, 0, 2);
    this.tableLayoutPanel1.Controls.Add(this.label5, 0, 3);
    this.tableLayoutPanel1.Controls.Add(this.label6, 2, 3);
    this.tableLayoutPanel1.Controls.Add(this.label9, 2, 4);
    this.tableLayoutPanel1.Controls.Add(this.textBox2, 1, 1);
    this.tableLayoutPanel1.Controls.Add(this.textBox3, 1, 2);
    this.tableLayoutPanel1.Controls.Add(this.textBox4, 1, 3);
    this.tableLayoutPanel1.Controls.Add(this.textBox5, 3, 0);
    this.tableLayoutPanel1.Controls.Add(this.maskedTextBox1, 1, 4);
    this.tableLayoutPanel1.Controls.Add(this.maskedTextBox2, 3, 4);
    this.tableLayoutPanel1.Controls.Add(this.comboBox1, 3, 3);
    this.tableLayoutPanel1.Controls.Add(this.textBox1, 1, 0);
    this.tableLayoutPanel1.Controls.Add(this.label7, 0, 5);
    this.tableLayoutPanel1.Controls.Add(this.label8, 0, 4);
    this.tableLayoutPanel1.Controls.Add(this.richTextBox1, 1, 5);
    this.tableLayoutPanel1.Location = new System.Drawing.Point(13, 13);
    this.tableLayoutPanel1.Name = "tableLayoutPanel1";
    this.tableLayoutPanel1.RowCount = 6;
    this.tableLayoutPanel1.RowStyles.Add(new

```

```
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Absolute, 28F));
    this.tableLayoutPanel1.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Absolute, 28F));
        this.tableLayoutPanel1.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Absolute, 28F));
            this.tableLayoutPanel1.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Absolute, 28F));
                this.tableLayoutPanel1.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 80F));
                    this.tableLayoutPanel1.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Absolute, 20F));
                        this.tableLayoutPanel1.Size = new System.Drawing.Size(623, 286);
                        this.tableLayoutPanel1.TabIndex = 0;
//
// label1
//
this.label1.Anchor = System.Windows.Forms.AnchorStyles.Right;
this.label1.AutoSize = true;
this.label1.Location = new System.Drawing.Point(3, 7);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(59, 14);
this.label1.TabIndex = 20;
this.label1.Text = "First Name";
//
// label2
//
this.label2.Anchor = System.Windows.Forms.AnchorStyles.Right;
this.label2.AutoSize = true;
this.label2.Location = new System.Drawing.Point(323, 7);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(59, 14);
this.label2.TabIndex = 21;
this.label2.Text = "Last Name";
//
// label3
//
this.label3.Anchor = System.Windows.Forms.AnchorStyles.Right;
this.label3.AutoSize = true;
this.label3.Location = new System.Drawing.Point(10, 35);
this.label3.Name = "label3";
this.label3.Size = new System.Drawing.Size(52, 14);
this.label3.TabIndex = 22;
this.label3.Text = "Address1";
//
// label4
//
this.label4.Anchor = System.Windows.Forms.AnchorStyles.Right;
this.label4.AutoSize = true;
this.label4.Location = new System.Drawing.Point(7, 63);
this.label4.Name = "label4";
this.label4.Size = new System.Drawing.Size(55, 14);
this.label4.TabIndex = 23;
this.label4.Text = "Address 2";
//
// label5
//
this.label5.Anchor = System.Windows.Forms.AnchorStyles.Right;
this.label5.AutoSize = true;
this.label5.Location = new System.Drawing.Point(38, 91);
this.label5.Name = "label5";
this.label5.Size = new System.Drawing.Size(24, 14);
this.label5.TabIndex = 24;
this.label5.Text = "City";
//
// label6
//
this.label6.Anchor = System.Windows.Forms.AnchorStyles.Right;
```

```
this.label6.AutoSize = true;
this.label6.Location = new System.Drawing.Point(351, 91);
this.label6.Name = "label6";
this.label6.Size = new System.Drawing.Size(31, 14);
this.label6.TabIndex = 25;
this.label6.Text = "State";
// 
// label9
//
this.label9.Anchor = System.Windows.Forms.AnchorStyles.Right;
this.label9.AutoSize = true;
this.label9.Location = new System.Drawing.Point(326, 119);
this.label9.Name = "label9";
this.label9.Size = new System.Drawing.Size(56, 14);
this.label9.TabIndex = 33;
this.label9.Text = "Phone (H)";
// 
// textBox2
//
this.textBox2.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Left | System.Windows.Forms.AnchorStyles.Right)));
this.tableLayoutPanel1.SetColumnSpan(this.textBox2, 3);
this.textBox2.Location = new System.Drawing.Point(68, 32);
this.textBox2.Name = "textBox2";
this.textBox2.Size = new System.Drawing.Size(552, 20);
this.textBox2.TabIndex = 2;
// 
// textBox3
//
this.textBox3.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Left | System.Windows.Forms.AnchorStyles.Right)));
this.tableLayoutPanel1.SetColumnSpan(this.textBox3, 3);
this.textBox3.Location = new System.Drawing.Point(68, 60);
this.textBox3.Name = "textBox3";
this.textBox3.Size = new System.Drawing.Size(552, 20);
this.textBox3.TabIndex = 3;
// 
// textBox4
//
this.textBox4.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Left | System.Windows.Forms.AnchorStyles.Right)));
this.textBox4.Location = new System.Drawing.Point(68, 88);
this.textBox4.Name = "textBox4";
this.textBox4.Size = new System.Drawing.Size(249, 20);
this.textBox4.TabIndex = 4;
// 
// textBox5
//
this.textBox5.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Left | System.Windows.Forms.AnchorStyles.Right)));
this.textBox5.Location = new System.Drawing.Point(388, 4);
this.textBox5.Name = "textBox5";
this.textBox5.Size = new System.Drawing.Size(232, 20);
this.textBox5.TabIndex = 1;
// 
// maskedTextBox1
//
this.maskedTextBox1.Anchor = System.Windows.Forms.AnchorStyles.Left;
this.maskedTextBox1.Location = new System.Drawing.Point(68, 116);
this.maskedTextBox1.Mask = "(999)000-0000";
this.maskedTextBox1.Name = "maskedTextBox1";
this.maskedTextBox1.TabIndex = 6;
// 
// maskedTextBox2
//
this.maskedTextBox2.Anchor = System.Windows.Forms.AnchorStyles.Left;
this.maskedTextBox2.Location = new System.Drawing.Point(388, 116);
this.maskedTextBox2.Mask = "(999)000-0000";
this.maskedTextBox2.Name = "maskedTextBox2";
```

```
    this.maskedTextBox2.TabIndex = 7;
//
// comboBox1
//
    this.comboBox1.Anchor = System.Windows.Forms.AnchorStyles.Left;
    this.comboBox1.FormattingEnabled = true;
    this.comboBox1.Items.AddRange(new object[] {
        "AK - Alaska",
        "WA - Washington"});
    this.comboBox1.Location = new System.Drawing.Point(388, 87);
    this.comboBox1.Name = "comboBox1";
    this.comboBox1.Size = new System.Drawing.Size(100, 21);
    this.comboBox1.TabIndex = 5;
//
// textBox1
//
    this.textBox1.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Left | System.Windows.Forms.AnchorStyles.Right)));
    this.textBox1.Location = new System.Drawing.Point(68, 4);
    this.textBox1.Name = "textBox1";
    this.textBox1.Size = new System.Drawing.Size(249, 20);
    this.textBox1.TabIndex = 0;
//
// label7
//
    this.label7.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Right)));
    this.label7.AutoSize = true;
    this.label7.Location = new System.Drawing.Point(28, 143);
    this.label7.Name = "label7";
    this.label7.Size = new System.Drawing.Size(34, 14);
    this.label7.TabIndex = 26;
    this.label7.Text = "Notes";
//
// label8
//
    this.label8.Anchor = System.Windows.Forms.AnchorStyles.Right;
    this.label8.AutoSize = true;
    this.label8.Location = new System.Drawing.Point(4, 119);
    this.label8.Name = "label8";
    this.label8.Size = new System.Drawing.Size(58, 14);
    this.label8.TabIndex = 32;
    this.label8.Text = "Phone (W)";
//
// richTextBox1
//
    this.tableLayoutPanel1.SetColumnSpan(this.richTextBox1, 3);
    this.richTextBox1.Dock = System.Windows.Forms.DockStyle.Fill;
    this.richTextBox1.Location = new System.Drawing.Point(68, 143);
    this.richTextBox1.Name = "richTextBox1";
    this.richTextBox1.Size = new System.Drawing.Size(552, 140);
    this.richTextBox1.TabIndex = 8;
    this.richTextBox1.Text = "";
//
// cancelBtn
//
    this.cancelBtn.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Bottom | System.Windows.Forms.AnchorStyles.Right)));
    this.cancelBtn.DialogResult = System.Windows.Forms.DialogResult.Cancel;
    this.cancelBtn.Location = new System.Drawing.Point(558, 306);
    this.cancelBtn.Name = "cancelBtn";
    this.cancelBtn.TabIndex = 1;
    this.cancelBtn.Text = "Cancel";
    this.cancelBtn.Click += new System.EventHandler(this.cancelBtn_Click);
//
// okBtn
//
    this.okBtn.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Bottom | System.Windows.Forms.AnchorStyles.Right)));

```

```

        this.okBtn.DialogResult = System.Windows.Forms.DialogResult.OK;
        this.okBtn.Location = new System.Drawing.Point(476, 306);
        this.okBtn.Name = "okBtn";
        this.okBtn.TabIndex = 0;
        this.okBtn.Text = "OK";
        this.okBtn.Click += new System.EventHandler(this.okBtn_Click);

        // BasicDataEntryForm
        //

        this.ClientSize = new System.Drawing.Size(642, 338);
        this.Controls.Add(this.okBtn);
        this.Controls.Add(this.cancelBtn);
        this.Controls.Add(this.tableLayoutPanel1);
        this.Name = "BasicDataEntryForm";
        this.Padding = new System.Windows.Forms.Padding(9);
        this.StartPosition = System.Windows.Forms.FormStartPosition.Manual;
        this.Text = "Basic Data Entry";
        this.tableLayoutPanel1.ResumeLayout(false);
        this.tableLayoutPanel1.PerformLayout();
        this.ResumeLayout(false);

    }

    private System.Windows.Forms.TableLayoutPanel tableLayoutPanel1;
    private System.Windows.Forms.Label label1;
    private System.Windows.Forms.Label label2;
    private System.Windows.Forms.Label label3;
    private System.Windows.Forms.Label label4;
    private System.Windows.Forms.Label label5;
    private System.Windows.Forms.Label label6;
    private System.Windows.Forms.Label label7;
    private System.Windows.Forms.Label label8;
    private System.Windows.Forms.Label label9;
    private System.Windows.Forms.Button cancelBtn;
    private System.Windows.Forms.Button okBtn;
    private System.Windows.Forms.TextBox textBox1;
    private System.Windows.Forms.TextBox textBox2;
    private System.Windows.Forms.TextBox textBox3;
    private System.Windows.Forms.TextBox textBox4;
    private System.Windows.Forms.TextBox textBox5;
    private System.Windows.Forms.MaskedTextBox maskedTextBox1;
    private System.Windows.Forms.MaskedTextBox maskedTextBox2;
    private System.Windows.Forms.ComboBox comboBox1;
    private System.Windows.Forms.RichTextBox richTextBox1;

    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.Run(new BasicDataEntryForm());
    }
}

```

```

Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Text
Imports System.Windows.Forms

' This form demonstrates how to build a form layout that adjusts well
' when the user resizes the form. It also demonstrates a layout that
' responds well to localization.
Class BasicDataEntryForm
    Inherits System.Windows.Forms.Form

```

```

Public Sub New()
    InitializeComponent()
End Sub

Private components As System.ComponentModel.IContainer = Nothing

Protected Overrides Sub Dispose(disposing As Boolean)
    If disposing AndAlso (components IsNot Nothing) Then
        components.Dispose()
    End If
    MyBase.Dispose(disposing)
End Sub

Public Overrides Function ToString() As String
    Return "Basic Data Entry Form"
End Function

Private Sub okBtn_Click(ByVal sender As Object, ByVal e As EventArgs) Handles okBtn.Click
    Me.Close()
End Sub

Private Sub cancelBtn_Click(ByVal sender As Object, ByVal e As EventArgs) Handles cancelBtn.Click
    Me.Close()
End Sub

Private Sub InitializeComponent()
    Me.tableLayoutPanel1 = New System.Windows.Forms.TableLayoutPanel
    Me.label1 = New System.Windows.Forms.Label
    Me.label2 = New System.Windows.Forms.Label
    Me.label3 = New System.Windows.Forms.Label
    Me.label4 = New System.Windows.Forms.Label
    Me.label5 = New System.Windows.Forms.Label
    Me.label6 = New System.Windows.Forms.Label
    Me.label9 = New System.Windows.Forms.Label
    Me.textBox2 = New System.Windows.Forms.TextBox
    Me.textBox3 = New System.Windows.Forms.TextBox
    Me.textBox4 = New System.Windows.Forms.TextBox
    Me.textBox5 = New System.Windows.Forms.TextBox
    Me.maskedTextBox1 = New System.Windows.Forms.MaskedTextBox
    Me.maskedTextBox2 = New System.Windows.Forms.MaskedTextBox
    Me.comboBox1 = New System.Windows.Forms.ComboBox
    Me.textBox1 = New System.Windows.Forms.TextBox
    Me.label7 = New System.Windows.Forms.Label
    Me.label8 = New System.Windows.Forms.Label
    Me.richTextBox1 = New System.Windows.Forms.RichTextBox
    Me.cancelBtn = New System.Windows.Forms.Button
    Me.okBtn = New System.Windows.Forms.Button
    Me.tableLayoutPanel1.SuspendLayout()
    Me.SuspendLayout()

    'tableLayoutPanel1
    'Me.tableLayoutPanel1.Anchor = CType((((System.Windows.Forms.AnchorStyles.Top Or
    System.Windows.Forms.AnchorStyles.Bottom) _
        Or System.Windows.Forms.AnchorStyles.Left) _
        Or System.Windows.Forms.AnchorStyles.Right), System.Windows.Forms.AnchorStyles)
    Me.tableLayoutPanel1.ColumnCount = 4
    Me.tableLayoutPanel1.ColumnStyles.Add(New System.Windows.Forms.ColumnStyle)
    Me.tableLayoutPanel1.ColumnStyles.Add(New
    System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 50.0!))
    Me.tableLayoutPanel1.ColumnStyles.Add(New
    System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 50.0!))
    Me.tableLayoutPanel1.ColumnStyles.Add(New
    System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 50.0!))

    Me.tableLayoutPanel1.Controls.Add(Me.label1, 0, 0)
    Me.tableLayoutPanel1.Controls.Add(Me.label2, 2, 0)
    Me.tableLayoutPanel1.Controls.Add(Me.label3, 0, 1)
    Me.tableLayoutPanel1.Controls.Add(Me.label4, 0, 2)

```

```
Me.tableLayoutPanel1.Controls.Add(Me.label5, 0, 3)
Me.tableLayoutPanel1.Controls.Add(Me.label6, 2, 3)
Me.tableLayoutPanel1.Controls.Add(Me.label9, 2, 4)
Me.tableLayoutPanel1.Controls.Add(Me.textBox2, 1, 1)
Me.tableLayoutPanel1.Controls.Add(Me.textBox3, 1, 2)
Me.tableLayoutPanel1.Controls.Add(Me.textBox4, 1, 3)
Me.tableLayoutPanel1.Controls.Add(Me.textBox5, 3, 0)
Me.tableLayoutPanel1.Controls.Add(Me.maskedTextBox1, 1, 4)
Me.tableLayoutPanel1.Controls.Add(Me.maskedTextBox2, 3, 4)
Me.tableLayoutPanel1.Controls.Add(Me.comboBox1, 3, 3)
Me.tableLayoutPanel1.Controls.Add(Me.textBox1, 1, 0)
Me.tableLayoutPanel1.Controls.Add(Me.label7, 0, 5)
Me.tableLayoutPanel1.Controls.Add(Me.label8, 0, 4)
Me.tableLayoutPanel1.Controls.Add(Me.richTextBox1, 1, 5)
Me.tableLayoutPanel1.Location = New System.Drawing.Point(13, 13)
Me.tableLayoutPanel1.Name = "tableLayoutPanel1"
Me.tableLayoutPanel1.RowCount = 6
Me.tableLayoutPanel1.RowStyles.Add(New
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Absolute, 28.0!))
Me.tableLayoutPanel1.RowStyles.Add(New
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 80.0!))
Me.tableLayoutPanel1.RowStyles.Add(New
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Absolute, 20.0!))
Me.tableLayoutPanel1.Size = New System.Drawing.Size(623, 286)
Me.tableLayoutPanel1.TabIndex = 0
'
'label1
'
Me.label1.Anchor = System.Windows.Forms.AnchorStyles.Right
Me.label1.AutoSize = True
Me.label1.Location = New System.Drawing.Point(3, 7)
Me.label1.Name = "label1"
Me.label1.Size = New System.Drawing.Size(59, 14)
Me.label1.TabIndex = 20
Me.label1.Text = "First Name"
'
'label2
'
Me.label2.Anchor = System.Windows.Forms.AnchorStyles.Right
Me.label2.AutoSize = True
Me.label2.Location = New System.Drawing.Point(323, 7)
Me.label2.Name = "label2"
Me.label2.Size = New System.Drawing.Size(59, 14)
Me.label2.TabIndex = 21
Me.label2.Text = "Last Name"
'
'label3
'
Me.label3.Anchor = System.Windows.Forms.AnchorStyles.Right
Me.label3.AutoSize = True
Me.label3.Location = New System.Drawing.Point(10, 35)
Me.label3.Name = "label3"
Me.label3.Size = New System.Drawing.Size(52, 14)
Me.label3.TabIndex = 22
Me.label3.Text = "Address1"
'
'label4
'
Me.label4.Anchor = System.Windows.Forms.AnchorStyles.Right
Me.label4.AutoSize = True
Me.label4.Location = New System.Drawing.Point(7, 63)
```

```
Me.label4.Name = "label4"
Me.label4.Size = New System.Drawing.Size(55, 14)
Me.label4.TabIndex = 23
Me.label4.Text = "Address 2"
'
'label5
'
Me.label5.Anchor = System.Windows.Forms.AnchorStyles.Right
Me.label5.AutoSize = True
Me.label5.Location = New System.Drawing.Point(38, 91)
Me.label5.Name = "label5"
Me.label5.Size = New System.Drawing.Size(24, 14)
Me.label5.TabIndex = 24
Me.label5.Text = "City"
'
'label6
'
Me.label6.Anchor = System.Windows.Forms.AnchorStyles.Right
Me.label6.AutoSize = True
Me.label6.Location = New System.Drawing.Point(351, 91)
Me.label6.Name = "label6"
Me.label6.Size = New System.Drawing.Size(31, 14)
Me.label6.TabIndex = 25
Me.label6.Text = "State"
'
'label9
'
Me.label9.Anchor = System.Windows.Forms.AnchorStyles.Right
Me.label9.AutoSize = True
Me.label9.Location = New System.Drawing.Point(326, 119)
Me.label9.Name = "label9"
Me.label9.Size = New System.Drawing.Size(56, 14)
Me.label9.TabIndex = 33
Me.label9.Text = "Phone (H)"
'
'textBox2
'
Me.textBox2.Anchor = CType((System.Windows.Forms.AnchorStyles.Left Or
System.Windows.Forms.AnchorStyles.Right), System.Windows.Forms.AnchorStyles)
Me.tableLayoutPanel1.SetColumnSpan(Me.textBox2, 3)
Me.textBox2.Location = New System.Drawing.Point(68, 32)
Me.textBox2.Name = "textBox2"
Me.textBox2.Size = New System.Drawing.Size(552, 20)
Me.textBox2.TabIndex = 2
'
'textBox3
'
Me.textBox3.Anchor = CType((System.Windows.Forms.AnchorStyles.Left Or
System.Windows.Forms.AnchorStyles.Right), System.Windows.Forms.AnchorStyles)
Me.tableLayoutPanel1.SetColumnSpan(Me.textBox3, 3)
Me.textBox3.Location = New System.Drawing.Point(68, 60)
Me.textBox3.Name = "textBox3"
Me.textBox3.Size = New System.Drawing.Size(552, 20)
Me.textBox3.TabIndex = 3
'
'textBox4
'
Me.textBox4.Anchor = CType((System.Windows.Forms.AnchorStyles.Left Or
System.Windows.Forms.AnchorStyles.Right), System.Windows.Forms.AnchorStyles)
Me.textBox4.Location = New System.Drawing.Point(68, 88)
Me.textBox4.Name = "textBox4"
Me.textBox4.Size = New System.Drawing.Size(249, 20)
Me.textBox4.TabIndex = 4
'
'textBox5
'
Me.textBox5.Anchor = CType((System.Windows.Forms.AnchorStyles.Left Or
System.Windows.Forms.AnchorStyles.Right), System.Windows.Forms.AnchorStyles)
Me.textBox5.Location = New System.Drawing.Point(388, 11)
```

```
Me.textBox5.Location = New System.Drawing.Point(500, 4)
Me.textBox5.Name = "textBox5"
Me.textBox5.Size = New System.Drawing.Size(232, 20)
Me.textBox5.TabIndex = 1
'
'maskedTextBox1
'
Me.maskedTextBox1.Anchor = System.Windows.Forms.AnchorStyles.Left
Me.maskedTextBox1.Location = New System.Drawing.Point(68, 116)
Me.maskedTextBox1.Mask = "(999)000-0000"
Me.maskedTextBox1.Name = "maskedTextBox1"
Me.maskedTextBox1.TabIndex = 6
'
'maskedTextBox2
'
Me.maskedTextBox2.Anchor = System.Windows.Forms.AnchorStyles.Left
Me.maskedTextBox2.Location = New System.Drawing.Point(388, 116)
Me.maskedTextBox2.Mask = "(999)000-0000"
Me.maskedTextBox2.Name = "maskedTextBox2"
Me.maskedTextBox2.TabIndex = 7
'
'comboBox1
'
Me.comboBox1.Anchor = System.Windows.Forms.AnchorStyles.Left
Me.comboBox1.FormattingEnabled = True
Me.comboBox1.Items.AddRange(New Object() {"AK - Alaska", "WA - Washington"})
Me.comboBox1.Location = New System.Drawing.Point(388, 87)
Me.comboBox1.Name = "comboBox1"
Me.comboBox1.Size = New System.Drawing.Size(100, 21)
Me.comboBox1.TabIndex = 5
'
'textBox1
'
Me.textBox1.Anchor = CType((System.Windows.Forms.AnchorStyles.Left Or
System.Windows.Forms.AnchorStyles.Right), System.Windows.Forms.AnchorStyles)
Me.textBox1.Location = New System.Drawing.Point(68, 4)
Me.textBox1.Name = "textBox1"
Me.textBox1.Size = New System.Drawing.Size(249, 20)
Me.textBox1.TabIndex = 0
'
'label7
'
Me.label7.Anchor = CType((System.Windows.Forms.AnchorStyles.Top Or
System.Windows.Forms.AnchorStyles.Right), System.Windows.Forms.AnchorStyles)
Me.label7.AutoSize = True
Me.label7.Location = New System.Drawing.Point(28, 143)
Me.label7.Name = "label7"
Me.label7.Size = New System.Drawing.Size(34, 14)
Me.label7.TabIndex = 26
Me.label7.Text = "Notes"
'
'label8
'
Me.label8.Anchor = System.Windows.Forms.AnchorStyles.Right
Me.label8.AutoSize = True
Me.label8.Location = New System.Drawing.Point(4, 119)
Me.label8.Name = "label8"
Me.label8.Size = New System.Drawing.Size(58, 14)
Me.label8.TabIndex = 32
Me.label8.Text = "Phone (W)"
'
'richTextBox1
'
Me.tableLayoutPanel1.SetColumnSpan(Me.richTextBox1, 3)
Me.richTextBox1.Dock = System.Windows.Forms.DockStyle.Fill
Me.richTextBox1.Location = New System.Drawing.Point(68, 143)
Me.richTextBox1.Name = "richTextBox1"
Me.richTextBox1.Size = New System.Drawing.Size(552, 140)
Me.richTextBox1.TabIndex = 8
Me.richTextBox1.Text = ""
```

```

Me.richTextBox1.Text = ""

'cancelBtn
'

Me.cancelBtn.Anchor = CType((System.Windows.Forms.AnchorStyles.Bottom Or
System.Windows.Forms.AnchorStyles.Right), System.Windows.Forms.AnchorStyles)
Me.cancelBtn.DialogResult = System.Windows.Forms.DialogResult.Cancel
Me.cancelBtn.Location = New System.Drawing.Point(558, 306)
Me.cancelBtn.Name = "cancelBtn"
Me.cancelBtn.TabIndex = 1
Me.cancelBtn.Text = "Cancel"
'

'okBtn
'

Me.okBtn.Anchor = CType((System.Windows.Forms.AnchorStyles.Bottom Or
System.Windows.Forms.AnchorStyles.Right), System.Windows.Forms.AnchorStyles)
Me.okBtn.DialogResult = System.Windows.Forms.DialogResult.OK
Me.okBtn.Location = New System.Drawing.Point(476, 306)
Me.okBtn.Name = "okBtn"
Me.okBtn.TabIndex = 0
Me.okBtn.Text = "OK"
'

'BasicDataEntryForm
'

Me.ClientSize = New System.Drawing.Size(642, 338)
Me.Controls.Add(Me.okBtn)
Me.Controls.Add(Me.cancelBtn)
Me.Controls.Add(Me.tableLayoutPanel1)
Me.Name = "BasicDataEntryForm"
Me.Padding = New System.Windows.Forms.Padding(9)
Me.StartPosition = System.Windows.Forms.FormStartPosition.Manual
Me.Text = "Basic Data Entry"
Me.tableLayoutPanel1.ResumeLayout(False)
Me.tableLayoutPanel1.PerformLayout()
Me.ResumeLayout(False)

End Sub

Friend WithEvents tableLayoutPanel1 As System.Windows.Forms.TableLayoutPanel
Friend WithEvents label1 As System.Windows.Forms.Label
Friend WithEvents label2 As System.Windows.Forms.Label
Friend WithEvents label3 As System.Windows.Forms.Label
Friend WithEvents label4 As System.Windows.Forms.Label
Friend WithEvents label5 As System.Windows.Forms.Label
Friend WithEvents label6 As System.Windows.Forms.Label
Friend WithEvents label7 As System.Windows.Forms.Label
Friend WithEvents label8 As System.Windows.Forms.Label
Friend WithEvents label9 As System.Windows.Forms.Label
Friend WithEvents cancelBtn As System.Windows.Forms.Button
Friend WithEvents okBtn As System.Windows.Forms.Button
Friend WithEvents textBox1 As System.Windows.Forms.TextBox
Friend WithEvents textBox2 As System.Windows.Forms.TextBox
Friend WithEvents textBox3 As System.Windows.Forms.TextBox
Friend WithEvents textBox4 As System.Windows.Forms.TextBox
Friend WithEvents textBox5 As System.Windows.Forms.TextBox
Friend WithEvents maskedTextBox1 As System.Windows.Forms.MaskedTextBox
Friend WithEvents maskedTextBox2 As System.Windows.Forms.MaskedTextBox
Friend WithEvents comboBox1 As System.Windows.Forms.ComboBox
Friend WithEvents richTextBox1 As System.Windows.Forms.RichTextBox

<STAThread()> _
Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New BasicDataEntryForm())
End Sub
End Class

```

Compiling the Code

This example requires:

- References to the System, System.Data, System.Drawing and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[FlowLayoutPanel](#)

[TableLayoutPanel](#)

[How to: Anchor and Dock Child Controls in a TableLayoutPanel Control](#)

[How to: Design a Windows Forms Layout that Responds Well to Localization](#)

[Microsoft Windows User Experience, Official Guidelines for User Interface Developers and Designers](#). Redmond, WA: Microsoft Press, 1999. (ISBN: 0-7356-0566-1)

How to: Design a Windows Forms Layout that Responds Well to Localization

5/4/2018 • 8 min to read • [Edit Online](#)

Creating forms that are ready to be localized greatly speeds development for international markets. You can use the [TableLayoutPanel](#) control to implement layouts that respond gracefully as controls resize due to changes in their [Text](#) property values.

Example

This form demonstrates how to create a layout that proportionally adjusts when you translate displayed string values into other languages. This process of translation is called *localization*. For more information, see [Localization](#).

There is extensive support for this task in Visual Studio. See also [Walkthrough: Creating a Layout That Adjusts Proportion for Localization](#).

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace TableLayoutPanelSample
{
    public class LocalizableForm : Form
    {
        private System.Windows.Forms.TableLayoutPanel tableLayoutPanel1;
        private System.Windows.Forms.ListView listView1;
        private System.Windows.Forms.Panel panel1;
        private System.Windows.Forms.Button button3;
        private System.Windows.Forms.Button button2;
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.TableLayoutPanel tableLayoutPanel2;
        private System.Windows.Forms.Button button4;
        private System.Windows.Forms.Button button5;
        private System.Windows.Forms.Button button6;
        private System.Windows.Forms.Label label1;

        private System.ComponentModel.IContainer components = null;

        public LocalizableForm()
        {
            InitializeComponent();
        }

        private void AddText(object sender, EventArgs e)
        {
            ((Button)sender).Text += "x";
        }

        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
        }
    }
}
```

```

        }

        base.Dispose(disposing);
    }

    private void InitializeComponent()
    {
        System.Windows.Forms.ListViewGroup listViewGroup16 = new System.Windows.Forms.ListViewGroup("First Group",
System.Windows.Forms.HorizontalAlignment.Left);
        System.Windows.Forms.ListViewGroup listViewGroup17 = new System.Windows.Forms.ListViewGroup("Second Group",
System.Windows.Forms.HorizontalAlignment.Left);
        System.Windows.Forms.ListViewGroup listViewGroup18 = new System.Windows.Forms.ListViewGroup("Third Group",
System.Windows.Forms.HorizontalAlignment.Left);
        System.Windows.Forms.ListViewItem listViewItem26 = new System.Windows.Forms.ListViewItem("Item 1");
        System.Windows.Forms.ListViewItem listViewItem27 = new System.Windows.Forms.ListViewItem("Item 2");
        System.Windows.Forms.ListViewItem listViewItem28 = new System.Windows.Forms.ListViewItem("Item 3");
        System.Windows.Forms.ListViewItem listViewItem29 = new System.Windows.Forms.ListViewItem("Item 4");
        System.Windows.Forms.ListViewItem listViewItem30 = new System.Windows.Forms.ListViewItem("Item 5");
        this.tableLayoutPanel1 = new System.Windows.Forms.TableLayoutPanel();
        this.listView1 = new System.Windows.Forms.ListView();
        this.panel1 = new System.Windows.Forms.Panel();
        this.button3 = new System.Windows.Forms.Button();
        this.button2 = new System.Windows.Forms.Button();
        this.button1 = new System.Windows.Forms.Button();
        this.tableLayoutPanel2 = new System.Windows.Forms.TableLayoutPanel();
        this.button4 = new System.Windows.Forms.Button();
        this.button5 = new System.Windows.Forms.Button();
        this.button6 = new System.Windows.Forms.Button();
        this.label1 = new System.Windows.Forms.Label();
        this.tableLayoutPanel1.SuspendLayout();
        this.panel1.SuspendLayout();
        this.tableLayoutPanel2.SuspendLayout();
        this.SuspendLayout();

        //
        // tableLayoutPanel1
        //
        this.tableLayoutPanel1.ColumnCount = 2;
        this.tableLayoutPanel1.ColumnStyles.Add(new System.Windows.Forms.ColumnStyle());
        this.tableLayoutPanel1.ColumnStyles.Add(new
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 100F));
        this.tableLayoutPanel1.Controls.Add(this.listView1, 1, 0);
        this.tableLayoutPanel1.Controls.Add(this.panel1, 0, 0);
        this.tableLayoutPanel1.Location = new System.Drawing.Point(2, 52);
        this.tableLayoutPanel1.Name = "tableLayoutPanel1";
        this.tableLayoutPanel1.RowCount = 1;
        this.tableLayoutPanel1.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 50F));
        this.tableLayoutPanel1.Size = new System.Drawing.Size(524, 270);
        this.tableLayoutPanel1.TabIndex = 1;
        //
        // listView1
        //
        this.listView1.Dock = System.Windows.Forms.DockStyle.Fill;
        listViewGroup16.Header = "First Group";
        listViewGroup16.Name = null;
        listViewGroup17.Header = "Second Group";
        listViewGroup17.Name = null;
        listViewGroup18.Header = "Third Group";
        listViewGroup18.Name = null;
        this.listView1.Groups.AddRange(new System.Windows.Forms.ListViewGroup[] {
            listViewGroup16,
            listViewGroup17,
            listViewGroup18});
        //this.listView1.IsBackgroundImageTiled = false;
        listViewItem26.Group = listViewGroup16;
        listViewItem27.Group = listViewGroup16;
        listViewItem28.Group = listViewGroup17;
        listViewItem29.Group = listViewGroup17;
        listViewItem30.Group = listViewGroup18;
        this.listView1.Items.AddRange(new System.Windows.Forms.ListViewItem[] {

```

```
listViewItem26,
listViewItem27,
listViewItem28,
listViewItem29,
listViewItem30});

this.listView1.Location = new System.Drawing.Point(90, 3);
this.listView1.Name = "listView1";
this.listView1.Size = new System.Drawing.Size(431, 264);
this.listView1.TabIndex = 1;
//
// panel1
//
this.panel1.Anchor = System.Windows.Forms.AnchorStyles.Top;
this.panel1.AutoSize = true;
this.panel1.AutoSizeMode = System.Windows.Forms.AutoSizeMode.GrowAndShrink;
this.panel1.Controls.Add(this.button3);
this.panel1.Controls.Add(this.button2);
this.panel1.Controls.Add(this.button1);
this.panel1.Location = new System.Drawing.Point(3, 3);
this.panel1.Name = "panel1";
this.panel1.Size = new System.Drawing.Size(81, 86);
this.panel1.TabIndex = 2;
//
// button3
//
this.button3.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Left | System.Windows.Forms.AnchorStyles.Right)));
this.button3.AutoSize = true;
this.button3.Location = new System.Drawing.Point(3, 60);
this.button3.Name = "button3";
this.button3.TabIndex = 2;
this.button3.Text = "Add Text";
this.button3.Click += new System.EventHandler(this.AddText);
//
// button2
//
this.button2.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Left | System.Windows.Forms.AnchorStyles.Right)));
this.button2.AutoSize = true;
this.button2.Location = new System.Drawing.Point(3, 31);
this.button2.Name = "button2";
this.button2.TabIndex = 1;
this.button2.Text = "Add Text";
this.button2.Click += new System.EventHandler(this.AddText);
//
// button1
//
this.button1.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Left | System.Windows.Forms.AnchorStyles.Right)));
this.button1.AutoSize = true;
this.button1.Location = new System.Drawing.Point(3, 2);
this.button1.Name = "button1";
this.button1.TabIndex = 0;
this.button1.Text = "Add Text";
this.button1.Click += new System.EventHandler(this.AddText);
//
// tableLayoutPanel2
//
this.tableLayoutPanel2.Anchor = ((System.Windows.Forms.AnchorStyles)
((System.Windows.Forms.AnchorStyles.Bottom | System.Windows.Forms.AnchorStyles.Right)));
this.tableLayoutPanel2.AutoSize = true;
this.tableLayoutPanel2.ColumnCount = 3;
this.tableLayoutPanel2.ColumnStyles.Add(new
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 33.3333F));
this.tableLayoutPanel2.ColumnStyles.Add(new
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 33.3333F));
this.tableLayoutPanel2.ColumnStyles.Add(new
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 33.3333F));
this.tableLayoutPanel2.Controls.Add(this.button4, 0, 0);
```

```
this.tableLayoutPanel2.Controls.Add(this.button5, 1, 0);
this.tableLayoutPanel2.Controls.Add(this.button6, 2, 0);
this.tableLayoutPanel2.Location = new System.Drawing.Point(284, 328);
this.tableLayoutPanel2.Name = "tableLayoutPanel2";
this.tableLayoutPanel2.RowCount = 1;
this.tableLayoutPanel2.RowStyles.Add(new System.Windows.Forms.RowStyle());
this.tableLayoutPanel2.Size = new System.Drawing.Size(243, 34);
this.tableLayoutPanel2.TabIndex = 2;
//
// button4
//
this.button4.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Left | System.Windows.Forms.AnchorStyles.Right)));
this.button4.AutoSize = true;
this.button4.Location = new System.Drawing.Point(3, 5);
this.button4.Name = "button4";
this.button4.TabIndex = 0;
this.button4.Text = "Add Text";
this.button4.Click += new System.EventHandler(this.AddText);
//
// button5
//
this.button5.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Left | System.Windows.Forms.AnchorStyles.Right)));
this.button5.AutoSize = true;
this.button5.Location = new System.Drawing.Point(84, 5);
this.button5.Name = "button5";
this.button5.TabIndex = 1;
this.button5.Text = "Add Text";
this.button5.Click += new System.EventHandler(this.AddText);
//
// button6
//
this.button6.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Left | System.Windows.Forms.AnchorStyles.Right)));
this.button6.AutoSize = true;
this.button6.Location = new System.Drawing.Point(165, 5);
this.button6.Name = "button6";
this.button6.TabIndex = 2;
this.button6.Text = "Add Text";
this.button6.Click += new System.EventHandler(this.AddText);
//
// label1
//
this.label1.Location = new System.Drawing.Point(8, 7);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(518, 40);
this.label1.TabIndex = 3;
this.label1.Text = "Click on any button to add text to the button. This simulates localizing strings," +
" and provides a good demonstration of how the dialog will automatically adjust w" +
"hen those longer strings are added to the UI.";
//
// LocalizableForm
//
this.ClientSize = new System.Drawing.Size(539, 374);
this.Controls.Add(this.label1);
this.Controls.Add(this.tableLayoutPanel2);
this.Controls.Add(this.tableLayoutPanel1);
this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle;
this.Name = "LocalizableForm";
this.Text = "Localizable Dialog";
this.tableLayoutPanel1.ResumeLayout(false);
this.tableLayoutPanel1.PerformLayout();
this.panel1.ResumeLayout(false);
this.panel1.PerformLayout();
this.tableLayoutPanel2.ResumeLayout(false);
this.tableLayoutPanel2.PerformLayout();
this.ResumeLayout(false);
this.PerformLayout();
```

```
}

[STAThread]
static void Main()
{
    Application.EnableVisualStyles();
    Application.Run(new LocalizableForm());
}

}
```

```

Me.button3 = New System.Windows.Forms.Button()
Me.button2 = New System.Windows.Forms.Button()
Me.button1 = New System.Windows.Forms.Button()
Me.tableLayoutPanel2 = New System.Windows.Forms.TableLayoutPanel()
Me.button4 = New System.Windows.Forms.Button()
Me.button5 = New System.Windows.Forms.Button()
Me.button6 = New System.Windows.Forms.Button()
Me.label11 = New System.Windows.Forms.Label()
Me.tableLayoutPanel1.SuspendLayout()
Me.panel11.SuspendLayout()
Me.tableLayoutPanel2.SuspendLayout()
Me.SuspendLayout()
'
' tableLayoutPanel1
'
Me.tableLayoutPanel1.ColumnCount = 2
Me.tableLayoutPanel1.ColumnStyles.Add(New System.Windows.Forms.ColumnStyle())
Me.tableLayoutPanel1.ColumnStyles.Add(New
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 100F))
Me.tableLayoutPanel1.Controls.Add(Me.listView1, 1, 0)
Me.tableLayoutPanel1.Controls.Add(Me.panel11, 0, 0)
Me.tableLayoutPanel1.Location = New System.Drawing.Point(2, 52)
Me.tableLayoutPanel1.Name = "tableLayoutPanel1"
Me.tableLayoutPanel1.RowCount = 1
Me.tableLayoutPanel1.RowStyles.Add(New
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 50F))
Me.tableLayoutPanel1.Size = New System.Drawing.Size(524, 270)
Me.tableLayoutPanel1.TabIndex = 1
'
' listView1
'
Me.listView1.Dock = System.Windows.Forms.DockStyle.Fill
listViewGroup16.Header = "First Group"
listViewGroup16.Name = Nothing
listViewGroup17.Header = "Second Group"
listViewGroup17.Name = Nothing
listViewGroup18.Header = "Third Group"
listViewGroup18.Name = Nothing
Me.listView1.Groups.AddRange(New System.Windows.Forms.ListViewGroup() {listViewGroup16, listViewGroup17,
listViewGroup18})
'this.listView1.IsBackgroundImageTiled = false;
listViewItem26.Group = listViewGroup16
listViewItem27.Group = listViewGroup16
listViewItem28.Group = listViewGroup17
listViewItem29.Group = listViewGroup17
listViewItem30.Group = listViewGroup18
Me.listView1.Items.AddRange(New System.Windows.Forms.ListViewItem() {listViewItem26, listViewItem27,
listViewItem28, listViewItem29, listViewItem30})
Me.listView1.Location = New System.Drawing.Point(90, 3)
Me.listView1.Name = "listView1"
Me.listView1.Size = New System.Drawing.Size(431, 264)
Me.listView1.TabIndex = 1
'
' panel1
'
Me.panel11.Anchor = System.Windows.Forms.AnchorStyles.Top
Me.panel11.AutoSize = True
Me.panel11.AutoSizeMode = System.Windows.Forms.AutoSizeMode.GrowAndShrink
Me.panel11.Controls.Add(Me.button3)
Me.panel11.Controls.Add(Me.button2)
Me.panel11.Controls.Add(Me.button1)
Me.panel11.Location = New System.Drawing.Point(3, 3)
Me.panel11.Name = "panel1"
Me.panel11.Size = New System.Drawing.Size(81, 86)
Me.panel11.TabIndex = 2
'
' button3
'
Me.button3.Anchor = CType((System.Windows.Forms.AnchorStyles.Left Or

```

```
Me.button3.Anchor = CType(System.Windows.Forms.AnchorStyles.Right, System.Windows.Forms.AnchorStyles)
System.Windows.Forms.AnchorStyles.Right, System.Windows.Forms.AnchorStyles)
Me.button3.AutoSize = True
Me.button3.Location = New System.Drawing.Point(3, 60)
Me.button3.Name = "button3"
Me.button3.TabIndex = 2
Me.button3.Text = "Add Text"
'
' button2
'
Me.button2.Anchor = CType(System.Windows.Forms.AnchorStyles.Left Or
System.Windows.Forms.AnchorStyles.Right, System.Windows.Forms.AnchorStyles)
Me.button2.AutoSize = True
Me.button2.Location = New System.Drawing.Point(3, 31)
Me.button2.Name = "button2"
Me.button2.TabIndex = 1
Me.button2.Text = "Add Text"
'
' button1
'
Me.button1.Anchor = CType(System.Windows.Forms.AnchorStyles.Left Or
System.Windows.Forms.AnchorStyles.Right, System.Windows.Forms.AnchorStyles)
Me.button1.AutoSize = True
Me.button1.Location = New System.Drawing.Point(3, 2)
Me.button1.Name = "button1"
Me.button1.TabIndex = 0
Me.button1.Text = "Add Text"
'
' tableLayoutPanel2
'
Me.tableLayoutPanel2.Anchor = CType(System.Windows.Forms.AnchorStyles.Bottom Or
System.Windows.Forms.AnchorStyles.Right, System.Windows.Forms.AnchorStyles)
Me.tableLayoutPanel2.AutoSize = True
Me.tableLayoutPanel2.ColumnCount = 3
Me.tableLayoutPanel2.ColumnStyles.Add(New
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 33.3333F))
Me.tableLayoutPanel2.ColumnStyles.Add(New
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 33.3333F))
Me.tableLayoutPanel2.ColumnStyles.Add(New
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 33.3333F))
Me.tableLayoutPanel2.Controls.Add(Me.button4, 0, 0)
Me.tableLayoutPanel2.Controls.Add(Me.button5, 1, 0)
Me.tableLayoutPanel2.Controls.Add(Me.button6, 2, 0)
Me.tableLayoutPanel2.Location = New System.Drawing.Point(284, 328)
Me.tableLayoutPanel2.Name = "tableLayoutPanel2"
Me.tableLayoutPanel2.RowCount = 1
Me.tableLayoutPanel2.RowStyles.Add(New System.Windows.Forms.RowStyle())
Me.tableLayoutPanel2.Size = New System.Drawing.Size(243, 34)
Me.tableLayoutPanel2.TabIndex = 2
'
' button4
'
Me.button4.Anchor = CType(System.Windows.Forms.AnchorStyles.Left Or
System.Windows.Forms.AnchorStyles.Right, System.Windows.Forms.AnchorStyles)
Me.button4.AutoSize = True
Me.button4.Location = New System.Drawing.Point(3, 5)
Me.button4.Name = "button4"
Me.button4.TabIndex = 0
Me.button4.Text = "Add Text"
'
' button5
'
Me.button5.Anchor = CType(System.Windows.Forms.AnchorStyles.Left Or
System.Windows.Forms.AnchorStyles.Right, System.Windows.Forms.AnchorStyles)
Me.button5.AutoSize = True
Me.button5.Location = New System.Drawing.Point(84, 5)
Me.button5.Name = "button5"
Me.button5.TabIndex = 1
Me.button5.Text = "Add Text"
'
```

```

' button6
'
Me.button6.Anchor = CType(System.Windows.Forms.AnchorStyles.Left Or
System.Windows.Forms.AnchorStyles.Right, System.Windows.Forms.AnchorStyles)
Me.button6.AutoSize = True
Me.button6.Location = New System.Drawing.Point(165, 5)
Me.button6.Name = "button6"
Me.button6.TabIndex = 2
Me.button6.Text = "Add Text"
'
' label1
'
Me.label1.Location = New System.Drawing.Point(8, 7)
Me.label1.Name = "label1"
Me.label1.Size = New System.Drawing.Size(518, 40)
Me.label1.TabIndex = 3
Me.label1.Text = "Click on any button to add text to the button. This simulates localizing strings," + "
and provides a good demonstration of how the dialog will automatically adjust when those longer strings
are added to the UI."
'
' LocalizableForm
'
Me.ClientSize = New System.Drawing.Size(539, 374)
Me.Controls.Add(label1)
Me.Controls.Add(tableLayoutPanel2)
Me.Controls.Add(tableLayoutPanel1)
Me.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle
Me.Name = "LocalizableForm"
Me.Text = "Localizable Dialog"
Me.tableLayoutPanel1.ResumeLayout(False)
Me.tableLayoutPanel1.PerformLayout()
Me.panel1.ResumeLayout(False)
Me.panel1.PerformLayout()
Me.tableLayoutPanel2.ResumeLayout(False)
Me.tableLayoutPanel2.PerformLayout()
Me.ResumeLayout(False)
Me.PerformLayout()
End Sub

<STAThread()> _
Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New LocalizableForm())
End Sub
End Class

```

1. [How to: Align and Stretch a Control in a TableLayoutPanel Control](#)
2. [Walkthrough: Arranging Controls on Windows Forms Using a FlowLayoutPanel](#)
3. [How to: Span Rows and Columns in a TableLayoutPanel Control](#)
4. [How to: Edit Columns and Rows in a TableLayoutPanel Control](#)
5. [Walkthrough: Performing Common Tasks Using Smart Tags on Windows Forms Controls](#)
6. [Walkthrough: Arranging Controls on Windows Forms Using a TableLayoutPanel](#)
7. [Walkthrough: Laying Out Windows Forms Controls with Padding, Margins, and the AutoSize Property](#)
8. [How to: Support Localization on Windows Forms Using AutoSize and the TableLayoutPanel Control](#)
9. [Walkthrough: Creating a Resizable Windows Form for Data Entry](#)

Compiling the Code

This example requires:

- References to the System, System.Data, System.Drawing and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[TableLayoutPanel](#)

[FlowLayoutPanel](#)

[Localization](#)

How to: Edit Columns and Rows in a TableLayoutPanel Control

5/4/2018 • 1 min to read • [Edit Online](#)

You can use the collection editor of the [TableLayoutPanel](#) control, called the **Column and Row Styles** dialog box, to edit the rows and columns of your controls.

NOTE

If you want a control to span multiple rows or columns, set the `RowSpan` and `ColumnSpan` properties on the control. For more information, see [Walkthrough: Arranging Controls on Windows Forms Using a TableLayoutPanel](#).

If you want to align a control within a cell, or if you want a control to stretch within a cell, use the control's `Anchor` property. For more information, see [Walkthrough: Arranging Controls on Windows Forms Using a TableLayoutPanel](#).

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To edit rows and columns

1. Drag a [TableLayoutPanel](#) control from the **Toolbox** onto your form.
2. Click the [TableLayoutPanel](#) control's smart tag glyph (⌚) and select **Edit Rows and Columns** to open the **Column and Row Styles** dialog box. You can also right click on the [TableLayoutPanel](#) control and select **Edit Rows and Columns** from the shortcut menu.
3. To add or remove columns, select **Columns** from the **Member type** drop-down list box.
4. To add or remove rows, select **Rows** from the **Member type** drop-down list box.
5. Click the **Add** button to add a row or column to the end of the **Member** list.
6. Click the **Insert** button to add a row or column before the currently selected item in the list.
7. If you are adding a row or column, select the **Size Type** for the new row or column. For more information, see [SizeType](#).
8. To remove a row or column, click the **Remove** button to delete the currently selected item in the **Member** list.

See Also

[SizeType](#)

[TableLayoutPanel Control](#)

How to: Span Rows and Columns in a TableLayoutPanel Control

5/4/2018 • 1 min to read • [Edit Online](#)

Controls in a [TableLayoutPanel](#) control can span adjacent rows and columns.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To span columns and rows

1. Drag a [TableLayoutPanel](#) control from the **Toolbox** onto your form.
2. Drag a [Button](#) control from the **Toolbox** into the upper-left cell of the [TableLayoutPanel](#) control.
3. Set the [Button](#) control's **ColumnSpan** property to **2**. Note that the [Button](#) control spans the first and second columns.
4. Set the [Button](#) control's **RowSpan** property to **2**. Note that the [Button](#) control spans the first and second rows.
5. Set the [Button](#) control's **ColumnSpan** property to **1**. Note that the [Button](#) control moves into the first column and spans the first and second rows.

See Also

[TableLayoutPanel Control](#)

Walkthrough: Arranging Controls on Windows Forms Using a TableLayoutPanel

5/4/2018 • 9 min to read • [Edit Online](#)

Some applications require a form with a layout that arranges itself appropriately as the form is resized or as the contents change in size. When you need a dynamic layout and you do not want to handle [Layout](#) events explicitly in your code, consider using a layout panel.

The [FlowLayoutPanel](#) control and the [TableLayoutPanel](#) control provide intuitive ways to arrange controls on your form. Both provide an automatic, configurable ability to control the relative positions of child controls contained within them, and both give you dynamic layout features at run time, so they can resize and reposition child controls as the dimensions of the parent form change. Layout panels can be nested within layout panels, to enable the realization of sophisticated user interfaces.

The [FlowLayoutPanel](#) arranges its contents in a specific flow direction: horizontal or vertical. Its contents can be wrapped from one row to the next, or from one column to the next. Alternately, its contents can be clipped instead of wrapped. For more information, see [Walkthrough: Arranging Controls on Windows Forms Using a FlowLayoutPanel](#).

The [TableLayoutPanel](#) arranges its contents in a grid, providing functionality similar to the HTML <table> element. The [TableLayoutPanel](#) control allows you to place controls in a grid layout without requiring you to precisely specify the position of each individual control. Its cells are arranged in rows and columns, and these can have different sizes. Cells can be merged across rows and columns. Cells can contain anything a form can contain and behave in most other respects as containers.

The [TableLayoutPanel](#) control also provides a proportional resizing capability at run time, so your layout can change smoothly as your form is resized. This makes the [TableLayoutPanel](#) control well suited for purposes such as data-entry forms and localized applications. For more information, see [Walkthrough: Creating a Resizable Windows Form for Data Entry](#) and [Walkthrough: Creating a Localizable Windows Form](#).

In general, you should not use a [TableLayoutPanel](#) control as a container for the whole layout. Use [TableLayoutPanel](#) controls to provide proportional resizing capabilities to parts of the layout.

Tasks illustrated in this walkthrough include:

- Creating a Windows Forms project
- Arranging Controls in Rows and Columns
- Setting Row and Column Properties
- Spanning Rows and Columns with a Control
- Automatic Handling of Overflows
- Inserting Controls by Double-clicking Them in the Toolbox
- Inserting a Control by Drawing Its Outline
- Reassigning Existing Controls to a Different Parent

When you are finished, you will have an understanding of the role played by these important layout features.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

Creating the Project

The first step is to create the project and set up the form.

To create the project

1. Create a Windows Application project called "TableLayoutPanelExample". For more information, see [How to: Create a Windows Application Project](#).
2. Select the form in the **Windows Forms Designer**.

Arranging Controls in Rows and Columns

The **TableLayoutPanel** control allows you to easily arrange controls into rows and columns.

To arrange controls in rows and columns using a TableLayoutPanel

1. Drag a **TableLayoutPanel** control from the **Toolbox** onto your form. Note that, by default, the **TableLayoutPanel** control has four cells.
2. Drag a **Button** control from the **Toolbox** into the **TableLayoutPanel** control and drop it into one of the cells. Note that the **Button** control is created within the cell you selected.
3. Drag three more **Button** controls from the **Toolbox** into the **TableLayoutPanel** control, so that each cell contains a button.
4. Grab the vertical sizing handle between the two columns and move it to the left. Note that the **Button** controls in the first column are resized to a smaller width, while size of the **Button** controls in the second column is unchanged.
5. Grab the vertical sizing handle between the two columns and move it to the right. Note that the **Button** controls in the first column return to their original size, while the **Button** controls in the second column are moved to the right.
6. Move the horizontal sizing handle up and down to see the effect on the controls in the panel.

Positioning Controls Within Cells Using Docking and Anchoring

The anchoring behavior of child controls in a **TableLayoutPanel** differs from the behavior in other container controls. The docking behavior of child controls is the same as other container controls.

Positioning controls within cells

1. Select the first **Button** control. Change the value of its **Dock** property to **Fill**. Note that the **Button** control expands to fill its cell.
2. Select one of the other **Button** controls. Change the value of its **Anchor** property to **Right**. Note that it is moved so that its right border is near the right border of the cell. The distance between the borders is the sum of the **Button** control's **Margin** property and the panel's **Padding** property.
3. Change the value of the **Button** control's **Anchor** property to **Right** and **Left**. Note that the control is sized to the width of the cell, with the **Margin** and **Padding** values taken into account.
4. Repeat steps 2 and 3 with the **Top** and **Bottom** styles.

Setting Row and Column Properties

You can set individual properties of rows and columns by using the [RowStyles](#) and [ColumnStyles](#) collections.

To set row and column properties

1. Select the [TableLayoutPanel](#) control in the **Windows Forms Designer**.
2. In the **Properties** windows, open the [ColumnStyles](#) collection by clicking the ellipsis (...) button next to the **Columns** entry.
3. Select the first column and change the value of its [SizeType](#) property to [AutoSize](#). Click **OK** to accept the change. Note that the width of the first column is reduced to fit the [Button](#) control. Also note that the width of the column is not resizable.
4. In the **Properties** window, open the [ColumnStyles](#) collection and select the first column. Change the value of its [SizeType](#) property to [Percent](#). Click **OK** to accept the change. Resize the [TableLayoutPanel](#) control to a larger width and note that the width of the first column expands. Resize the [TableLayoutPanel](#) control to a smaller width and note that the buttons in the first column are sized to fit the cell. Also note that the width of the column is resizable.
5. In the **Properties** window, open the [ColumnStyles](#) collection and select all the listed columns. Set the value of every [SizeType](#) property to [Percent](#). Click **OK** to accept the change. Repeat with the [RowStyles](#) collection.
6. Grab one of the corner resizing handles and resize both the width and height of the [TableLayoutPanel](#) control. Note that the rows and columns are resized as the [TableLayoutPanel](#) control's size changes. Also note that the rows and columns are resizable with the horizontal and vertical sizing handles.

Spanning Rows and Columns with a Control

The [TableLayoutPanel](#) control adds several new properties to controls at design time. Two of these properties are [RowSpan](#) and [ColumnSpan](#). You can use these properties to make a control span more than one row or column.

To span rows and columns with a control

1. Select the [Button](#) control in the first row and first column.
2. In the **Properties** windows, change the value of the [ColumnSpan](#) property to **2**. Note that the [Button](#) control fills the first column and the second column. Also note than an extra row has been added to accommodate this change.
3. Repeat step 2 for the [RowSpan](#) property.

Inserting Controls by Double-clicking Them in the Toolbox

You can populate your [TableLayoutPanel](#) control by double-clicking controls in the **Toolbox**.

To insert controls by double-clicking in the Toolbox

1. Drag a [TableLayoutPanel](#) control from the **Toolbox** onto your form.
2. Double-click the [Button](#) control icon in the **Toolbox**. Note that a new button control appears in the [TableLayoutPanel](#) control's first cell.
3. Double-click several more controls in the **Toolbox**. Note that the new controls appear successively in the [TableLayoutPanel](#) control's unoccupied cells. Also note that the [TableLayoutPanel](#) control expands to accommodate the new controls if no open cells are available.

Automatic Handling of Overflows

When you are inserting controls into the **TableLayoutPanel** control, you may run out of empty cells for your new controls. The **TableLayoutPanel** control handles this situation automatically by increasing the number of cells.

To observe automatic handling of overflows

1. If there are still empty cells in the **TableLayoutPanel** control, continue inserting new **Button** controls until the **TableLayoutPanel** control is full.
2. Once the **TableLayoutPanel** control is full, double-click the **Button** icon in the **Toolbox** to insert another **Button** control. Note that the **TableLayoutPanel** control creates new cells to accommodate the new control. Insert a few more controls and observe the resizing behavior.
3. Change the value of the **TableLayoutPanel** control's **GrowStyle** property to **FixedSize**. Double-click the **Button** icon in the **Toolbox** to insert **Button** controls until the **TableLayoutPanel** control is full. Double-click the **Button** icon in the **Toolbox** again. Note that you receive an error message from the **Windows Forms Designer** informing you that additional rows and columns cannot be created.

Inserting a Control by Drawing Its Outline

You can insert a control into a **TableLayoutPanel** control and specify its size by drawing its outline in a cell.

To insert a Control by drawing its outline

1. Drag a **TableLayoutPanel** control from the **Toolbox** onto your form.
2. In the **Toolbox**, click the **Button** control icon. Do not drag it onto the form.
3. Move the mouse pointer over the **TableLayoutPanel** control. Note that the pointer changes to a crosshair with the **Button** control icon attached.
4. Click and hold the mouse button.
5. Drag the mouse pointer to draw the outline of the **Button** control. When you are satisfied with the size, release the mouse button. Note that the **Button** control is created in the cell in which you drew the control's outline.

Multiple Controls Within Cells Are Not Permitted

The **TableLayoutPanel** control can contain only one child control per cell.

To demonstrate that multiple controls within cells are not permitted

- Drag a **Button** control from the **Toolbox** into the **TableLayoutPanel** control and drop it into one of the occupied cells. Note that the **TableLayoutPanel** control does not allow you to drop the **Button** control into the occupied cell.

Swapping Controls

The **TableLayoutPanel** control enables you to swap the controls occupying two different cells.

To swap controls

- Drag one of the **Button** controls from an occupied cell and drop it onto another occupied cell. Note that the two controls are moved from one cell into the other.

Next Steps

You can achieve a complex layout using a combination of layout panels and controls. Suggestions for more exploration include:

- Try resizing one of the **Button** controls to a larger size and note the effect on the layout.

- Paste a selection of multiple controls into the [TableLayoutPanel](#) control and note how the controls are inserted.
- Layout panels can contain other layout panels. Experiment with dropping a [TableLayoutPanel](#) control into the existing control.
- Dock the [TableLayoutPanel](#) control to the parent form. Resize the form and note the effect on the layout.

See Also

[FlowLayoutPanel](#)

[TableLayoutPanel](#)

[Walkthrough: Arranging Controls on Windows Forms Using a FlowLayoutPanel](#)

[Walkthrough: Arranging Controls on Windows Forms Using Snaplines](#)

[Microsoft Windows User Experience, Official Guidelines for User Interface Developers and Designers.](#)

[Redmond, WA: Microsoft Press, 1999. \(ISBN: 0-7356-0566-1\)](#)

[Walkthrough: Creating a Resizable Windows Form for Data Entry](#)

[Walkthrough: Creating a Localizable Windows Form](#)

[Best Practices for the TableLayoutPanel Control](#)

[AutoSize Property Overview](#)

[How to: Dock Controls on Windows Forms](#)

[How to: Anchor Controls on Windows Forms](#)

[Walkthrough: Laying Out Windows Forms Controls with Padding, Margins, and the AutoSize Property](#)

TextBox Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

Windows Forms text boxes are used to get input from the user or to display text. The `TextBox` control is generally used for editable text, although it can also be made read-only. Text boxes can display multiple lines, wrap text to the size of the control, and add basic formatting. The `TextBox` control allows a single format for text displayed or entered in the control.

In This Section

[TextBox Control Overview](#)

Explains what this control is and its key features and properties.

[How to: Control the Insertion Point in a Windows Forms TextBox Control](#)

Gives directions for specifying where the insertion point appears when an edit control first gets the focus.

[How to: Create a Password Text Box with the Windows Forms TextBox Control](#)

Explains how to conceal what is typed into a text box.

[How to: Create a Read-Only Text Box](#)

Describes how to prevent the contents of a text box from being changed.

[How to: Put Quotation Marks in a String](#)

Explains adding quotation marks to a string in a text box.

[How to: Select Text in the Windows Forms TextBox Control](#)

Explains how to highlight text in a text box.

[How to: View Multiple Lines in the Windows Forms TextBox Control](#)

Describes how to make a text box scrollable.

Reference

[TextBox class](#)

Describes this class and has links to all its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

TextBox Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

Windows Forms text boxes are used to get input from the user or to display text. The [TextBox](#) control is generally used for editable text, although it can also be made read-only. Text boxes can display multiple lines, wrap text to the size of the control, and add basic formatting. The [TextBox](#) control provides a single format style for text displayed or entered into the control. To display multiple types of formatted text, use the [RichTextBox](#) control. For more information, see [RichTextBox Control Overview](#).

Working with the TextBox Control

The text displayed by the control is contained in the [Text](#) property. By default, you can enter up to 2048 characters in a text box. If you set the [Multiline](#) property to `true`, you can enter up to 32 KB of text. The [Text](#) property can be set at design time with the Properties Window, at run time in code, or by user input at run time. The current contents of a text box can be retrieved at run time by reading the [Text](#) property.

The following code example sets text in the control at run time. The `InitializeMyControl` procedure will not execute automatically; it must be called.

```
Private Sub InitializeMyControl()
    ' Put some text into the control first.
    TextBox1.Text = "This is a TextBox control."
End Sub
```

```
private void InitializeMyControl() {
    // Put some text into the control first.
    textBox1.Text = "This is a TextBox control.";
}
```

```
private:
void InitializeMyControl()
{
    // Put some text into the control first.
    textBox1->Text = "This is a TextBox control.";
}
```

See Also

[TextBox](#)

[How to: Control the Insertion Point in a Windows Forms TextBox Control](#)

[How to: Create a Password Text Box with the Windows Forms TextBox Control](#)

[How to: Create a Read-Only Text Box](#)

[How to: Put Quotation Marks in a String](#)

[How to: Select Text in the Windows Forms TextBox Control](#)

[How to: View Multiple Lines in the Windows Forms TextBox Control](#)

[TextBox Control](#)

How to: Control the Insertion Point in a Windows Forms TextBox Control

5/4/2018 • 1 min to read • [Edit Online](#)

When a Windows Forms [TextBox](#) control first receives the focus, the default insertion within the text box is to the left of any existing text. The user can move the insertion point with the keyboard or the mouse. If the text box loses and then regains the focus, the insertion point will be wherever the user last placed it.

In some cases, this behavior can be disconcerting to the user. In a word processing application, the user might expect new characters to appear after any existing text. In a data entry application, the user might expect new characters to replace any existing entry. The [SelectionStart](#) and [SelectionLength](#) properties enable you to modify the behavior to suit your purpose.

To control the insertion point in a TextBox control

1. Set the [SelectionStart](#) property to an appropriate value. Zero places the insertion point immediately to the left of the first character.
2. (Optional) Set the [SelectionLength](#) property to the length of the text you want to select.

The code below always returns the insertion point to 0. The `TextBox1_Enter` event handler must be bound to the control; for more information, see [Creating Event Handlers in Windows Forms](#).

```
Private Sub TextBox1_Enter(ByVal sender As Object, ByVal e As System.EventArgs) Handles TextBox1.Enter
    TextBox1.SelectionStart = 0
    TextBox1.SelectionLength = 0
End Sub
```

```
private void textBox1_Enter(Object sender, System.EventArgs e) {
    textBox1.SelectionStart = 0;
    textBox1.SelectionLength = 0;
}
```

```
private:
void textBox1_Enter(System::Object ^  sender,
                    System::EventArgs ^  e)
{
    textBox1->SelectionStart = 0;
    textBox1->SelectionLength = 0;
}
```

Making the Insertion Point Visible by Default

The [TextBox](#) insertion point is visible by default in a new form only if the [TextBox](#) control is first in the tab order. Otherwise, the insertion point appears only if you give the [TextBox](#) the focus with either the keyboard or the mouse.

To make the text box insertion point visible by default on a new form

- Set the [TextBox](#) control's [TabIndex](#) property to `0`.

See Also

[TextBox](#)

[TextBox Control Overview](#)

[How to: Create a Password Text Box with the Windows Forms TextBox Control](#)

[How to: Create a Read-Only Text Box](#)

[How to: Put Quotation Marks in a String](#)

[How to: Select Text in the Windows Forms TextBox Control](#)

[How to: View Multiple Lines in the Windows Forms TextBox Control](#)

[TextBox Control](#)

How to: Create a Password Text Box with the Windows Forms TextBox Control

5/4/2018 • 2 min to read • [Edit Online](#)

A password box is a Windows Forms text box that displays placeholder characters while a user types a string.

To create a password text box

1. Set the [PasswordChar](#) property of the [TextBox](#) control to a specific character.

The [PasswordChar](#) property specifies the character displayed in the text box. For example, if you want asterisks displayed in the password box, specify * for the [PasswordChar](#) property in the Properties window. Then, regardless of what character a user types in the text box, an asterisk is displayed.

2. (Optional) Set the [MaxLength](#) property. The property determines how many characters can be typed in the text box. If the maximum length is exceeded, the system emits a beep and the text box does not accept any more characters. Note that you may not wish to do this as the maximum length of a password may be of use to hackers who are trying to guess the password.

The following code example shows how to initialize a text box that will accept a string up to 14 characters long and display asterisks in place of the string. The `InitializeMyControl` procedure will not execute automatically; it must be called.

IMPORTANT

Using the [PasswordChar](#) property on a text box can help ensure that other people will not be able to determine a user's password if they observe the user entering it. This security measure does not cover any sort of storage or transmission of the password that can occur due to your application logic. Because the text entered is not encrypted in any way, you should treat it as you would any other confidential data. Even though it does not appear as such, the password is still being treated as a plain-text string (unless you have implemented some additional security measure).

```
Private Sub InitializeMyControl()
    ' Set to no text.
    TextBox1.Text = ""
    ' The password character is an asterisk.
    TextBox1.PasswordChar = "*"
    ' The control will allow no more than 14 characters.
    TextBox1.MaxLength = 14
End Sub
```

```
private void InitializeMyControl()
{
    // Set to no text.
    textBox1.Text = "";
    // The password character is an asterisk.
    textBox1.PasswordChar = '*';
    // The control will allow no more than 14 characters.
    textBox1.MaxLength = 14;
}
```

```
private:  
    void InitializeMyControl()  
    {  
        // Set to no text.  
        textBox1->Text = "";  
        // The password character is an asterisk.  
        textBox1->PasswordChar = '*';  
        // The control will allow no more than 14 characters.  
        textBox1->MaxLength = 14;  
    }
```

See Also

[TextBox](#)

[TextBox Control Overview](#)

[How to: Control the Insertion Point in a Windows Forms TextBox Control](#)

[How to: Create a Read-Only Text Box](#)

[How to: Put Quotation Marks in a String](#)

[How to: Select Text in the Windows Forms TextBox Control](#)

[How to: View Multiple Lines in the Windows Forms TextBox Control](#)

[TextBox Control](#)

How to: Create a Read-Only Text Box (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

You can transform an editable Windows Forms text box into a read-only control. For example, the text box may display a value that is usually edited but may not be currently, due to the state of the application.

To create a read-only text box

1. Set the [TextBox](#) control's [ReadOnly](#) property to `true`. With the property set to `true`, users can still scroll and highlight text in a text box without allowing changes. A **Copy** command is functional in a text box, but **Cut** and **Paste** commands are not.

NOTE

The [ReadOnly](#) property only affects user interaction at run time. You can still change text box contents programmatically at run time by changing the [Text](#) property of the text box.

See Also

[TextBox](#)

[TextBox Control Overview](#)

[How to: Control the Insertion Point in a Windows Forms TextBox Control](#)

[How to: Create a Password Text Box with the Windows Forms TextBox Control](#)

[How to: Put Quotation Marks in a String](#)

[How to: Select Text in the Windows Forms TextBox Control](#)

[How to: View Multiple Lines in the Windows Forms TextBox Control](#)

[TextBox Control](#)

How to: Put Quotation Marks in a String (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

Sometimes you might want to place quotation marks (" ") in a string of text. For example:

She said, "You deserve a treat!"

As an alternative, you can also use the [Quote](#) field as a constant.

To place quotation marks in a string in your code

1. In Visual Basic, insert two quotation marks in a row as an embedded quotation mark. In Visual C# and Visual C++, insert the escape sequence \" as an embedded quotation mark. For example, to create the preceding string, use the following code.

```
Private Sub InsertQuote()
    TextBox1.Text = "She said, ""You deserve a treat!"" "
End Sub
```

```
private void InsertQuote(){
    textBox1.Text = "She said, \"You deserve a treat!\" ";
}
```

```
private:
void InsertQuote()
{
    textBox1->Text = "She said, \"You deserve a treat!\" ";
}
```

-or-

2. Insert the ASCII or Unicode character for a quotation mark. In Visual Basic, use the ASCII character (34). In Visual C#, use the Unicode character (\u0022).

```
Private Sub InsertAscii()
    TextBox1.Text = "She said, " & Chr(34) & "You deserve a treat!" & Chr(34)
End Sub
```

```
private void InsertAscii(){
    textBox1.Text = "She said, " + '\u0022' + "You deserve a treat!" + '\u0022';
}
```

NOTE

In this example, you cannot use \u0022 because you cannot use a universal character name that designates a character in the basic character set. Otherwise, you produce C3851. For more information, see [Compiler Error C3851](#).

-or-

3. You can also define a constant for the character, and use it where needed.

```
Const quote As String = """"
TextBox1.Text = "She said, " & quote & "You deserve a treat!" & quote
```

```
const string quote = "\"";
textBox1.Text = "She said, " + quote + "You deserve a treat!" + quote ;
```

```
const String^ quote = "\"";
textBox1->Text = String::Concat("She said, ",
    const_cast<String^>(quote), "You deserve a treat!",
    const_cast<String^>(quote));
```

See Also

[TextBox](#)

[Quote](#)

[TextBox Control Overview](#)

[How to: Control the Insertion Point in a Windows Forms TextBox Control](#)

[How to: Create a Password Text Box with the Windows Forms TextBox Control](#)

[How to: Create a Read-Only Text Box](#)

[How to: Select Text in the Windows Forms TextBox Control](#)

[How to: View Multiple Lines in the Windows Forms TextBox Control](#)

[TextBox Control](#)

How to: Select Text in the Windows Forms TextBox Control

5/4/2018 • 1 min to read • [Edit Online](#)

You can select text programmatically in the Windows Forms `TextBox` control. For example, if you create a function that searches text for a particular string, you can select the text to visually alert the reader of the found string's position.

To select text programmatically

1. Set the `SelectionStart` property to the beginning of the text you want to select.

The `SelectionStart` property is a number that indicates the insertion point within the string of text, with 0 being the left-most position. If the `SelectionStart` property is set to a value equal to or greater than the number of characters in the text box, the insertion point is placed after the last character.

2. Set the `SelectionLength` property to the length of the text you want to select.

The `SelectionLength` property is a numeric value that sets the width of the insertion point. Setting the `SelectionLength` to a number greater than 0 causes that number of characters to be selected, starting from the current insertion point.

3. (Optional) Access the selected text through the `SelectedText` property.

The code below selects the contents of a text box when the control's `Enter` event occurs. This example checks if the text box has a value for the `Text` property that is not `null` or an empty string. When the text box receives the focus, the current text in the text box is selected. The `TextBox1_Enter` event handler must be bound to the control; for more information, see [How to: Create Event Handlers at Run Time for Windows Forms](#).

To test this example, press the Tab key until the text box has the focus. If you click in the text box, the text is unselected.

```
Private Sub TextBox1_Enter(ByVal sender As Object, ByVal e As System.EventArgs) Handles TextBox1.Enter
    If (Not String.IsNullOrEmpty(TextBox1.Text)) Then
        TextBox1.SelectionStart = 0
        TextBox1.SelectionLength = TextBox1.Text.Length
    End If
End Sub
```

```
private void textBox1_Enter(object sender, System.EventArgs e){
    if (!String.IsNullOrEmpty(textBox1.Text))
    {
        textBox1.SelectionStart = 0;
        textBox1.SelectionLength = textBox1.Text.Length;
    }
}
```

```
private:  
    void textBox1_Enter(System::Object ^ sender,  
        System::EventArgs ^ e) {  
        if (!System::String::IsNullOrEmpty(textBox1->Text))  
        {  
            textBox1->SelectionStart = 0;  
            textBox1->SelectionLength = textBox1->Text->Length;  
        }  
    }  
}
```

See Also

[TextBox](#)

[TextBox Control Overview](#)

[How to: Control the Insertion Point in a Windows Forms TextBox Control](#)

[How to: Create a Password Text Box with the Windows Forms TextBox Control](#)

[How to: Create a Read-Only Text Box](#)

[How to: Put Quotation Marks in a String](#)

[How to: View Multiple Lines in the Windows Forms TextBox Control](#)

[TextBox Control](#)

How to: View Multiple Lines in the Windows Forms TextBox Control

5/4/2018 • 1 min to read • [Edit Online](#)

By default, the Windows Forms [TextBox](#) control displays a single line of text and does not display scroll bars. If the text is longer than the available space, only part of the text is visible. You can change this default behavior by setting the [Multiline](#), [WordWrap](#), and [ScrollBars](#) properties to appropriate values.

To display a carriage return in the TextBox control

- To display a carriage return in a multi-line [TextBox](#), use the [NewLine](#) property.

Be aware that the interpretation of escape characters (\) is language-specific. Visual Basic uses

`Chr$(13) & Chr$(10)` for the carriage return and linefeed character combination.

To view multiple lines in the TextBox control

1. Set the [Multiline](#) property to `true`. If [WordWrap](#) is `true` (the default), then the text in the control will appear as one or more paragraphs; otherwise it will appear as a list, in which some lines may be clipped at the edge of the control.
2. Set the [ScrollBars](#) property to an appropriate value.

VALUE	DESCRIPTION
<code>None</code>	Use this value if the text will be a paragraph that almost always fits the control. The user can use the mouse pointer to move around inside the control if the text is too long to display all at once.
<code>Horizontal</code>	Use this value if you want to display a list of lines, some of which may be longer than the width of the TextBox control.
<code>Both</code>	Use this value if the list may be longer than the height of the control.

3. Set the [WordWrap](#) property to an appropriate value.

VALUE	DESCRIPTION
<code>false</code>	Text in the control will not automatically be wrapped, so it will scroll to the right until a line break is reached. Use this value if you chose <code>Horizontal</code> scroll bars or <code>Both</code> , above.
<code>true</code> (default)	The horizontal scrollbar will not appear. Use this value if you chose <code>Vertical</code> scroll bars or <code>None</code> , above, to display one or more paragraphs.

See Also

[TextBox](#)

[TextBox Control Overview](#)

[How to: Control the Insertion Point in a Windows Forms TextBox Control](#)

[How to: Create a Password Text Box with the Windows Forms TextBox Control](#)

[How to: Create a Read-Only Text Box](#)

[How to: Put Quotation Marks in a String](#)

[How to: Select Text in the Windows Forms TextBox Control](#)

[TextBox Control](#)

Timer Component (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [Timer](#) is a component that raises an event at regular intervals. This component is designed for a Windows Forms environment.

In This Section

[Timer Component Overview](#)

Introduces the general concepts of the [Timer](#) component, which allows you to set up your application to respond to periodic events.

[Limitations of the Windows Forms Timer Component's Interval Property](#)

Describes known limitations of the timer's interval that may affect how you can use it.

[How to: Run Procedures at Set Intervals with the Windows Forms Timer Component](#)

Describes how to react to timed intervals in your Windows-based applications.

Reference

[System.Windows.Forms.Timer](#) class

Provides reference information on the class, used for Windows Forms timers, and its members.

[System.Timers.Timer](#) class

Provides reference information on the [System.Timers.Timer](#) class that is used by server-based timers.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

[Timer Control for Visual Basic 6.0 Users](#)

Describes how timer functionality has changed in Visual Basic as compared to previous versions.

Timer Component Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [Timer](#) is a component that raises an event at regular intervals. This component is designed for a Windows Forms environment. If you need a timer that is suitable for a server environment, see [Introduction to Server-Based Timers](#).

Key Properties, Methods, and Events

The length of the intervals is defined by the [Interval](#) property, whose value is in milliseconds. When the component is enabled, the [Tick](#) event is raised every interval. This is where you would add code to be executed. For more information, see [How to: Run Procedures at Set Intervals with the Windows Forms Timer Component](#). The key methods of the [Timer](#) component are [Start](#) and [Stop](#), which turn the timer on and off. When the timer is switched off, it resets; there is no way to pause a [Timer](#) component.

See Also

[Timer](#)

[Timer Component](#)

[Limitations of the Windows Forms Timer Component's Interval Property](#)

Limitations of the Windows Forms Timer Component's Interval Property

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [Timer](#) component has an [Interval](#) property that specifies the number of milliseconds that pass between one timer event and the next. Unless the component is disabled, a timer continues to receive the [Tick](#) event at roughly equal intervals of time.

This component is designed for a Windows Forms environment. If you need a timer that is suitable for a server environment, see [Introduction to Server-Based Timers](#).

The Interval Property

The [Interval](#) property has a few limitations to consider when you are programming a [Timer](#) component:

- If your application or another application is making heavy demands on the system — such as long loops, intensive calculations, or drive, network, or port access — your application may not get timer events as often as the [Interval](#) property specifies.
- The interval is not guaranteed to elapse exactly on time. To ensure accuracy, the timer should check the system clock as needed, rather than try to keep track of accumulated time internally.
- The precision of the [Interval](#) property is in milliseconds. Some computers provide a high-resolution counter that has a resolution higher than milliseconds. The availability of such a counter depends on the processor hardware of your computer. For more information, see article 172338, "How To Use QueryPerformanceCounter to Time Code," in the Microsoft Knowledge Base at <http://support.microsoft.com>.

See Also

[Timer](#)

[Timer Component](#)

[Timer Component Overview](#)

How to: Run Procedures at Set Intervals with the Windows Forms Timer Component

5/4/2018 • 4 min to read • [Edit Online](#)

You might sometimes want to create a procedure that runs at specific time intervals until a loop has finished or that runs when a set time interval has elapsed. The [Timer](#) component makes such a procedure possible.

This component is designed for a Windows Forms environment. If you need a timer that is suitable for a server environment, see [Introduction to Server-Based Timers](#).

NOTE

There are some limitations when using the [Timer](#) component. For more information, see [Limitations of the Windows Forms Timer Component's Interval Property](#).

To run a procedure at set intervals with the Timer component

1. Add a [Timer](#) to your form. See the following Example section for an illustration of how to do this programmatically. Visual Studio also has support for adding components to a form. Also see [How to: Add Controls Without a User Interface to Windows Forms](#).
2. Set the [Interval](#) property (in milliseconds) for the timer. This property determines how much time will pass before the procedure is run again.

NOTE

The more often a timer event occurs, the more processor time is used in responding to the event. This can slow down overall performance. Do not set a smaller interval than you need.

3. Write appropriate code in the [Tick](#) event handler. The code you write in this event will run at the interval specified in the [Interval](#) property.
4. Set the [Enabled](#) property to `true` to start the timer. The [Tick](#) event will begin to occur, running your procedure at the set interval.
5. At the appropriate time, set the [Enabled](#) property to `false` to stop the procedure from running again. Setting the interval to `0` does not cause the timer to stop.

Example

This first code example tracks the time of day in one-second increments. It uses a [Button](#), a [Label](#), and a [Timer](#) component on a form. The [Interval](#) property is set to 1000 (equal to one second). In the [Tick](#) event, the label's caption is set to the current time. When the button is clicked, the [Enabled](#) property is set to `false`, stopping the timer from updating the label's caption. The following code example requires that you have a form with a [Button](#) control named `Button1`, a [Timer](#) control named `Timer1`, and a [Label](#) control named `Label1`.

```

Private Sub InitializeTimer()
    ' Run this procedure in an appropriate event.
    ' Set to 1 second.
    Timer1.Interval = 1000
    ' Enable timer.
    Timer1.Enabled = True
    Button1.Text = "Enabled"
End Sub
X
Private Sub Timer1_Tick(ByVal Sender As Object, ByVal e As EventArgs) Handles Timer1.Tick
    ' Set the caption to the current time.
    Label1.Text = DateTime.Now
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    If Button1.Text = "Stop" Then
        Button1.Text = "Start"
        Timer1.Enabled = False
    Else
        Button1.Text = "Stop"
        Timer1.Enabled = True
    End If
End Sub

```

```

private void InitializeTimer()
{
    // Call this procedure when the application starts.
    // Set to 1 second.
    Timer1.Interval = 1000;
    Timer1.Tick += new EventHandler(Timer1_Tick);

    // Enable timer.
    Timer1.Enabled = true;

    Button1.Text = "Stop";
    Button1.Click += new EventHandler(Button1_Click);
}

private void Timer1_Tick(object Sender, EventArgs e)
{
    // Set the caption to the current time.
    Label1.Text = DateTime.Now.ToString();
}

private void Button1_Click(object sender, EventArgs e)
{
    if ( Button1.Text == "Stop" )
    {
        Button1.Text = "Start";
        Timer1.Enabled = false;
    }
    else
    {
        Button1.Text = "Stop";
        Timer1.Enabled = true;
    }
}

```

```

private:
    void InitializeTimer()
    {
        // Run this procedure in an appropriate event.
        // Set to 1 second.
        timer1->Interval = 1000;
        // Enable timer.
        timer1->Enabled = true;
        this->timer1->Tick += gcnew System::EventHandler(this,
            &Form1::timer1_Tick);

        button1->Text = S"Stop";
        this->button1->Click += gcnew System::EventHandler(this,
            &Form1::button1_Click);
    }

    void timer1_Tick(System::Object ^ sender,
        System::EventArgs ^ e)
    {
        // Set the caption to the current time.
        label1->Text = DateTime::Now.ToString();
    }

    void button1_Click(System::Object ^ sender,
        System::EventArgs ^ e)
    {
        if ( button1->Text == "Stop" )
        {
            button1->Text = "Start";
            timer1->Enabled = false;
        }
        else
        {
            button1->Text = "Stop";
            timer1->Enabled = true;
        }
    }
}

```

Example

This second code example runs a procedure every 600 milliseconds until a loop has finished. The following code example requires that you have a form with a [Button](#) control named `Button1`, a [Timer](#) control named `Timer1`, and a [Label](#) control named `Label1`.

```

' This variable will be the loop counter.
Private counter As Integer

Private Sub InitializeTimer()
    ' Run this procedure in an appropriate event.
    counter = 0
    Timer1.Interval = 600
    Timer1.Enabled = True
End Sub

Private Sub Timer1_Tick(ByVal sender As Object, ByVal e As System.EventArgs) Handles Timer1.Tick
    If counter >= 10 Then
        ' Exit loop code.
        Timer1.Enabled = False
        counter = 0
    Else
        ' Run your procedure here.
        ' Increment counter.
        counter = counter + 1
        Label1.Text = "Procedures Run: " & counter.ToString()
    End If
End Sub

```

```

// This variable will be the loop counter.
private int counter;

private void InitializeTimer()
{
    // Run this procedure in an appropriate event.
    counter = 0;
    timer1.Interval = 600;
    timer1.Enabled = true;
    // Hook up timer's tick event handler.
    this.timer1.Tick += new System.EventHandler(this.timer1_Tick);
}

private void timer1_Tick(object sender, System.EventArgs e)
{
    if (counter >= 10)
    {
        // Exit loop code.
        timer1.Enabled = false;
        counter = 0;
    }
    else
    {
        // Run your procedure here.
        // Increment counter.
        counter = counter + 1;
        label1.Text = "Procedures Run: " + counter.ToString();
    }
}

```

```
private:
    int counter;

    void InitializeTimer()
    {
        // Run this procedure in an appropriate event.
        counter = 0;
        timer1->Interval = 600;
        timer1->Enabled = true;
        // Hook up timer's tick event handler.
        this->timer1->Tick += gcnew System::EventHandler(this, &Form1::timer1_Tick);
    }

    void timer1_Tick(System::Object ^ sender,
                     System::EventArgs ^ e)
    {
        if (counter >= 10)
        {
            // Exit loop code.
            timer1->Enabled = false;
            counter = 0;
        }
        else
        {
            // Run your procedure here.
            // Increment counter.
            counter = counter + 1;
            label1->Text = String::Concat("Procedures Run: ",
                counter.ToString());
        }
    }
}
```

See Also

[Timer](#)

[Timer Component](#)

[Timer Component Overview](#)

ToolBar Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

NOTE

The [ToolStrip](#) control replaces and adds functionality to the [ToolBar](#) control; however, the [ToolBar](#) control is retained for both backward compatibility and future use, if you choose.

The Windows Forms [ToolBar](#) control is used on forms as a control bar that displays a row of drop-down menus and bitmapped buttons that activate commands. Thus, clicking a toolbar button is equivalent to choosing a menu command. The buttons can be configured to appear and behave as push buttons, drop-down menus, or separators. Typically, a toolbar contains buttons and menus that correspond to items in an application's menu structure, providing quick access to an application's most frequently used functions and commands.

NOTE

The [ToolBar](#) control's [DropDownMenu](#) property takes an instance of the [ContextMenu](#) class as a reference. Carefully consider the reference you pass when implementing this sort of button on toolbars in your application, as the property will accept any object that inherits from the [Menu](#) class.

In This Section

[ToolBar Control Overview](#)

Introduces the general concepts of the [ToolBar](#) control, which allows you to design custom toolbars that your users can work with.

[How to: Add Buttons to a ToolBar Control](#)

Describes how to add buttons to a [ToolBar](#) control.

[How to: Define an Icon for a ToolBar Button](#)

Describes how to display icons within a [ToolBar](#) control's buttons.

[How to: Trigger Menu Events for Toolbar Buttons](#)

Gives directions on writing code to interpret which button the user clicks in a [ToolBar](#) control.

Also see [How to: Define an Icon for a ToolBar Button Using the Designer](#), [How to: Add Buttons to a ToolBar Control Using the Designer](#).

Reference

[ToolBar class](#)

Provides reference information on the class and its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

[ToolStrip Control](#)

Describes toolbars that can host menus, controls, and user controls in Windows Forms applications.

ToolBar Control Overview (Windows Forms)

5/4/2018 • 2 min to read • [Edit Online](#)

NOTE

The [ToolStrip](#) control replaces and adds functionality to the [ToolBar](#) control; however, the [ToolBar](#) control is retained for both backward compatibility and future use, if you choose.

The Windows Forms [ToolBar](#) control is used on forms as a control bar that displays a row of drop-down menus and bitmapped buttons that activate commands. Thus, clicking a toolbar button can be an equivalent to choosing a menu command. The buttons can be configured to appear and behave as pushbuttons, drop-down menus, or separators. Typically, a toolbar contains buttons and menus that correspond to items in an application's menu structure, providing quick access to an application's most frequently used functions and commands.

Working with the ToolBar Control

A [ToolBar](#) control is usually "docked" along the top of its parent window, but it can also be docked to any side of the window. A toolbar can display tooltips when the user points the mouse pointer at a toolbar button. A [ToolTip](#) is a small pop-up window that briefly describes the button or menu's purpose. To display [ToolTips](#), the [ShowToolTips](#) property must be set to `true`.

NOTE

Certain applications feature controls very similar to the toolbar that have the ability to "float" above the application window and be repositioned. The Windows Forms [ToolBar](#) control is not able to do these actions.

When the [Appearance](#) property is set to [ToolBarAppearance](#), the toolbar buttons appear raised and three-dimensional. You can set the [Appearance](#) property of the toolbar to [ToolBarAppearance](#) to give the toolbar and its buttons a flat appearance. When the mouse pointer moves over a flat button, the button's appearance changes to three-dimensional. Toolbar buttons can be divided into logical groups by using separators. A separator is a toolbar button with the [Style](#) property set to [ToolBarButtonStyle](#). It appears as empty space on the toolbar. When the toolbar has a flat appearance, button separators appear as lines rather than spaces between the buttons.

The [ToolBar](#) control allows you to create toolbars by adding [Button](#) objects to a [Buttons](#) collection. You can use the Collection Editor to add buttons to a [ToolBar](#) control; each [Button](#) object should have text or an image assigned, although you can assign both. The image is supplied by an associated [ImageList](#) component. At run time, you can add or remove buttons from the [ToolBar.ToolBarButtonCollection](#) using the [Add](#) and [Remove](#) methods. To program the buttons of a [ToolBar](#), add code to the [ButtonClick](#) events of the [ToolBar](#), using the [Button](#) property of the [ToolBarButtonClickEventArgs](#) class to determine which button was clicked.

See Also

[ToolBar](#)

[ToolBar Control](#)

[How to: Add Buttons to a ToolBar Control](#)

[How to: Define an Icon for a ToolBar Button](#)

[How to: Trigger Menu Events for Toolbar Buttons](#)

How to: Add Buttons to a ToolBar Control

5/4/2018 • 2 min to read • [Edit Online](#)

NOTE

The [ToolStrip](#) control replaces and adds functionality to the [ToolBar](#) control; however, the [ToolBar](#) control is retained for both backward compatibility and future use, if you choose.

An integral part of the [ToolBar](#) control is the buttons you add to it. These can be used to provide easy access to menu commands or, alternately, they can be placed in another area of the user interface of your application to expose commands to your users that are not available in the menu structure.

The examples below assume that a [ToolBar](#) control has been added to a Windows Form (`Form1`).

To add buttons programmatically

1. In a procedure, create toolbar buttons by adding them to the [ToolBar.Buttons](#) collection.
2. Specify property settings for an individual button by passing the button's index via the [Buttons](#) property.

The example below assumes a form with a [ToolBar](#) control already added.

NOTE

The [ToolBar.Buttons](#) collection is a zero-based collection, so code should proceed accordingly.

```
Public Sub CreateToolBarButtons()
    ' Create buttons and set text property.
    ToolBar1.Buttons.Add("One")
    ToolBar1.Buttons.Add("Two")
    ToolBar1.Buttons.Add("Three")
    ToolBar1.Buttons.Add("Four")
    ' Set properties of StatusBar panels.
    ' Set Style property.
    ToolBar1.Buttons(0).Style = ToolBarButtonStyle.PushButton
    ToolBar1.Buttons(1).Style = ToolBarButtonStyle.Separator
    ToolBar1.Buttons(2).Style = ToolBarButtonStyle.ToggleButton
    ToolBar1.Buttons(3).Style = ToolBarButtonStyle.DropDownButton
    ' Set the ToggleButton's PartialPush property.
    ToolBar1.Buttons(2).PartialPush = True
    ' Instantiate a ContextMenu component and menu items.
    ' Set the DropDownButton's DropDownMenu property to the context menu.
    Dim cm As New ContextMenu()
    Dim miOne As New MenuItem("One")
    Dim miTwo As New MenuItem("Two")
    Dim miThree As New MenuItem("Three")
    cm.MenuItems.Add(miOne)
    cm.MenuItems.Add(miTwo)
    cm.MenuItems.Add(miThree)
    ToolBar1.Buttons(3).DropDownMenu = cm
    ' Set the PushButton's Pushed property.
    ToolBar1.Buttons(0).Pushed = True
    ' Set the ToolTipText property of one of the buttons.
    ToolBar1.Buttons(1).ToolTipText = "Button 2"
End Sub
```

```
public void CreateToolBarButtons()
{
    // Create buttons and set text property.
    toolBar1.Buttons.Add("One");
    toolBar1.Buttons.Add("Two");
    toolBar1.Buttons.Add("Three");
    toolBar1.Buttons.Add("Four");

    // Set properties of StatusBar panels.
    // Set Style property.
    toolBar1.Buttons[0].Style = ToolBarButtonStyle.PushButton;
    toolBar1.Buttons[1].Style = ToolBarButtonStyle.Separator;
    toolBar1.Buttons[2].Style = ToolBarButtonStyle.ToggleButton;
    toolBar1.Buttons[3].Style = ToolBarButtonStyle.DropDownButton;

    // Set the ToggleButton's PartialPush property.
    toolBar1.Buttons[2].PartialPush = true;

    // Instantiate a ContextMenu component and menu items.
    // Set the DropDownButton's DropDownMenu property to
    // the context menu.
    ContextMenu cm = new ContextMenu();
    MenuItem miOne = new MenuItem("One");
    MenuItem miTwo = new MenuItem("Two");
    MenuItem miThree = new MenuItem("Three");
    cm.MenuItems.Add(miOne);
    cm.MenuItems.Add(miTwo);
    cm.MenuItems.Add(miThree);

    toolBar1.Buttons[3].DropDownMenu = cm;
    // Set the PushButton's Pushed property.
    toolBar1.Buttons[0].Pushed = true;
    // Set the ToolTipText property of 1 of the buttons.
    toolBar1.Buttons[1].ToolTipText = "Button 2";
}
```

```

public:
    void CreateToolBarButtons()
    {
        // Create buttons and set text property.
        toolBar1->Buttons->Add( "One" );
        toolBar1->Buttons->Add( "Two" );
        toolBar1->Buttons->Add( "Three" );
        toolBar1->Buttons->Add( "Four" );

        // Set properties of StatusBar panels.
        // Set Style property.
        toolBar1->Buttons[0]->Style = ToolBarButtonStyle::PushButton;
        toolBar1->Buttons[1]->Style = ToolBarButtonStyle::Separator;
        toolBar1->Buttons[2]->Style = ToolBarButtonStyle::ToggleButton;
        toolBar1->Buttons[3]->Style = ToolBarButtonStyle::DropDownButton;

        // Set the ToggleButton's PartialPush property.
        toolBar1->Buttons[2]->PartialPush = true;

        // Instantiate a ContextMenu component and menu items.
        // Set the DropDownButton's DropDownMenu property to
        // the context menu.
        System::Windows::Forms::ContextMenu^ cm = gcnew System::Windows::Forms::ContextMenu;
        MenuItem^ miOne = gcnew MenuItem( "One" );
        MenuItem^ miTwo = gcnew MenuItem( "Two" );
        MenuItem^ miThree = gcnew MenuItem( "Three" );
        cm->MenuItems->Add( miOne );
        cm->MenuItems->Add( miTwo );
        cm->MenuItems->Add( miThree );
        toolBar1->Buttons[3]->DropDownMenu = cm;

        // Set the PushButton's Pushed property.
        toolBar1->Buttons[0]->Pushed = true;

        // Set the ToolTipText property of 1 of the buttons.
        toolBar1->Buttons[1]->ToolTipText = "Button 2";
    }

```

See Also

[ToolBar](#)

[How to: Define an Icon for a ToolBar Button](#)

[How to: Trigger Menu Events for Toolbar Buttons](#)

[ToolBar Control Overview](#)

[ToolBar Control](#)

How to: Add Buttons to a ToolBar Control Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

NOTE

The [ToolStrip](#) control replaces and adds functionality to the [ToolBar](#) control; however, the [ToolBar](#) control is retained for both backward compatibility and future use, if you choose.

An integral part of the [ToolBar](#) control is the buttons you add to it. These can be used to provide easy access to menu commands or, alternately, they can be placed in another area of the user interface of your application to expose commands to your users that are not available in the menu structure.

The following procedure requires a **Windows Application** project with a form containing a [ToolBar](#) control. For information about setting up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To add buttons at design time

1. Select the [ToolBar](#) control.
2. In the **Properties** window, click the [Buttons](#) property to select it and click the **Ellipsis** (...) button to open the **ToolBarButton Collection Editor**.
3. Use the **Add** and **Remove** buttons to add and remove buttons from the [ToolBar](#) control.
4. Configure the properties of the individual buttons in the **Properties** window that appears in the pane on the right side of the editor. The following table shows some important properties to consider.

PROPERTY	DESCRIPTION
DropDownMenu	Sets the menu to be displayed in the drop-down toolbar button. The toolbar button's Style property must be set to DropDownButton . This property takes an instance of the ContextMenu class as a reference.
PartialPush	Sets whether a toggle-style toolbar button is partially pushed. The toolbar button's Style property must be set to ToggleButton .
Pushed	Sets whether a toggle-style toolbar button is currently in the pushed state. The toolbar button's Style property must be set to ToggleButton or PushButton .

PROPERTY	DESCRIPTION
Style	Sets the style of the toolbar button. Must be one of the values in the ToolBarButtonStyle enumeration.
Text	The text string displayed by the button.
ToolTipText	The text that appears as a ToolTip for the button.

5. Click **OK** to close the dialog box and create the panels you specified.

See Also

[ToolBar](#)

[How to: Define an Icon for a ToolBar Button](#)

[How to: Trigger Menu Events for Toolbar Buttons](#)

[ToolBar Control Overview](#)

[ToolBar Control](#)

How to: Define an Icon for a ToolBar Button

5/4/2018 • 2 min to read • [Edit Online](#)

NOTE

The [ToolStrip](#) control replaces and adds functionality to the [ToolBar](#) control; however, the [ToolBar](#) control is retained for both backward compatibility and future use, if you choose.

[ToolBar](#) buttons are able to display icons within them for easy identification by users. This is achieved through adding images to the [ImageList Component](#) component and then associating the [ImageList](#) component with the [ToolBar](#) control.

To set an icon for a toolbar button programmatically

1. In a procedure, instantiate an [ImageList](#) component and a [ToolBar](#) control.
2. In the same procedure, assign an image to the [ImageList](#) component.
3. In the same procedure, assign the [ImageList](#) control to the [ToolBar](#) control and assign the [ImageIndex](#) property of the individual toolbar buttons.

In the following code example, the path set for the location of the image is the **My Documents** folder. This is done, because you can assume that most computers running the Windows operating system will include this directory. This also allows users with minimal system access levels to safely run the application. The example below assumes a form with a [PictureBox](#) control already added.

Following the steps above, you should have written code similar to that displayed below.

```
Public Sub InitializeMyToolBar()
    ' Instantiate an ImageList component and a ToolBar control.
    Dim ToolBar1 As New ToolBar
    Dim ImageList1 As New ImageList
    ' Assign an image to the ImageList component.
    ' You should replace the bold image
    ' in the sample below with an icon of your own choosing.
    Dim myImage As System.Drawing.Image = _
        Image.FromFile Image.FromFile _
        (System.Environment.GetFolderPath _
        (System.Environment.SpecialFolder.Personal) _
        & "\Image.gif")
    ImageList1.Images.Add(myImage)
    ' Create a ToolBarButton.
    Dim ToolBarButton1 As New ToolBarButton()
    ' Add the ToolBarButton to the ToolBar.
    ToolBar1.Buttons.Add(toolBarButton1)
    ' Assign an ImageList to the ToolBar.
    ToolBar1.ImageList = ImageList1
    ' Assign the ImageIndex property of the ToolBarButton.
    ToolBarButton1.ImageIndex = 0
End Sub
```

```

public void InitializeMyToolBar()
{
    // Instantiate an ImageList component and a ToolBar control.
    ToolBar toolBar1 = new ToolBar();
    ImageList imageList1 = new ImageList();
    // Assign an image to the ImageList component.
    // You should replace the bold image
    // in the sample below with an icon of your own choosing.
    // Note the escape character used (@) when specifying the path.
    Image myImage = Image.FromFile
    (System.Environment.GetFolderPath
    (System.Environment.SpecialFolder.Personal)
    + @"\Image.gif");
    imageList1.Images.Add(myImage);
    // Create a ToolBarButton.
    ToolBarButton toolBarButton1 = new ToolBarButton();
    // Add the ToolBarButton to the ToolBar.
    toolBar1.Buttons.Add(toolBarButton1);
    // Assign an ImageList to the ToolBar.
    toolBar1.ImageList = imageList1;
    // Assign ImageIndex property of the ToolBarButton.
    toolBarButton1.ImageIndex = 0;
}

```

```

public:
void InitializeMyToolBar()
{
    // Instantiate an ImageList component and a ToolBar control.
    ToolBar ^ toolBar1 = gcnew ToolBar();
    ImageList ^ imageList1 = gcnew ImageList();
    // Assign an image to the ImageList component.
    // You should replace the bold image
    // in the sample below with an icon of your own choosing.
    Image ^ myImage = Image::FromFile(String::Concat
        (System::Environment::GetFolderPath
        (System::Environment::SpecialFolder::Personal),
        "\\Image.gif"));
    imageList1->Images->Add(myImage);
    // Create a ToolBarButton.
    ToolBarButton ^ toolBarButton1 = gcnew ToolBarButton();
    // Add the ToolBarButton to the ToolBar.
    toolBar1->Buttons->Add(toolBarButton1);
    // Assign an ImageList to the ToolBar.
    toolBar1->ImageList = imageList1;
    // Assign ImageIndex property of the ToolBarButton.
    toolBarButton1->ImageIndex = 0;
}

```

See Also

[ToolBar](#)

[How to: Trigger Menu Events for Toolbar Buttons](#)

[ToolBar Control](#)

[ImageList Component](#)

How to: Define an Icon for a ToolBar Button Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

NOTE

The [ToolStrip](#) control replaces and adds functionality to the [ToolBar](#) control; however, the [ToolBar](#) control is retained for both backward compatibility and future use, if you choose.

ToolBar buttons are able to display icons within them for easy identification by users. This is achieved through adding images to the [ImageList](#) component and associating it with the [ToolBar](#) control.

The following procedure requires a **Windows Application** project with a form containing a [ToolBar](#) control and an [ImageList](#) component. For information about setting up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To set an icon for a toolbar button at design time

1. Add images to the [ImageList](#) component. For more information, see [How to: Add or Remove ImageList Images with the Designer](#).
2. Select the [ToolBar](#) control on your form.
3. In the **Properties** window, set the [ToolBar](#) control's [ImageList](#) property to the [ImageList](#) component.
4. Click the [ToolBar](#) control's [Buttons](#) property to select it, and click the ellipsis () button to open the **ToolBarButton Collection Editor**.
5. Use the **Add** button to add buttons to the [ToolBar](#) control.
6. In the **Properties** window that appears in the pane on the right side of the **ToolBarButton Collection Editor**, set the [ImageIndex](#) property of each toolbar button to one of the values in the list, which is drawn from the images you added to the [ImageList](#) component.

See Also

[ToolBar](#)

[How to: Trigger Menu Events for Toolbar Buttons](#)

[ToolBar Control](#)

[ImageList Component](#)

How to: Trigger Menu Events for Toolbar Buttons

5/4/2018 • 2 min to read • [Edit Online](#)

NOTE

The [ToolStrip](#) control replaces and adds functionality to the [ToolBar](#) control; however, the [ToolBar](#) control is retained for both backward compatibility and future use, if you choose.

If your Windows Form features a [ToolBar](#) control with toolbar buttons, you will want to know which button the user clicks.

On the [ButtonClick](#) event of the [ToolBar](#) control, you can evaluate the [Button](#) property of the [ToolBarButtonClickEventArgs](#) class. In the example below, a message box is shown, indicating which button was clicked. For details, see [MessageBox](#).

The example below assumes a [ToolBar](#) control has been added to a Windows Form.

To handle the Click event on a toolbar

1. In a procedure, add toolbar buttons to the [ToolBar](#) control.

```
Public Sub ToolBarConfig()
    ' Instantiate the toolbar buttons, set their Text properties
    ' and add them to the ToolBar control.
    ToolBar1.Buttons.Add(New ToolBarButton("One"))
    ToolBar1.Buttons.Add(New ToolBarButton("Two"))
    ToolBar1.Buttons.Add(New ToolBarButton("Three"))
    ' Add the event handler delegate.
    AddHandler ToolBar1.ButtonClick, AddressOf Me.ToolBar1_ButtonClick
End Sub
```

```
public void ToolBarConfig()
{
    toolBar1.Buttons.Add(new ToolBarButton("One"));
    toolBar1.Buttons.Add(new ToolBarButton("Two"));
    toolBar1.Buttons.Add(new ToolBarButton("Three"));

    toolBar1.ButtonClick +=
        new ToolBarButtonClickEventHandler(this.toolBar1_ButtonClick);
}
```

```
public:
void ToolBarConfig()
{
    toolBar1->Buttons->Add(gcnew ToolBarButton("One"));
    toolBar1->Buttons->Add(gcnew ToolBarButton("Two"));
    toolBar1->Buttons->Add(gcnew ToolBarButton("Three"));

    toolBar1->ButtonClick +=
        gcnew ToolBarButtonClickEventHandler(this,
        &Form1::toolBar1_ButtonClick);
}
```

2. Add an event handler for the [ToolBar](#) control's [ButtonClick](#) event. Use a case switching statement and the

[ToolBarButtonClickEventArgs](#) class to determine the toolbar button that was clicked. Based on this, show an appropriate message box.

NOTE

A message box is being used solely as a placeholder in this example. Feel free to add other code to execute when the toolbar buttons are clicked.

```
Protected Sub ToolBar1_ButtonClick(ByVal sender As Object, _
ByVal e As ToolBarButtonClickEventArgs)
    ' Evaluate the Button property of the ToolBarButtonClickEventArgs
    ' to determine which button was clicked.
    Select Case ToolBar1.Buttons.IndexOf(e.Button)
        Case 0
            MessageBox.Show("First toolbar button clicked")
        Case 1
            MessageBox.Show("Second toolbar button clicked")
        Case 2
            MessageBox.Show("Third toolbar button clicked")
    End Select
End Sub
```

```
protected void toolBar1_ButtonClick(object sender,
ToolBarButtonClickEventArgs e)
{
    // Evaluate the Button property of the ToolBarButtonClickEventArgs
    // to determine which button was clicked.
    switch (toolBar1.Buttons.IndexOf(e.Button))
    {
        case 0 :
            MessageBox.Show("First toolbar button clicked");
            break;
        case 1 :
            MessageBox.Show("Second toolbar button clicked");
            break;
        case 2 :
            MessageBox.Show("Third toolbar button clicked");
            break;
    }
}
```

```
protected:
void toolBar1_ButtonClick(System::Object ^ sender,
    ToolBarButtonClickEventArgs ^ e)
{
    // Evaluate the Button property of the ToolBarButtonClickEventArgs
    // to determine which button was clicked.
    switch (toolBar1->Buttons->IndexOf(e->Button))
    {
        case 0 :
            MessageBox::Show("First toolbar button clicked");
            break;
        case 1 :
            MessageBox::Show("Second toolbar button clicked");
            break;
        case 2 :
            MessageBox::Show("Third toolbar button clicked");
            break;
    }
}
```

See Also

[ToolBar](#)

[How to: Add Buttons to a ToolBar Control](#)

[How to: Define an Icon for a ToolBar Button](#)

[ToolBar Control](#)

ToolStrip Control (Windows Forms)

5/4/2018 • 2 min to read • [Edit Online](#)

[ToolStrip](#) controls are toolbars that can host menus, controls, and user controls in your Windows Forms applications.

In This Section

[ToolStrip Control Overview](#)

Explains what this control is and its key features and properties.

[How to: Manage ToolStrip Overflow](#)

Demonstrates how to enable overflow functionality and control overflow behavior.

[How to: Enable Reordering of ToolStrip Items at Run Time](#)

Demonstrates how to enable run-time rearrangement of [ToolStripItem](#) controls.

[How to: Enable AutoComplete in ToolStrip Controls](#)

Demonstrates automatic completion functionality in a [ToolStripComboBox](#).

[How to: Change the Spacing and Alignment of ToolStrip Items in Windows Forms](#)

Describes various ways to arrange [ToolStripItems](#) on the [ToolStrip](#).

[How to: Change the Appearance of ToolStrip Text and Images in Windows Forms](#)

Describes how to define and modify the arrangement of text and images on [ToolStripItem](#) controls.

[How to: Wrap a Windows Forms Control with ToolStripControlHost](#)

Demonstrates hosting a [MonthCalendar](#) control in a [ToolStripControlHost](#).

[How to: Create and Set a Custom Renderer for the ToolStrip Control in Windows Forms](#)

Describes how to achieve standard and customized painting in [ToolStrip](#) controls.

[How to: Custom Draw a ToolStrip Control](#)

Describes various aspects of custom (owner) drawing in [ToolStrip](#) controls.

[How to: Create Toggle Buttons in ToolStrip Controls](#)

Describes the creation of a toggling [ToolStripButton](#).

[How to: Use ToolTips in ToolStrip Controls](#)

Describes defining tool tips for [ToolStrip](#) items.

[How to: Move a ToolStrip Out of a ToolStripContainer onto a Form](#)

Describes how to have a [ToolStrip](#) with no container but a form.

[How to: Position a ToolStripItem on a ToolStrip](#)

Describes how to position a [ToolStripButton](#) on at the leftmost or rightmost end of a [ToolStrip](#).

[How to: Enable the TAB Key to Move Out of a ToolStrip Control](#)

Describes how to tab within a [ToolStrip](#) and tab out of it to the next control in the tab order.

[How to: Detect When the Mouse Pointer Is Over a ToolStripItem](#)

Describes how to set up [ToolStrip](#) items so that they detect when the mouse is over them without having to synchronize various mouse events.

[How to: Create an MDI Form with ToolStripPanel Controls](#)

Describes how to create a multiple document interface (MDI) form with [ToolStripPanel](#) controls.

[How to: Create an MDI Form with Menu Merging and ToolStrip Controls](#)

Describes how to create an MDI form that supports [ToolStrip](#) controls and menu merging.

[How to: Create a Professionally Styled ToolStrip Control](#)

Describes how to use the [ToolStripProfessionalRenderer](#) class to create a composite control that mimics the **Navigation Pane** provided by Microsoft® Outlook®.

[How to: Implement a Custom ToolStripRenderer](#)

Describes how to customize the appearance of a [ToolStrip](#) control by implementing a class that derives from [ToolStripRenderer](#).

- [How to: Create a Basic Windows Forms ToolStrip with Standard Items Using the Designer](#)
- [How to: Move a ToolStrip Out of a ToolStripContainer onto a Form](#)
- [Walkthrough: Creating a Professionally Styled ToolStrip Control](#)
- [Walkthrough: Creating an MDI Form with Menu Merging and ToolStrip Controls](#)
- [ToolStrip Tasks Dialog Box](#)
- [ToolStrip Items Collection Editor](#)

Reference

[ToolStrip](#) class

Describes this class and has links to all its members.

[ToolStrip](#)

Describes the [ToolStrip](#) class and has links to all its members.

[ToolStripItem](#)

Describes the [ToolStripItem](#) class and has links to all its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

ToolStrip Control Overview (Windows Forms)

5/4/2018 • 2 min to read • [Edit Online](#)

The Windows Forms [ToolStrip](#) control and its associated classes provide a common framework for combining user interface elements into toolbars, status bars, and menus. [ToolStrip](#) controls offer a rich design-time experience that includes in-place activation and editing, custom layout, and rafting, which is the ability of toolbars to share horizontal or vertical space.

Although [ToolStrip](#) replaces and adds functionality to the control in previous versions, [ToolBar](#) is retained for both backward compatibility and future use if desired.

Features of the ToolStrip Controls

Use the [ToolStrip](#) control to:

- Present a common user interface across containers.
- Create easily customized, commonly employed toolbars that support advanced user interface and layout features, such as docking, rafting, buttons with text and images, drop-down buttons and controls, overflow buttons, and run-time reordering of [ToolStrip](#) items.
- Support overflow and run-time item reordering. The overflow feature moves items to a drop-down menu when there is not enough room to display them in a [ToolStrip](#).
- Support the typical appearance and behavior of the operating system through a common rendering model.
- Handle events consistently for all containers and contained items, in the same way you handle events for other controls.
- Drag items from one [ToolStrip](#) to another or within a [ToolStrip](#).
- Create drop-down controls and user interface type editors with advanced layouts in a [ToolStripDropDown](#).

Use the [ToolStripControlHost](#) class to use other controls on a [ToolStrip](#) and gain [ToolStrip](#) functionality for them.

You can extend the functionality and modify the appearance and behavior by using the [ToolStripRenderer](#), [ToolStripProfessionalRenderer](#), and [ToolStripManager](#) along with the [ToolStripRenderMode](#) and [ToolStripManagerRenderMode](#) enumerations.

The [ToolStrip](#) control is highly configurable and extensible, and it provides many properties, methods, and events to customize appearance and behavior. Below are some noteworthy members:

Important ToolStrip Members

NAME	DESCRIPTION
Dock	Gets or sets which edge of the parent container a ToolStrip is docked to.
AllowItemReorder	Gets or sets a value indicating whether drag-and-drop and item reordering are handled privately by the ToolStrip class.

NAME	DESCRIPTION
LayoutStyle	Gets or sets a value indicating how the ToolStrip lays out its items.
Overflow	Gets or sets whether a ToolStripItem is attached to the ToolStrip or ToolStripOverflowButton or can float between the two.
IsDropDown	Gets a value indicating whether a ToolStripItem displays other items in a drop-down list when the ToolStripItem is clicked.
OverflowButton	Gets the ToolStripItem that is the overflow button for a ToolStrip with overflow enabled.
Renderer	Gets or sets a ToolStripRenderer used to customize the appearance and behavior (look and feel) of a ToolStrip .
RenderMode	Gets or sets the painting styles to be applied to the ToolStrip .
RendererChanged	Raised when the Renderer property changes.

The [ToolStrip](#) control's flexibility is achieved through the use of a number of companion classes. Below are some of the most noteworthy:

Important ToolStrip Companion Classes

NAME	DESCRIPTION
MenuStrip	Replaces and adds functionality to the MainMenu class.
StatusStrip	Replaces and adds functionality to the StatusBar class.
ContextMenuStrip	Replaces and adds functionality to the ContextMenu class.
ToolStripItem	Abstract base class that manages events and layout for all the elements that a ToolStrip , ToolStripControlHost , or ToolStripDropDown can contain.
ToolStripContainer	Provides a container with a panel on each side of the form in which controls can be arranged in various ways.
ToolStripRenderer	Handles the painting functionality for ToolStrip objects.
ToolStripProfessionalRenderer	Provides Microsoft Office-style appearance.
ToolStripManager	Controls ToolStrip rendering and rafting, and the merging of MenuStrip , ToolStripDropDownMenu , and ToolStripMenuItem objects.
ToolStripManagerRenderMode	Specifies the painting style (custom, Windows XP, or Microsoft Office Professional) applied to multiple ToolStrip objects contained in a form.

NAME	DESCRIPTION
ToolStripRenderMode	Specifies the painting style (custom, Windows XP, or Microsoft Office Professional) applied to one ToolStrip object contained in a form.
ToolStripControlHost	Hosts other controls that are not specifically ToolStrip controls but for which you want ToolStrip functionality.
ToolStripItemPlacement	Specifies whether a ToolStripItem is to be laid out on the main ToolStrip , on the overflow ToolStrip , or neither.

For more information, see [ToolStrip Technology Summary](#) and [ToolStrip Control Architecture](#).

See Also

- [ToolStrip](#)
- [MenuStrip](#)
- [ContextMenuStrip](#)
- [StatusStrip](#)
- [ToolStripItem](#)
- [ToolStripDropDown](#)

ToolStrip Technology Summary

5/4/2018 • 4 min to read • [Edit Online](#)

This topic summarizes information about the `ToolStrip` control and the classes that support its use.

The `ToolStrip` control and its associated classes provide a complete solution for creating toolbars, status bars, and menus.

Namespaces

[System.Windows.Forms](#)

Background

With the `ToolStrip` control and its associated classes, you can create advanced toolbar functionality that has consistent and professional appearance and behavior. The `ToolStrip` control and classes offer the following improvements over previous controls:

- A more consistent event model.
- A more consistent design-time behavior that contains task lists and item collection editors.
- Custom rendering with `ToolStripManager` and `ToolStripRenderer`.
- Built-in rafting (sharing of horizontal or vertical space within the tool area when docked) with the `ToolStripContainer` and `ToolStripPanel`.
- Design-time and run-time reordering of items with the `AllowItemReorder` property.
- Relocation of items to an overflow menu with the `CanOverflow` property.
- Completely configurable control location with the `ToolStripContainer`, `ToolStripPanel`, and `ToolStripContentPanel`.
- Hosting of `ToolStrip`, traditional, or custom controls using `ToolStripControlHost`.
- Merging of `ToolStrip` controls using `ToolStripPanel`.

`ToolStrip` is the extensible base class for `MenuStrip`, `ContextMenuStrip`, and `StatusStrip`. These controls are `ToolStripItem` containers that inherit common behavior and event handling, extended so that each implementation deals with the behavior that is appropriate for it. Controls that derive from `ToolStripItem` are listed in the following table. The base `ToolStrip` class handles painting, user input, and drag-and-drop events for these controls.

The `ToolStrip`, `MenuStrip`, `ContextMenuStrip`, and `StatusStrip` controls replace the previous toolbar, menu, shortcut menu, and status bar controls, although those controls are retained for backward compatibility.

ToolStrip Classes at a Glance

The following table shows the ToolStrip classes grouped by technology area.

TECHNOLOGY AREA	CLASS
-----------------	-------

TECHNOLOGY AREA	CLASS
Toolbar, Status, and Menu containers	ToolStrip MenuStrip ContextMenuStrip StatusStrip ToolStripDropDownMenu
ToolStrip items	ToolStripLabel ToolStripDropDownItem ToolStripMenuItem ToolStripButton ToolStripStatusLabel ToolStripSeparator ToolStripControlHost ToolStripComboBox ToolStripTextBox ToolStripProgressBar ToolStripDropDownButton ToolStripSplitButton
Location	ToolStripContainer ToolStripContentPanel ToolStripPanel
Presentation and rendering	ToolStripManager ToolStripRenderer ToolStripProfessionalRenderer ToolStripRenderMode ToolStripManagerRenderMode

ToolStrip Design-Time Features

The [ToolStrip](#) family of controls provides a rich set of tools and templates for in-place editing and defining the foundation of the user interface so that you can quickly create a working application.

Task Dialog Boxes

In Visual Studio, clicking the smart tag on a control in the designer displays a task list for convenient access to many frequently used commands.

- [MenuStrip Tasks Dialog Box](#)
- [ToolStrip Tasks Dialog Box](#)
- [ContextMenuStrip Tasks Dialog Box](#)
- [StatusStrip Tasks Dialog Box](#)
- [ToolStripContainer Tasks Dialog Box](#)

Items Collection Editors

In Visual Studio, when you click **Edit Items** on the task list or right-click the control and select **Edit Items** in the shortcut menu, the collection editor for the control is displayed. Collection editors let you add, remove, and reorder items that the control contains. You can also view and change the properties for the control and the control's items.

- [MenuStrip Items Collection Editor](#)
- [StatusStrip Items Collection Editor](#)
- [ContextMenuStrip Items Collection Editor](#)
- [ToolStrip Items Collection Editor](#)

Hosting Controls

The [ToolStripControlHost](#) class provides built-in wrappers for [ToolStripComboBox](#), [ToolStripTextBox](#), and [ToolStripProgressBar](#) controls. You can also host any other existing or COM control in a [ToolStripControlHost](#).

For an example of control hosting, see [How to: Wrap a Windows Forms Control with ToolStripControlHost](#).

Rendering

[ToolStrip](#) classes implement a rendering scheme that is significantly different from other Windows Forms controls. With this scheme, you can easily apply styles and themes.

To apply a style to a [ToolStrip](#) and all the [ToolStripItem](#) objects it contains, you do not have to handle the [Paint](#) event for each item. Instead, you can set the [RenderMode](#) property to one of the [ToolStripRenderMode](#) values other than [Custom](#). Alternatively, you can set the [Renderer](#) directly to any class that inherits from the [ToolStripRenderer](#) class. Setting this property automatically sets the [RenderMode](#).

You can apply the same style to multiple [ToolStrip](#) objects in the same application by setting the [RenderMode](#) to [ManagerRenderMode](#) and setting the [RenderMode](#) or [Renderer](#) property to [ToolStripManagerRenderMode](#) that you want or [ToolStripRenderer](#) value, respectively.

For examples of rendering, see [How to: Create and Set a Custom Renderer for the ToolStrip Control in Windows Forms](#).

Styles and Themes

[ToolStrip](#) and associated classes provide an easy way to support visual styles and custom appearance that do not require overriding the [OnPaint](#) methods for each item. Use the [DisplayStyle](#) and the [RenderMode](#) and [Renderer](#) properties.

Rafting and Docking

You can raft, dock, or absolutely position [ToolStrip](#) controls. [ToolStrip](#) items are laid out by the [LayoutEngine](#) of the container.

Rafting is the ability of toolbars to share horizontal or vertical space. A Windows form can have a [ToolStripContainer](#) that in turn has panels on the form's left, right, top, and bottom sides for positioning and rafting [ToolStrip](#), [MenuStrip](#), and [StatusStrip](#) controls. Multiple [ToolStrip](#) controls stack vertically if you put them in the left or right [ToolStripContainer](#). They stack horizontally if you put them in the top or bottom [ToolStripContainer](#). You can use the central [ToolStripContentPanel](#) of the [ToolStripContainer](#) to position traditional controls on the form.

Any or all [ToolStripContainer](#) controls are directly selectable at design time and can be deleted. A [ToolStripContainer](#) is expandable and collapsible, and resizes with the controls that it contains.

Docking is the specifying of a control's simple location on the form's left, right, top, or bottom side.

The advantage of rafting over docking is that [ToolStrip](#), [MenuStrip](#), and [StatusStrip](#) controls can share horizontal or vertical space with other controls.

Most of the [ToolStrip](#) controls can be docked to the form like other controls instead of using rafting. You can also specify that a [ToolStrip](#) control be freely positioned on the form by removing it from its [ToolStripContainer](#) and setting its `Dock` property to `None`, or you can specify its absolute position by setting the respective [Location](#) property. See [How to: Move a ToolStrip Out of a ToolStripContainer onto a Form](#).

Use one or more [ToolStripPanel](#) controls for more flexibility, especially for Multiple Document Interface (MDI) applications, or if you do not need a [ToolStripContainer](#). A [ToolStripPanel](#) provides a dockable space for locating and rafting [ToolStrip](#) controls but not traditional controls. By default, the [ToolStripPanel](#) does not appear in the designer [Toolbox](#), but you can put it there by right-clicking the [Toolbox](#), and then click **Choose Items**. You can also programmatically access the [ToolStripPanel](#) like any other class.

The [ToolStrip](#), [MenuStrip](#), and [StatusStrip](#) let items overflow. This is similar to the way these items behave on Microsoft Office toolbars.

See Also

[ToolStrip Control Overview](#)

[ToolStrip Control Architecture](#)

ToolStrip Control Architecture

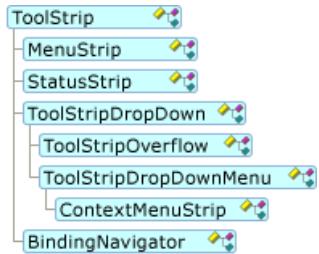
5/4/2018 • 14 min to read • [Edit Online](#)

The [ToolStrip](#) and [ToolStripItem](#) classes provide a flexible, extensible system for displaying toolbar, status, and menu items. These classes are all contained in the [System.Windows.Forms](#) namespace and they are all typically named with the "ToolStrip" prefix (such as [ToolStripOverflow](#)) or with the "Strip" suffix (such as [MenuStrip](#)).

ToolStrip

The following topics describe [ToolStrip](#) and the controls that derive from it.

[ToolStrip](#) is the abstract base class for [MenuStrip](#), [StatusStrip](#), and [ContextMenuStrip](#). The following object model shows the [ToolStrip](#) inheritance hierarchy.



ToolStrip object model

You can access all the items in a [ToolStrip](#) through the [Items](#) collection. You can access all the items in a [ToolStripDropDownItem](#) through the [DropDownItems](#) collection. In a class derived from [ToolStrip](#), you can also use the [DisplayedItems](#) property to access only those items that are currently displayed. These are the items that are not currently in an overflow menu.

The following items are specifically designed to work seamlessly with both [ToolStripSystemRenderer](#) and [ToolStripProfessionalRenderer](#) in all orientations. They are available by default at design time for the [ToolStrip](#) control:

- [ToolStripButton](#)
- [ToolStripSeparator](#)
- [ToolStripLabel](#)
- [ToolStripDropDownButton](#)
- [ToolStripSplitButton](#)
- [ToolStripTextBox](#)
- [ToolStripComboBox](#)

MenuStrip

[MenuStrip](#) is the top-level container that supersedes [MainMenu](#). It also provides key handling and multiple document interface (MDI) features. Functionally, [ToolStripDropDownItem](#) and [ToolStripMenuItem](#) work along with [MenuStrip](#), although they are derived from [ToolStripItem](#).

The following items are specifically designed to work seamlessly with both [ToolStripSystemRenderer](#) and [ToolStripProfessionalRenderer](#) in all orientations. They are available by default at design time for the [MenuStrip](#) control:

- [ToolStripMenuItem](#)
- [ToolStripTextBox](#)
- [ToolStripComboBox](#)

MenuStrip

[StatusStrip](#) replaces the [StatusBar](#) control. Special features of [StatusStrip](#) include a custom table layout, support for the form's sizing and moving grips, and the `Spring` property, which allows a [ToolStripStatusLabel](#) to fill available space automatically.

The following items are specifically designed to work seamlessly with both [ToolStripSystemRenderer](#) and [ToolStripProfessionalRenderer](#) in all orientations. They are available by default at design time for the [StatusStrip](#) control:

- [ToolStripStatusLabel](#)
- [ToolStripDropDownButton](#)
- [ToolStripSplitButton](#)
- [ToolStripProgressBar](#)

ContextMenuStrip

[ContextMenuStrip](#) replaces [ContextMenu](#). You can associate a [ContextMenuStrip](#) with any control, and a right mouse click automatically displays the context menu (or shortcut menu). You can show a [ContextMenuStrip](#) programmatically by using the [Show](#) method. [ContextMenuStrip](#) supports cancelable [Opening](#) and [Closing](#) events to handle dynamic population and multiple-click scenarios. [ContextMenuStrip](#) supports images, menu-item check state, text, access keys, shortcuts, and cascading menus.

The following items are specifically designed to work seamlessly with both [ToolStripSystemRenderer](#) and [ToolStripProfessionalRenderer](#) in all orientations. They are available by default at design time for the [ContextMenuStrip](#) control:

- [ToolStripMenuItem](#)
- [ToolStripSeparator](#)
- [ToolStripTextBox](#)
- [ToolStripComboBox](#)

ToolStrip Generic Features

The following topics describe features and behavior that are generic to the [ToolStrip](#) and derived controls.

Painting

You can do custom painting in [ToolStrip](#) controls in several ways. As with other Windows Forms controls, the [ToolStrip](#) and [ToolStripItem](#) both have overridable `OnPaint` methods and `Paint` events. As with regular painting, the coordinate system is relative to the client area of the control; that is, the upper left-hand corner of the control is 0, 0. The `Paint` event and `OnPaint` method for a [ToolStripItem](#) behave like other control paint events.

The [ToolStrip](#) controls also provide finer access to the rendering of the items and container through the [ToolStripRenderer](#) class, which has overridable methods for painting the background, item background, item image, item arrow, item text, and border of the [ToolStrip](#). The event arguments for these methods expose several properties such as rectangles, colors, and text formats that you can adjust as desired.

To adjust just a few aspects of how an item is painted, you typically override the [ToolStripRenderer](#).

If you are writing a new item and want to control all aspects of the painting, override the `OnPaint` method. From within `OnPaint`, you can use methods from the [ToolStripRenderer](#).

By default, the [ToolStrip](#) is double buffered, taking advantage of the [OptimizedDoubleBuffer](#) setting.

Parenting

The concept of container ownership and parenting is more complex in [ToolStrip](#) controls than in other Windows Forms container controls. That is necessary to support dynamic scenarios such as overflow, sharing drop-down items across multiple [ToolStrip](#) items, and to support the generation of a [ContextMenuStrip](#) from a control.

The following list describes members related to parenting and explains their use.

- [OwnerItem](#) accesses the item that is the source of the drop-down item. This is similar to [SourceControl](#), but instead of returning a control, it returns a [ToolStripItem](#).
- [SourceControl](#) determines which control is the source of the [ContextMenuStrip](#) when multiple controls share the same [ContextMenuStrip](#).
- [GetCurrentParent](#) is a read-only accessor to the [Parent](#) property. A parent differs from an owner in that a parent denotes the returned current [ToolStrip](#) in which the item is displayed, which might be in the overflow area.
- [Owner](#) returns the [ToolStrip](#) whose Items collection contains the current [ToolStripItem](#). This is the best way to reference [ImageList](#) or other properties in the top-level [ToolStrip](#) without writing special code to handle overflow.

Behavior of Inherited Controls

The following controls are locked whenever they are used in inheritance:

- [ToolStrip](#)
- [MenuStrip](#)
- [ContextMenuStrip](#)
- [StatusStrip](#)
- [ToolStripPanel](#) that includes the panels in a [ToolStripContainer](#) and also individual [ToolStripPanel](#) controls.

For example, create a new Windows Forms application by using one or more of the controls in the previous list. Set the access modifier of one or more controls to `public` or `protected`, and then build the project. Add a form that inherits from the first form, and then select an inherited control. The control appears locked, behaving as if its access modifier was `private`.

ToolStripContainer Support of Inheritance

The [ToolStripContainer](#) control supports limited inherited scenarios, similar to the following example:

1. Create a new Windows Forms application.
2. Add a [ToolStripContainer](#) to the form.
3. Set the access modifier of the [ToolStripContainer](#) to `public` or `protected`.
4. Add any combination of [ToolStrip](#), [MenuStrip](#), and [ContextMenuStrip](#) controls to the [ToolStripPanel](#) regions of the [ToolStripContainer](#).
5. Build the project.
6. Add a form that inherits from the first form.
7. Select the inherited [ToolStripContainer](#) on the form.

Inherited Behavior of Child Controls

After you complete the previous steps, the following inherited behavior occurs:

- In the designer, the control appears with an inherited icon.
- The [ToolStripPanel](#) controls are locked; you cannot select or rearrange their contents.
- You can add controls to the [ToolStripContentPanel](#), move the controls, and make them child controls of the [ToolStripContentPanel](#).
- Your changes persist after building the form.

NOTE

Remove the access modifiers from all [ToolStripPanel](#) controls that are part of a [ToolStripContainer](#). The access modifier of the [ToolStripContainer](#) governs the whole control.

Partial Trust

The limitations of [ToolStrip](#)s under partial trust are designed to prevent inadvertent entry of personal information that might be used by unauthorized persons or services. The protective measures are as follows:

- [ToolStripDropDown](#) controls require [AllWindows](#) to display items in a [ToolStripControlHost](#). This applies to both intrinsic controls such as [ToolStripTextBox](#), [ToolStripComboBox](#), and [ToolStripProgressBar](#) as well as to user-created controls. If this requirement is not met, these items are not displayed. No exception is thrown.
- Setting the [AutoClose](#) property to `false` is not allowed, and the cancelable [Closing](#) event parameter is ignored. This makes it impossible to enter more than one keystroke without dismissing the drop-down item. If this requirement is not met, such items are not displayed. No exception is thrown.
- Many keystroke handling events will not be raised if they occur in partial trust contexts other than [AllWindows](#).
- Access keys are not processed when [AllWindows](#) is not granted.

Usage

The following usage patterns have a bearing on [ToolStrip](#) layout, keyboard interaction, and end-user behavior:

- Joined in a [ToolStripPanel](#)

The [ToolStrip](#) can be repositioned within the [ToolStripPanel](#) and across [ToolStripPanels](#). The [Dock](#) property is ignored, and if the [Stretch](#) property is `false`, the size of the [ToolStrip](#) grows as items are added to the [ToolStripPanel](#). Typically, the [ToolStrip](#) does not participate in the tab order.

- Docked

The [ToolStrip](#) is placed on one side of a container in a fixed position, and its size expands over the entire edge to which it is docked. Typically, the [ToolStrip](#) does not participate in the tab order.

- Absolutely positioned

The [ToolStrip](#) is like other controls, in that it is placed by the [Location](#) property, has a fixed size, and typically participates in the tab order.

Keyboard Interaction

Access Keys

Combined with or following the ALT key, access keys are one way to activate a control using the keyboard. [ToolStrip](#) supports both explicit and implicit access keys. Explicit definition uses an ampersand (&) character preceding the letter. Implicit definition uses an algorithm that attempts to find a matching item based on the order of characters in a given [Text](#) property.

Shortcut Keys

The shortcut keys used by a [MenuStrip](#) use a combination of the [Keys](#) enumeration (which is not order-specific)

to define the shortcut key. You can also use the [ShortcutKeyDisplayString](#) property to display a shortcut key with text only, such as displaying "Del" instead of "Delete."

Navigation

The ALT key activates the [MenuStrip](#) pointed to by [MainMenuStrip](#). From there, CTRL+TAB navigates between [ToolStrip](#) controls within [ToolStripPanel](#)s. The TAB key and the arrow keys on the numeric keypad navigate between items in a [ToolStrip](#). A special algorithm handles navigation in the overflow region. SPACEBAR selects a [ToolStripButton](#), [ToolStripDropDownButton](#), or [ToolStripSplitButton](#).

Focus and Validation

When activated by the ALT key, the [MenuStrip](#) or [ToolStrip](#) typically neither take nor remove the focus from the control that currently has the focus. If there is a control hosted within the [MenuStrip](#) or a drop-down of the [MenuStrip](#), the control gains focus when the user presses the TAB key. In general, the [GotFocus](#), [LostFocus](#), [Enter](#), and [Leave](#) events of [MenuStrip](#) might not be raised when they are activated by the keyboard. In such cases, use the [MenuActivate](#) and [MenuDeactivate](#) events instead.

By default, [CausesValidation](#) is `false`. Call [Validate](#) explicitly on your form to perform validation.

Layout

You control [ToolStrip](#) layout by choosing one of the members of [ToolStripLayoutStyle](#) with the [LayoutStyle](#) property.

Stack Layouts

Stacking is the arranging of items beside each other at both ends of the [ToolStrip](#). The following list describes the stack layouts.

- [StackWithOverflow](#) is the default. This setting causes the [ToolStrip](#) to alter its layout automatically in accordance with the [Orientation](#) property to handle dragging and docking scenarios.
- [VerticalStackWithOverflow](#) renders the [ToolStrip](#) items beside each other vertically.
- [HorizontalStackWithOverflow](#) renders the [ToolStrip](#) items beside each other horizontally.

Other Features of Stack Layouts

[Alignment](#) determines the end of the [ToolStrip](#) to which the item is aligned.

When items do not fit within the [ToolStrip](#), an overflow button automatically appears. The [Overflow](#) property setting determines whether an item appears in the overflow area always, as needed, or never.

In the [LayoutCompleted](#) event, you can inspect the [Placement](#) property to determine whether an item was placed on the main [ToolStrip](#), the overflow [ToolStrip](#), or if it is not currently showing at all. The typical reasons why an item is not displayed are that the item did not fit on the main [ToolStrip](#) and its [Overflow](#) property was set to [Never](#).

Make a [ToolStrip](#) movable by putting it in a [ToolStripPanel](#) and setting its [GripStyle](#) to [Visible](#).

Other Layout Options

The other layout options are [Flow](#) and [Table](#).

Flow Layout

[Flow](#) layout is the default for [ContextMenuStrip](#), [ToolStripDropDownMenu](#), and [ToolStripOverflow](#). It is similar to the [FlowLayoutPanel](#). The features of [Flow](#) layout are as follows:

- All of the features of [FlowLayoutPanel](#) are exposed by the [LayoutSettings](#) property. You must cast the [LayoutSettings](#) class to a [FlowLayoutSettings](#) class.
- You can use the [Dock](#) and [Anchor](#) properties in code to align the items within the row.
- The [Alignment](#) property is ignored.
- In the [LayoutCompleted](#) event, you can inspect the [Placement](#) property to determine whether an item was placed on the main [ToolStrip](#) or did not fit.

- The grip is not rendered, and therefore a [ToolStrip](#) in [Flow](#) layout style in a [ToolStripPanel](#) cannot be moved.
- The [ToolStrip](#) overflow button is not rendered, and [Overflow](#) is ignored.

Table Layout

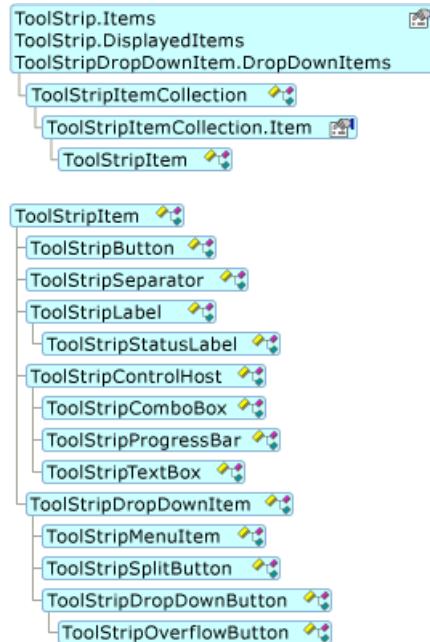
[Table](#) layout is the default for [StatusStrip](#). It is similar to [TableLayoutPanel](#). The features of [Flow](#) layout are as follows:

- All of the features of [TableLayoutPanel](#) are exposed by the [LayoutSettings](#) property. You must cast the [LayoutSettings](#) class to a [TableLayoutPanelSettings](#) class.
- You can use the [Dock](#) and [Anchor](#) properties in code to align the items within the table cell.
- The [Alignment](#) property is ignored.
- In the [LayoutCompleted](#) event, you can inspect the [Placement](#) property to determine whether an item was placed on the main [ToolStrip](#) or did not fit.
- The grip is not rendered, and therefore a [ToolStrip](#) in [Table](#) layout style in a [ToolStripPanel](#) cannot be moved.
- The [ToolStrip](#) overflow button is not rendered, and [Overflow](#) is ignored.

ToolStripItem

The following topics describe [ToolStripItem](#) and the controls that derive from it.

[ToolStripItem](#) is the abstract base class for all the items that go into a [ToolStrip](#). The following object model shows the [ToolStripItem](#) inheritance hierarchy.



ToolStripItem object model

[ToolStripItem](#) classes either inherit directly from [ToolStripItem](#), or they inherit indirectly from [ToolStripItem](#) through [ToolStripControlHost](#) or [ToolStripDropDownItem](#).

[ToolStripItem](#) controls must be contained in a [ToolStrip](#), [MenuStrip](#), [StatusStrip](#), or [ContextMenuStrip](#) and cannot be added directly to a form. The various container classes are designed to contain an appropriate subset of [ToolStripItem](#) controls.

The following table lists the stock [ToolStripItem](#) controls and the containers in which they look best. Although any [ToolStrip](#) item can be hosted in any [ToolStrip](#)-derived container, these items were designed to look best in the

following containers:

NOTE

[ToolStripDropDown](#) does not appear in the designer toolbox.

CONTAINED ITEM	TOOLSTRIP	MENUSTrip	CONTEXTMENUSTrip	STATUSSTRIP	TOOLSTRIPDROPDOWN
ToolStripButton	Yes	No	No	No	Yes
ToolStripComboBox	Yes	Yes	Yes	No	Yes
ToolStripSplitButton	Yes	No	No	Yes	Yes
ToolStripLabel	Yes	No	No	Yes	Yes
ToolStripSeparator	Yes	Yes	Yes	No	Yes
ToolStripDropDownButton	Yes	No	No	Yes	Yes
ToolStripTextBox	Yes	Yes	Yes	No	Yes
ToolStripMenultem	No	Yes	Yes	No	No
ToolStripStatusLabel	No	No	No	Yes	No
ToolStripProgressBar	Yes	No	No	Yes	No
ToolStripControlHost	Yes	Yes	No	Yes	Yes

ToolStripButton

[ToolStripButton](#) is the button item for [ToolStrip](#). You can display it with various border styles, and you can use it to represent and activate operational states. You can also define it to have the focus by default.

ToolStripLabel

The [ToolStripLabel](#) provides label functionality in [ToolStrip](#) controls. The [ToolStripLabel](#) is like a [ToolStripButton](#) that does not get focus by default and that does not render as pushed or highlighted.

[ToolStripLabel](#) as a hosted item supports access keys.

Use the [LinkColor](#), [LinkVisited](#), and [LinkBehavior](#) properties on a [ToolStripLabel](#) to support link control in a [ToolStrip](#).

ToolStripStatusLabel

[ToolStripStatusLabel](#) is a version of [ToolStripLabel](#) designed specifically for use in [StatusStrip](#). The special features include [BorderStyle](#), [BorderSides](#), and [Spring](#).

ToolStripSeparator

The [ToolStripSeparator](#) adds a vertical or horizontal line to a toolbar or menu, depending on the orientation. It provides grouping of or distinction between items, such as those on a menu.

You can add a [ToolStripSeparator](#) at design time by choosing it from a drop-down list. However, you can also automatically create a [ToolStripSeparator](#) by typing a hyphen (-) in either the designer template node or in the [Add](#) method.

ToolStripControlHost

[ToolStripControlHost](#) is the abstract base class for [ToolStripComboBox](#), [ToolStripTextBox](#), and [ToolStripProgressBar](#). [ToolStripControlHost](#) can host other controls, including custom controls, in two ways:

- Construct a [ToolStripControlHost](#) with a class that derives from [Control](#). To fully access the hosted control and properties, you must cast the [Control](#) property back to the actual class it represents.
- Extend [ToolStripControlHost](#), and in the inherited class's default constructor, call the base class constructor passing a class that derives from [Control](#). This option lets you wrap common control methods and properties for easy access in a [ToolStrip](#).

ToolStripComboBox

[ToolStripComboBox](#) is the [ComboBox](#) optimized for hosting in a [ToolStrip](#). A subset of the hosted control's properties and events are exposed at the [ToolStripComboBox](#) level, but the underlying [ComboBox](#) control is fully accessible through the [ComboBox](#) property.

ToolStripTextBox

[ToolStripTextBox](#) is the [TextBox](#) optimized for hosting in a [ToolStrip](#). A subset of the hosted control's properties and events are exposed at the [ToolStripTextBox](#) level, but the underlying [TextBox](#) control is fully accessible through the [TextBox](#) property.

ToolStripProgressBar

[ToolStripProgressBar](#) is the [ProgressBar](#) optimized for hosting in a [ToolStrip](#). A subset of the hosted control's properties and events are exposed at the [ToolStripProgressBar](#) level, but the underlying [ProgressBar](#) control is fully accessible through the [ProgressBar](#) property.

ToolStripDropDownItem

[ToolStripDropDownItem](#) is the abstract base class for [ToolStripMenuItem](#), [ToolStripDropDownButton](#), and [ToolStripSplitButton](#), which can host items directly or host additional items in a drop-down container. You do this by setting the [DropDown](#) property to a [ToolStripDropDown](#) and setting the [Items](#) property of the [ToolStripDropDown](#). Access these drop-down items directly through the [DropDownItems](#) property.

ToolStripMenuItem

[ToolStripMenuItem](#) is a [ToolStripDropDownItem](#) that works with [ToolStripDropDownMenu](#) and [ContextMenuStrip](#) to handle the special highlighting, layout, and column arrangement for menus.

ToolStripDropDownButton

[ToolStripDropDownButton](#) looks like [ToolStripButton](#), but it shows a drop-down area when the user clicks it. Hide or show the drop-down arrow by setting the [ShowDropDownArrow](#) property. [ToolStripDropDownButton](#) hosts a [ToolStripOverflowButton](#) that displays items that overflow the [ToolStrip](#).

ToolStripSplitButton

[ToolStripSplitButton](#) combines button and drop-down button functionality.

Use the [DefaultItem](#) property to synchronize the [Click](#) event of the chosen drop-down item with the item shown on the button.

ToolStripItem Generic Features

[ToolStripItem](#) provides the following generic features and options to inheriting controls:

- Core events
- Image handling
- Alignment
- Text and image relationship
- Display style

Core Events

[ToolStripItem](#) controls receive their own click, mouse, and paint events, and can perform some keyboard preprocessing also.

Image Handling

The [Image](#), [ImageAlign](#), [ImageIndex](#), [ImageKey](#), and [ImageScaling](#) properties pertain to various aspects of image handling. Use images in [ToolStrip](#) controls by setting these properties directly or by setting the run-time-only [ImageList](#) property.

Image scaling is determined by the interaction of properties in both [ToolStrip](#) and [ToolStripItem](#), as follows:

- [ImageScalingSize](#) is the scale of the final image as determined by the combination of the image's [ImageScaling](#) setting and the container's [AutoSize](#) setting.
 - If [AutoSize](#) is `true` (the default) and [ToolStripItemImageScaling](#) is [SizeToFit](#), no image scaling occurs, and the [ToolStrip](#) size is that of the largest item, or a prescribed minimum size.
 - If [AutoSize](#) is `false` and [ToolStripItemImageScaling](#) is [None](#), neither image nor [ToolStrip](#) scaling occurs.

Alignment

The value of the [Alignment](#) property determines the end of the [ToolStrip](#) at which an item appears. The [Alignment](#) property works only when the layout style of the [ToolStrip](#) is set to one of the stack overflow values.

Items are placed on the [ToolStrip](#) in the order in which the items appear in the Items collection. To programmatically change where an item is laid out, use the [Insert](#) method to move the item in the collection. This method moves the item but does not duplicate it.

Text and Image Relationship

The [TextImageRelation](#) property defines the relative placement of the image with respect to the text on a [ToolStripItem](#). Items that lack an image, text, or both are treated as special cases so that the [ToolStripItem](#) does not display a blank spot for the missing element or elements.

Display Style

[DisplayStyle](#) allows you to set the values of an item's Text and Image properties while displaying only what you want. This is typically used to change only the display style when showing the same item in a different context.

Accessory Classes

Classes that provide various other functionality include:

- [ToolStripManager](#) supports [ToolStrip](#)-related tasks for entire applications, such as merging, settings, and renderer options.
- [ToolStripRenderer](#) allows you to apply a particular style or theme to a [ToolStrip](#) easily.
- [ToolStripProfessionalRenderer](#) creates pens and brushes based on a replaceable color table ([ProfessionalColorTable](#)).

- [ToolStripSystemRenderer](#) applies system colors and a flat visual style to [ToolStrip](#) applications.
- [ToolStripContainer](#) is similar to [SplitContainer](#). It uses four docked side panels (instances of [ToolStripPanel](#)) and one central panel (an instance of [ToolStripContentPanel](#)) to create a typical arrangement. You cannot remove the side panels, but you can hide them. You can neither remove nor hide the central panel. You can arrange one or more [ToolStrip](#), [MenuStrip](#), or [StatusStrip](#) controls in the side panels, and you can use the central panel for other controls. The [ToolStripContentPanel](#) also provides a way to get renderer support into the body of your form for a consistent appearance. [ToolStripContainer](#) does not support multiple document interface (MDI).
- [ToolStripPanel](#) provides space for moving and arranging [ToolStrip](#) controls. You can use only one panel if you so choose, and [ToolStripPanel](#) works well in MDI scenarios.

See Also

[ToolStrip Control Overview](#)

[ToolStrip Technology Summary](#)

[ToolStrip Control](#)

[MenuStrip Control](#)

[StatusStrip Control](#)

[ContextMenuStrip Control](#)

[BindingNavigator Control](#)

How to: Add ToolStrip Items Dynamically

5/4/2018 • 4 min to read • [Edit Online](#)

You can dynamically populate the menu item collection of a [ToolStrip](#) control when the menu opens.

Example

The following code example demonstrates how to dynamically add items to a [ContextMenuStrip](#) control. The example also shows how to reuse the same [ContextMenuStrip](#) for three different controls on the form.

In the example, an [Opening](#) event handler populates the menu item collection. The [Opening](#) event handler examines the [ContextMenuStrip.SourceControl](#) and [ToolStripItem.OwnerItem](#) properties and adds a [ToolStripItem](#) describing the source control.

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Drawing;
```

```
Imports System
Imports System.Collections.Generic
Imports System.Windows.Forms
Imports System.Drawing
```

```
// This code example demonstrates how to handle the Opening event.
// It also demonstrates dynamic item addition and dynamic
// SourceControl determination with reuse.
class Form3 : Form
{
    // Declare the ContextMenuStrip control.
    private ContextMenuStrip fruitContextMenuStrip;

    public Form3()
    {
        // Create a new ContextMenuStrip control.
        fruitContextMenuStrip = new ContextMenuStrip();

        // Attach an event handler for the
        // ContextMenuStrip control's Opening event.
        fruitContextMenuStrip.Opening += new System.ComponentModel.CancelEventHandler(cms_Opening);

        // Create a new ToolStrip control.
        ToolStrip ts = new ToolStrip();

        // Create a ToolStripDropDownButton control and add it
        // to the ToolStrip control's Items collections.
        ToolStripDropDownButton fruitToolStripDropDownButton = new ToolStripDropDownButton("Fruit", null, null,
            "Fruit");
        ts.Items.Add(fruitToolStripDropDownButton);

        // Dock the ToolStrip control to the top of the form.
        ts.Dock = DockStyle.Top;

        // Assign the ContextMenuStrip control as the
        // ToolStripDropDownButton control's DropDown menu.
        fruitToolStripDropDownButton.DropDown = fruitContextMenuStrip;
```

```

// Create a new MenuStrip control and add a ToolStripMenuItem.
MenuStrip ms = new MenuStrip();
ToolStripMenuItem fruitToolStripMenuItem = new ToolStripMenuItem("Fruit", null, null, "Fruit");
ms.Items.Add(fruitToolStripMenuItem);

// Dock the MenuStrip control to the top of the form.
ms.Dock = DockStyle.Top;

// Assign the MenuStrip control as the
// ToolStripMenuItem's DropDown menu.
fruitToolStripMenuItem.DropDown = fruitContextMenuStrip;

// Assign the ContextMenuStrip to the form's
// ContextMenuStrip property.
this.ContextMenuStrip = fruitContextMenuStrip;

// Add the ToolStrip control to the Controls collection.
this.Controls.Add(ts);

//Add a button to the form and assign its ContextMenuStrip.
Button b = new Button();
b.Location = new System.Drawing.Point(60, 60);
this.Controls.Add(b);
b.ContextMenuStrip = fruitContextMenuStrip;

// Add the MenuStrip control last.
// This is important for correct placement in the z-order.
this.Controls.Add(ms);
}

// This event handler is invoked when the ContextMenuStrip
// control's Opening event is raised. It demonstrates
// dynamic item addition and dynamic SourceControl
// determination with reuse.
void cms_Opening(object sender, System.ComponentModel.CancelEventArgs e)
{
    // Acquire references to the owning control and item.
    Control c = fruitContextMenuStrip.SourceControl as Control;
    ToolStripDropDownItem tsi = fruitContextMenuStrip.OwnerItem as ToolStripDropDownItem;

    // Clear the ContextMenuStrip control's Items collection.
    fruitContextMenuStrip.Items.Clear();

    // Check the source control first.
    if (c != null)
    {
        // Add custom item (Form)
        fruitContextMenuStrip.Items.Add("Source: " + c.GetType().ToString());
    }
    else if (tsi != null)
    {
        // Add custom item (ToolStripDropDownButton or ToolStripMenuItem)
        fruitContextMenuStrip.Items.Add("Source: " + tsi.GetType().ToString());
    }

    // Populate the ContextMenuStrip control with its default items.
    fruitContextMenuStrip.Items.Add("-");
    fruitContextMenuStrip.Items.Add("Apples");
    fruitContextMenuStrip.Items.Add("Oranges");
    fruitContextMenuStrip.Items.Add("Pears");

    // Set Cancel to false.
    // It is optimized to true based on empty entry.
    e.Cancel = false;
}
}

```

```

' This code example demonstrates how to handle the Opening event.
' It also demonstrates dynamic item addition and dynamic
' SourceControl determination with reuse.
Class Form3
    Inherits Form

    ' Declare the ContextMenuStrip control.
    Private fruitContextMenuStrip As ContextMenuStrip

    Public Sub New()
        ' Create a new ContextMenuStrip control.
        fruitContextMenuStrip = New ContextMenuStrip()

        ' Attach an event handler for the
        ' ContextMenuStrip control's Opening event.
        AddHandler fruitContextMenuStrip.Opening, AddressOf cms_Opening

        ' Create a new ToolStrip control.
        Dim ts As New ToolStrip()

        ' Create a ToolStripDropDownButton control and add it
        ' to the ToolStrip control's Items collections.
        Dim fruitToolStripDropDownButton As New ToolStripDropDownButton("Fruit", Nothing, Nothing, "Fruit")
        ts.Items.Add(fruitToolStripDropDownButton)

        ' Dock the ToolStrip control to the top of the form.
        ts.Dock = DockStyle.Top

        ' Assign the ContextMenuStrip control as the
        ' ToolStripDropDownButton control's DropDown menu.
        fruitToolStripDropDownButton.DropDown = fruitContextMenuStrip

        ' Create a new MenuStrip control and add a ToolStripMenuItem.
        Dim ms As New MenuStrip()
        Dim fruitToolStripMenuItem As New ToolStripMenuItem("Fruit", Nothing, Nothing, "Fruit")
        ms.Items.Add(fruitToolStripMenuItem)

        ' Dock the MenuStrip control to the top of the form.
        ms.Dock = DockStyle.Top

        ' Assign the MenuStrip control as the
        ' ToolStripMenuItem's DropDown menu.
        fruitToolStripMenuItem.DropDown = fruitContextMenuStrip

        ' Assign the ContextMenuStrip to the form's
        ' ContextMenuStrip property.
        Me.ContextMenuStrip = fruitContextMenuStrip

        ' Add the ToolStrip control to the Controls collection.
        Me.Controls.Add(ts)

        'Add a button to the form and assign its ContextMenuStrip.
        Dim b As New Button()
        b.Location = New System.Drawing.Point(60, 60)
        Me.Controls.Add(b)
        b.ContextMenuStrip = fruitContextMenuStrip

        ' Add the MenuStrip control last.
        ' This is important for correct placement in the z-order.
        Me.Controls.Add(ms)
    End Sub

    ' This event handler is invoked when the ContextMenuStrip
    ' control's Opening event is raised. It demonstrates
    ' dynamic item addition and dynamic SourceControl
    ' determination with reuse.
    Sub cms_Opening(ByVal sender As Object, ByVal e As System.ComponentModel.CancelEventArgs)

```

```

' Acquire references to the owning control and item.
Dim c As Control = fruitContextMenuStrip.SourceControl
Dim tsi As ToolStripDropDownItem = fruitContextMenuStrip.OwnerItem

' Clear the ContextMenuStrip control's
' Items collection.
fruitContextMenuStrip.Items.Clear()

' Check the source control first.
If (c IsNot Nothing) Then
    ' Add custom item (Form)
    fruitContextMenuStrip.Items.Add(("Source: " + c.GetType().ToString()))
ElseIf (tsi IsNot Nothing) Then
    ' Add custom item (ToolStripDropDownButton or ToolStripMenuItem)
    fruitContextMenuStrip.Items.Add(("Source: " + tsi.GetType().ToString()))
End If

' Populate the ContextMenuStrip control with its default items.
fruitContextMenuStrip.Items.Add("-")
fruitContextMenuStrip.Items.Add("Apples")
fruitContextMenuStrip.Items.Add("Oranges")
fruitContextMenuStrip.Items.Add("Pears")

' Set Cancel to false.
' It is optimized to true based on empty entry.
e.Cancel = False
End Sub
End Class

```

Compiling the Code

This example requires:

- References to the System.Drawing and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project.

See Also

[ContextMenuStrip](#)
[MenuStrip](#)
[ToolStrip](#)
[ToolStripItem](#)
[ToolStripMenuItem](#)
[ToolStripDropDownButton](#)
[ToolStrip Control](#)

How to: Create a Basic Windows Forms ToolStrip with Standard Items Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

The following procedure demonstrates how to create a [ToolStrip](#) and add seven [ToolStripButton](#) controls that represent typical tasks.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To add standard items in the designer

1. Create a [ToolStrip](#) control.
2. In the upper right corner of the [ToolStrip](#), click the smart task arrow to display the **ToolStrip Tasks** pane.
3. In the **ToolStrip Tasks** pane, choose **Insert Standard Items**.

See Also

[ToolStrip](#)

[ToolStrip Control Overview](#)

[ToolStrip Control](#)

[ToolStrip Control Architecture](#)

[ToolStrip Technology Summary](#)

How to: Create a Professionally Styled ToolStrip Control

5/4/2018 • 15 min to read • [Edit Online](#)

You can give your application's [ToolStrip](#) controls a professional appearance and behavior (look and feel) by writing your own class derived from the [ToolStripProfessionalRenderer](#) type.

There is extensive support for this feature in Visual Studio.

See [Walkthrough: Creating a Professionally Styled ToolStrip Control](#).

Example

The following code example demonstrates how to use [ToolStrip](#) controls to create a composite control that mimics the **Navigation Pane** provided by Microsoft® Outlook®.


```

    MemoryStream stream =
        new MemoryStream(Convert.FromBase64String(data));

    // Create a new bitmap from the stream.
    Bitmap b = new Bitmap(stream);

    return b;
}

// This method handles the Load event for the UserControl.
private void StackView_Load(object sender, EventArgs e)
{
    // Dock bottom.
    this.Dock = DockStyle.Bottom;

    // Set AutoSize.
    this.AutoSize = true;
}

// This method handles the Click event for all
// the ToolStripButton controls in the StackView.
private void stackButton_Click(object sender, EventArgs e)
{
    // Define a "one of many" state, similar to
    // the logic of a RadioButton control.
    foreach (ToolStripItem item in this.stackStrip.Items)
    {
        if ((item != sender) &&
            (item is ToolStripButton))
        {
            ((ToolStripButton)item).Checked = false;
        }
    }
}

protected override void Dispose(bool disposing)
{
    if (disposing && (components != null))
    {
        components.Dispose();
    }
    base.Dispose(disposing);
}

internal class StackRenderer : ToolStripProfessionalRenderer
{
    private static Bitmap titleBarGripBmp;
    private static string titleBarGripEnc =
@"Qk16AQAAAAAADYAAAoAAAAIwAAAAMAAAABgAAAAAAAAADEdGAxA4AAAAAAAAAUgMy+/n5+/n5uGMyuGMy+/n5+/n5uGMyuGMy+ /n5+/n5uGMyuGMy+/n5+/n5uGMyuGMy+/n5+/n5uGMyuGMy+/n5+/n5uGMyuGMy+/n5+/n5uGMyuGMy+/n5+/n5uGMyuGMy+/n5+/n5ANj+RzIo mHRh+/n5wm8/RzIomHRh+/n5wm8/RzIomHRh+/n5wm8/RzIomHRh+/n5wm8/RzIomHRh+/n5wm8/RzIomHRh+/n5wm8/RzI omHRh+/n5wm8/RzIomHRh+/n5ANj+RzIoRzIozHtMzHtMRzIoRzIozHtMzHtMRzIoRzIozHtMzHtMRzIoRzIozHtMzHtMRz IoRzIozHtMzHtMRzIoRzIozHtMzHtMRzIoRzIozHtMzHtMRzIoRzIozHtMANj+";

    // Define titlebar colors.
    private static Color titlebarColor1 = Color.FromArgb(89, 135, 214);
    private static Color titlebarColor2 = Color.FromArgb(76, 123, 204);
    private static Color titlebarColor3 = Color.FromArgb(63, 111, 194);
    private static Color titlebarColor4 = Color.FromArgb(50, 99, 184);
    private static Color titlebarColor5 = Color.FromArgb(38, 88, 174);
    private static Color titlebarColor6 = Color.FromArgb(25, 76, 164);
    private static Color titlebarColor7 = Color.FromArgb(12, 64, 154);
    private static Color borderColor = Color.FromArgb(0, 0, 128);

    static StackRenderer()
    {
        titleBarGripBmp = StackView.DeserializeFromBase64(titleBarGripEnc);
    }
}

```

```

public StackKenderer()
{
}

private void DrawTitleBar(Graphics g, Rectangle rect)
{
    // Assign the image for the grip.
    Image titlebarGrip = titleBarGripBmp;

    // Fill the titlebar.
    // This produces the gradient and the rounded-corner effect.
    g.DrawLine(new Pen(titlebarColor1), rect.X, rect.Y, rect.X + rect.Width, rect.Y);
    g.DrawLine(new Pen(titlebarColor2), rect.X, rect.Y + 1, rect.X + rect.Width, rect.Y + 1);
    g.DrawLine(new Pen(titlebarColor3), rect.X, rect.Y + 2, rect.X + rect.Width, rect.Y + 2);
    g.DrawLine(new Pen(titlebarColor4), rect.X, rect.Y + 3, rect.X + rect.Width, rect.Y + 3);
    g.DrawLine(new Pen(titlebarColor5), rect.X, rect.Y + 4, rect.X + rect.Width, rect.Y + 4);
    g.DrawLine(new Pen(titlebarColor6), rect.X, rect.Y + 5, rect.X + rect.Width, rect.Y + 5);
    g.DrawLine(new Pen(titlebarColor7), rect.X, rect.Y + 6, rect.X + rect.Width, rect.Y + 6);

    // Center the titlebar grip.
    g.DrawImage(
        titlebarGrip,
        new Point(rect.X + ((rect.Width / 2) - (titlebarGrip.Width / 2)),
        rect.Y + 1));
}

// This method handles the RenderGrip event.
protected override void OnRenderGrip(ToolStripGripRenderEventArgs e)
{
    DrawTitleBar(
        e.Graphics,
        new Rectangle(0, 0, e.ToolStrip.Width, 7));
}

// This method handles the RenderToolStripBorder event.
protected override void OnRenderToolStripBorder(ToolStripRenderEventArgs e)
{
    DrawTitleBar(
        e.Graphics,
        new Rectangle(0, 0, e.ToolStrip.Width, 7));
}

// This method handles the RenderButtonBackground event.
protected override void OnRenderButtonBackground(ToolStripItemRenderEventArgs e)
{
    Graphics g = e.Graphics;
    Rectangle bounds = new Rectangle(Point.Empty, e.Item.Size);

    Color gradientBegin = Color.FromArgb(203, 225, 252);
    Color gradientEnd = Color.FromArgb(125, 165, 224);

    ToolStripButton button = e.Item as ToolStripButton;
    if (button.Pressed || button.Checked)
    {
        gradientBegin = Color.FromArgb(254, 128, 62);
        gradientEnd = Color.FromArgb(255, 223, 154);
    }
    else if (button.Selected)
    {
        gradientBegin = Color.FromArgb(255, 255, 222);
        gradientEnd = Color.FromArgb(255, 203, 136);
    }

    using (Brush b = new LinearGradientBrush(
        bounds,
        gradientBegin,
        gradientEnd,
        LinearGradientMode.Vertical))
    {
}

```

```

        g.FillRectangle(b, bounds);
    }

    e.Graphics.DrawRectangle(
        SystemPens.ControlDarkDark,
        bounds);

    g.DrawLine(
        SystemPens.ControlDarkDark,
        bounds.X,
        bounds.Y,
        bounds.Width - 1,
        bounds.Y);

    g.DrawLine(
        SystemPens.ControlDarkDark,
        bounds.X,
        bounds.Y,
        bounds.X,
        bounds.Height - 1);

    ToolStrip toolStrip = button.Owner;
    ToolStripButton nextItem = button.Owner.GetItemAt(
        button.Bounds.X,
        button.Bounds.Bottom + 1) as ToolStripButton;

    if (nextItem == null)
    {
        g.DrawLine(
            SystemPens.ControlDarkDark,
            bounds.X,
            bounds.Height - 1,
            bounds.X + bounds.Width - 1,
            bounds.Height - 1);
    }
}
}

#endregion Component Designer generated code

private void InitializeComponent()
{
    this.stackStrip = new System.Windows.Forms.ToolStrip();
    this.mailStackButton = new System.Windows.Forms.ToolStripButton();
    this.calendarStackButton = new System.Windows.Forms.ToolStripButton();
    this.contactsStackButton = new System.Windows.Forms.ToolStripButton();
    this.tasksStackButton = new System.Windows.Forms.ToolStripButton();
    this.stackStrip.SuspendLayout();
    this.SuspendLayout();
    //
    // stackStrip
    //
    this.stackStrip.CanOverflow = false;
    this.stackStrip.Dock = System.Windows.Forms.DockStyle.Bottom;
    this.stackStrip.Font = new System.Drawing.Font("Tahoma", 10F, System.Drawing.FontStyle.Bold);
    this.stackStrip.GripStyle = System.Windows.Forms.ToolStripGripStyle.Hidden;
    this.stackStrip.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
    this.mailStackButton,
    this.calendarStackButton,
    this.contactsStackButton,
    this.tasksStackButton});
    this.stackStrip.LayoutStyle = System.Windows.Forms.ToolStripLayoutStyle.VerticalStackWithOverflow;
    this.stackStrip.Location = new System.Drawing.Point(0, 11);
    this.stackStrip.Name = "stackStrip";
    this.stackStrip.Padding = new System.Windows.Forms.Padding(0, 7, 0, 0);
    this.stackStrip.RenderMode = System.Windows.Forms.ToolStripRenderMode.Professional;
    this.stackStrip.Size = new System.Drawing.Size(150, 139);
    this.stackStrip.TabIndex = 0;
    this.stackStrip.Text = "toolStrip1";
}

```

```
//  
// mailStackButton  
//  
this.mailStackButton.CheckOnClick = true;  
this.mailStackButton.ImageAlign = System.Drawing.ContentAlignment.MiddleLeft;  
this.mailStackButton.ImageScaling = System.Windows.Forms.ToolStripItemImageScaling.None;  
this.mailStackButton.ImageTransparentColor = System.Drawing.Color.FromArgb(((int)((byte)(238))),  
((int)((byte)(238))), ((int)((byte)(238))));  
this.mailStackButton.Margin = new System.Windows.Forms.Padding(0);  
this.mailStackButton.Name = "mailStackButton";  
this.mailStackButton.Padding = new System.Windows.Forms.Padding(3);  
this.mailStackButton.Size = new System.Drawing.Size(149, 27);  
this.mailStackButton.Text = " Mail";  
this.mailStackButton.TextAlign = System.Drawing.ContentAlignment.MiddleLeft;  
this.mailStackButton.Click += new System.EventHandler(this.stackButton_Click);  
//  
// calendarStackButton  
//  
this.calendarStackButton.CheckOnClick = true;  
this.calendarStackButton.ImageAlign = System.Drawing.ContentAlignment.MiddleLeft;  
this.calendarStackButton.ImageScaling = System.Windows.Forms.ToolStripItemImageScaling.None;  
this.calendarStackButton.ImageTransparentColor = System.Drawing.Color.FromArgb(((int)((byte)  
(238))), ((int)((byte)(238))), ((int)((byte)(238))));  
this.calendarStackButton.Margin = new System.Windows.Forms.Padding(0);  
this.calendarStackButton.Name = "calendarStackButton";  
this.calendarStackButton.Padding = new System.Windows.Forms.Padding(3);  
this.calendarStackButton.Size = new System.Drawing.Size(149, 27);  
this.calendarStackButton.Text = " Calendar";  
this.calendarStackButton.TextAlign = System.Drawing.ContentAlignment.MiddleLeft;  
this.calendarStackButton.Click += new System.EventHandler(this.stackButton_Click);  
//  
// contactsStackButton  
//  
this.contactsStackButton.CheckOnClick = true;  
this.contactsStackButton.ImageAlign = System.Drawing.ContentAlignment.MiddleLeft;  
this.contactsStackButton.ImageScaling = System.Windows.Forms.ToolStripItemImageScaling.None;  
this.contactsStackButton.ImageTransparentColor = System.Drawing.Color.FromArgb(((int)((byte)  
(238))), ((int)((byte)(238))), ((int)((byte)(238))));  
this.contactsStackButton.Margin = new System.Windows.Forms.Padding(0);  
this.contactsStackButton.Name = "contactsStackButton";  
this.contactsStackButton.Padding = new System.Windows.Forms.Padding(3);  
this.contactsStackButton.Size = new System.Drawing.Size(149, 27);  
this.contactsStackButton.Text = " Contacts";  
this.contactsStackButton.TextAlign = System.Drawing.ContentAlignment.MiddleLeft;  
this.contactsStackButton.Click += new System.EventHandler(this.stackButton_Click);  
//  
// tasksStackButton  
//  
this.tasksStackButton.CheckOnClick = true;  
this.tasksStackButton.ImageAlign = System.Drawing.ContentAlignment.MiddleLeft;  
this.tasksStackButton.ImageScaling = System.Windows.Forms.ToolStripItemImageScaling.None;  
this.tasksStackButton.ImageTransparentColor = System.Drawing.Color.FromArgb(((int)((byte)(238))),  
((int)((byte)(238))), ((int)((byte)(238))));  
this.tasksStackButton.Margin = new System.Windows.Forms.Padding(0);  
this.tasksStackButton.Name = "tasksStackButton";  
this.tasksStackButton.Padding = new System.Windows.Forms.Padding(3);  
this.tasksStackButton.Size = new System.Drawing.Size(149, 27);  
this.tasksStackButton.Text = " Tasks";  
this.tasksStackButton.TextAlign = System.Drawing.ContentAlignment.MiddleLeft;  
this.tasksStackButton.Click += new System.EventHandler(this.stackButton_Click);  
//  
// StackView  
//  
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);  
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;  
this.Controls.Add(this.stackStrip);  
this.Name = "StackView";  
this.Load += new System.EventHandler(this.StackView_Load);  
this.stackStrip.ResumeLayout(false);
```

```
        this.stackStrip.PerformLayout();
        this.ResumeLayout(false);
        this.PerformLayout();

    }

#endregion

}

}
```



```

Protected Overrides Sub OnRenderGrip(e As ToolStripGripEventArgs)
    DrawTitleBar(e.Graphics, New Rectangle(0, 0, e.ToolStrip.Width, 7))
End Sub

' This method handles the RenderToolStripBorder event.
Protected Overrides Sub OnRenderToolStripBorder(e As ToolStripRenderEventArgs)
    DrawTitleBar(e.Graphics, New Rectangle(0, 0, e.ToolStrip.Width, 7))
End Sub

' This method handles the RenderButtonBackground event.
Protected Overrides Sub OnRenderButtonBackground(e As ToolStripItemRenderEventArgs)
    Dim g As Graphics = e.Graphics
    Dim bounds As New Rectangle(Point.Empty, e.Item.Size)

    Dim gradientBegin As Color = Color.FromArgb(203, 225, 252)
    Dim gradientEnd As Color = Color.FromArgb(125, 165, 224)

    Dim button As ToolStripButton = CType(e.Item, ToolStripButton)

    If button.Pressed OrElse button.Checked Then
        gradientBegin = Color.FromArgb(254, 128, 62)
        gradientEnd = Color.FromArgb(255, 223, 154)
    ElseIf button.Selected Then
        gradientBegin = Color.FromArgb(255, 255, 222)
        gradientEnd = Color.FromArgb(255, 203, 136)
    End If

    Dim b = New LinearGradientBrush(bounds, gradientBegin, gradientEnd, LinearGradientMode.Vertical)
    Try
        g.FillRectangle(b, bounds)
    Finally
        b.Dispose()
    End Try

    e.Graphics.DrawRectangle(SystemPens.ControlDarkDark, bounds)

    g.DrawLine(SystemPens.ControlDarkDark, bounds.X, bounds.Y, bounds.Width - 1, bounds.Y)

    g.DrawLine(SystemPens.ControlDarkDark, bounds.X, bounds.Y, bounds.X, bounds.Height - 1)

    Dim toolStrip As ToolStrip = button.Owner
    Dim nextItem As ToolStripButton = CType(button.Owner.GetItemAt(button.Bounds.X,
button.Bounds.Bottom + 1), ToolStripButton)

    If nextItem Is Nothing Then
        g.DrawLine(SystemPens.ControlDarkDark, bounds.X, bounds.Height - 1, bounds.X + bounds.Width - 1,
bounds.Height - 1)
    End If
    End Sub
End Class

#Region "Component Designer generated code"

Private Sub InitializeComponent()
    Me.stackStrip = New System.Windows.Forms.ToolStrip
    Me.mailStackButton = New System.Windows.Forms.ToolStripButton
    Me.calendarStackButton = New System.Windows.Forms.ToolStripButton
    Me.contactsStackButton = New System.Windows.Forms.ToolStripButton
    Me.tasksStackButton = New System.Windows.Forms.ToolStripButton
    Me.stackStrip.SuspendLayout()
    Me.SuspendLayout()
    '
    'stackStrip
    '
    Me.stackStrip.CanOverflow = False
    Me.stackStrip.Dock = System.Windows.Forms.DockStyle.Bottom

```

```

Me.stackStrip.Font = New System.Drawing.Font("Tahoma", 10.0!, System.Drawing.FontStyle.Bold)
Me.stackStrip.GripStyle = System.Windows.Forms.ToolStripGripStyle.Hidden
Me.stackStrip.Items.AddRange(New System.Windows.Forms.ToolStripItem() {Me.mailStackButton,
Me.calendarStackButton, Me.contactsStackButton, Me.tasksStackButton})
Me.stackStrip.LayoutStyle = System.Windows.Forms.ToolStripLayoutStyle.VerticalStackWithOverflow
Me.stackStrip.Location = New System.Drawing.Point(0, 11)
Me.stackStrip.Name = "stackStrip"
Me.stackStrip.Padding = New System.Windows.Forms.Padding(0, 7, 0, 0)
Me.stackStrip.RenderMode = System.Windows.Forms.ToolStripRenderMode.Professional
Me.stackStrip.Size = New System.Drawing.Size(150, 139)
Me.stackStrip.TabIndex = 0
Me.stackStrip.Text = "toolStrip1"
'

'mailStackButton
'

Me.mailStackButton.CheckOnClick = True
Me.mailStackButton.ImageAlign = System.Drawing.ContentAlignment.MiddleLeft
Me.mailStackButton.ImageScaling = System.Windows.Forms.ToolStripItemImageScaling.None
Me.mailStackButton.ImageTransparentColor = System.Drawing.Color.FromArgb(CType(CType(238, Byte),
Integer), CType(CType(238, Byte), Integer), CType(CType(238, Byte), Integer))
Me.mailStackButton.Margin = New System.Windows.Forms.Padding(0)
Me.mailStackButton.Name = "mailStackButton"
Me.mailStackButton.Padding = New System.Windows.Forms.Padding(3)
Me.mailStackButton.Size = New System.Drawing.Size(149, 27)
Me.mailStackButton.Text = " Mail"
Me.mailStackButton.TextAlign = System.Drawing.ContentAlignment.MiddleLeft
'

'calendarStackButton
'

Me.calendarStackButton.CheckOnClick = True
Me.calendarStackButton.ImageAlign = System.Drawing.ContentAlignment.MiddleLeft
Me.calendarStackButton.ImageScaling = System.Windows.Forms.ToolStripItemImageScaling.None
Me.calendarStackButton.ImageTransparentColor = System.Drawing.Color.FromArgb(CType(CType(238, Byte),
Integer), CType(CType(238, Byte), Integer), CType(CType(238, Byte), Integer))
Me.calendarStackButton.Margin = New System.Windows.Forms.Padding(0)
Me.calendarStackButton.Name = "calendarStackButton"
Me.calendarStackButton.Padding = New System.Windows.Forms.Padding(3)
Me.calendarStackButton.Size = New System.Drawing.Size(149, 27)
Me.calendarStackButton.Text = " Calendar"
Me.calendarStackButton.TextAlign = System.Drawing.ContentAlignment.MiddleLeft
'

'contactsStackButton
'

Me.contactsStackButton.CheckOnClick = True
Me.contactsStackButton.ImageAlign = System.Drawing.ContentAlignment.MiddleLeft
Me.contactsStackButton.ImageScaling = System.Windows.Forms.ToolStripItemImageScaling.None
Me.contactsStackButton.ImageTransparentColor = System.Drawing.Color.FromArgb(CType(CType(238, Byte),
Integer), CType(CType(238, Byte), Integer), CType(CType(238, Byte), Integer))
Me.contactsStackButton.Margin = New System.Windows.Forms.Padding(0)
Me.contactsStackButton.Name = "contactsStackButton"
Me.contactsStackButton.Padding = New System.Windows.Forms.Padding(3)
Me.contactsStackButton.Size = New System.Drawing.Size(149, 27)
Me.contactsStackButton.Text = " Contacts"
Me.contactsStackButton.TextAlign = System.Drawing.ContentAlignment.MiddleLeft
'

'tasksStackButton
'

Me.tasksStackButton.CheckOnClick = True
Me.tasksStackButton.ImageAlign = System.Drawing.ContentAlignment.MiddleLeft
Me.tasksStackButton.ImageScaling = System.Windows.Forms.ToolStripItemImageScaling.None
Me.tasksStackButton.ImageTransparentColor = System.Drawing.Color.FromArgb(CType(CType(238, Byte),
Integer), CType(CType(238, Byte), Integer), CType(CType(238, Byte), Integer))
Me.tasksStackButton.Margin = New System.Windows.Forms.Padding(0)
Me.tasksStackButton.Name = "tasksStackButton"
Me.tasksStackButton.Padding = New System.Windows.Forms.Padding(3)
Me.tasksStackButton.Size = New System.Drawing.Size(149, 27)
Me.tasksStackButton.Text = " Tasks"
Me.tasksStackButton.TextAlign = System.Drawing.ContentAlignment.MiddleLeft
'

```

```
'StackView
'
Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)
Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
Me.Controls.Add(Me.stackStrip)
Me.Name = "StackView"
Me.stackStrip.ResumeLayout(False)
Me.stackStrip.PerformLayout()
Me.ResumeLayout(False)
Me.PerformLayout()

End Sub

#End Region
End Class
```

Compiling the Code

This example requires:

- References to the System.Drawing and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [Walkthrough: Creating a Professionally Styled ToolStrip Control](#).

See Also

[MenuStrip](#)

[ToolStrip](#)

[StatusStrip](#)

[ToolStrip Control](#)

[How to: Provide Standard Menu Items to a Form](#)

How to: Create an MDI Form with Menu Merging and ToolStrip Controls

5/4/2018 • 7 min to read • [Edit Online](#)

The `System.Windows.Forms` namespace supports multiple document interface (MDI) applications, and the `MenuStrip` control supports menu merging. MDI forms can also `ToolStrip` controls.

There is extensive support for this feature in Visual Studio.

Also see [Walkthrough: Creating an MDI Form with Menu Merging and ToolStrip Controls](#).

Example

The following code example demonstrates how to use `ToolStripPanel` controls with an MDI form. The form also supports menu merging with child menus.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace MdiFormCS
{
    // This code example demonstrates an MDI form
    // that supports menu merging and moveable
    // ToolStrip controls
    public class Form1 : Form
    {
        private MenuStrip menuStrip1;
        private ToolStripMenuItem toolStripMenuItem1;
        private ToolStripMenuItem newToolStripMenuItem;
        private ToolStripPanel toolStripPanel1;
        private ToolStrip toolStrip1;
        private ToolStripPanel toolStripPanel2;
        private ToolStrip toolStrip2;
        private ToolStripPanel toolStripPanel3;
        private ToolStrip toolStrip3;
        private ToolStripPanel toolStripPanel4;
        private ToolStrip toolStrip4;

        private System.ComponentModel.IContainer components = null;

        public Form1()
        {
            InitializeComponent();
        }

        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        // This method creates a new ChildForm instance
        // and attaches it to the MDI parent form.
        private void newToolStripMenuItem_Click(object sender, EventArgs e)
```

```

{
    ChildForm f = new ChildForm();
    f.MdiParent = this;
    f.Text = "Form - " + this.MdiChildren.Length.ToString();
    f.Show();
}

#region Windows Form Designer generated code

private void InitializeComponent()
{
    this.menuStrip1 = new System.Windows.Forms.MenuStrip();
    this.toolStripMenuItem1 = new System.Windows.Forms.ToolStripItem();
    this.newToolStripMenuItem = new System.Windows.Forms.ToolStripItem();
    this.toolStripPanel1 = new System.Windows.Forms.ToolStripPanel();
    this.toolStrip1 = new System.Windows.Forms.ToolStrip();
    this.toolStripPanel2 = new System.Windows.Forms.ToolStripPanel();
    this.toolStrip2 = new System.Windows.Forms.ToolStrip();
    this.toolStripPanel3 = new System.Windows.Forms.ToolStripPanel();
    this.toolStrip3 = new System.Windows.Forms.ToolStrip();
    this.toolStripPanel4 = new System.Windows.Forms.ToolStripPanel();
    this.toolStrip4 = new System.Windows.Forms.ToolStrip();
    this.menuStrip1.SuspendLayout();
    this.toolStripPanel1.SuspendLayout();
    this.toolStripPanel2.SuspendLayout();
    this.toolStripPanel3.SuspendLayout();
    this.toolStripPanel4.SuspendLayout();
    this.SuspendLayout();
    //
    // menuStrip1
    //
    this.menuStrip1.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
    this.toolStripMenuItem1});
    this.menuStrip1.Location = new System.Drawing.Point(0, 0);
    this.menuStrip1.MdiWindowListItem = this.toolStripMenuItem1;
    this.menuStrip1.Name = "menuStrip1";
    this.menuStrip1.Size = new System.Drawing.Size(292, 24);
    this.menuStrip1.TabIndex = 0;
    this.menuStrip1.Text = "menuStrip1";
    //
    // toolStripMenuItem1
    //
    this.toolStripMenuItem1.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] {
    this.newToolStripMenuItem});
    this.toolStripMenuItem1.Name = "toolStripMenuItem1";
    this.toolStripMenuItem1.Size = new System.Drawing.Size(57, 20);
    this.toolStripMenuItem1.Text = "Window";
    //
    // newToolStripMenuItem
    //
    this.newToolStripMenuItem.Name = "newToolStripMenuItem";
    this.newToolStripMenuItem.Size = new System.Drawing.Size(106, 22);
    this.newToolStripMenuItem.Text = "New";
    this.newToolStripMenuItem.Click += new System.EventHandler(this.newToolStripMenuItem_Click);
    //
    // toolStripPanel1
    //
    this.toolStripPanel1.Controls.Add(this.toolStrip1);
    this.toolStripPanel1.Dock = System.Windows.Forms.DockStyle.Left;
    this.toolStripPanel1.Location = new System.Drawing.Point(0, 49);
    this.toolStripPanel1.Name = "toolStripPanel1";
    this.toolStripPanel1.Orientation = System.Windows.Forms.Orientation.Vertical;
    this.toolStripPanel1.RowMargin = new System.Windows.Forms.Padding(0, 3, 0, 0);
    this.toolStripPanel1.Size = new System.Drawing.Size(26, 199);
    //
    // toolStrip1
    //
    this.toolStrip1.Dock = System.Windows.Forms.DockStyle.None;
    this.toolStrip1.Location = new System.Drawing.Point(0, 3);
}

```

```
this.toolStrip1.Name = "toolStrip1";
this.toolStrip1.Size = new System.Drawing.Size(26, 109);
this.toolStrip1.TabIndex = 0;
//
// toolStripPanel2
//
this.toolStripPanel2.Controls.Add(this.toolStrip2);
this.toolStripPanel2.Dock = System.Windows.Forms.DockStyle.Top;
this.toolStripPanel2.Location = new System.Drawing.Point(0, 24);
this.toolStripPanel2.Name = "toolStripPanel2";
this.toolStripPanel2.Orientation = System.Windows.Forms.Orientation.Horizontal;
this.toolStripPanel2.RowMargin = new System.Windows.Forms.Padding(3, 0, 0, 0);
this.toolStripPanel2.Size = new System.Drawing.Size(292, 25);
//
// toolStrip2
//
this.toolStrip2.Dock = System.Windows.Forms.DockStyle.None;
this.toolStrip2.Location = new System.Drawing.Point(3, 0);
this.toolStrip2.Name = "toolStrip2";
this.toolStrip2.Size = new System.Drawing.Size(109, 25);
this.toolStrip2.TabIndex = 0;
//
// toolStripPanel3
//
this.toolStripPanel3.Controls.Add(this.toolStrip3);
this.toolStripPanel3.Dock = System.Windows.Forms.DockStyle.Right;
this.toolStripPanel3.Location = new System.Drawing.Point(266, 49);
this.toolStripPanel3.Name = "toolStripPanel3";
this.toolStripPanel3.Orientation = System.Windows.Forms.Orientation.Vertical;
this.toolStripPanel3.RowMargin = new System.Windows.Forms.Padding(0, 3, 0, 0);
this.toolStripPanel3.Size = new System.Drawing.Size(26, 199);
//
// toolStrip3
//
this.toolStrip3.Dock = System.Windows.Forms.DockStyle.None;
this.toolStrip3.Location = new System.Drawing.Point(0, 3);
this.toolStrip3.Name = "toolStrip3";
this.toolStrip3.Size = new System.Drawing.Size(26, 109);
this.toolStrip3.TabIndex = 0;
//
// toolStripPanel4
//
this.toolStripPanel4.Controls.Add(this.toolStrip4);
this.toolStripPanel4.Dock = System.Windows.Forms.DockStyle.Bottom;
this.toolStripPanel4.Location = new System.Drawing.Point(0, 248);
this.toolStripPanel4.Name = "toolStripPanel4";
this.toolStripPanel4.Orientation = System.Windows.Forms.Orientation.Horizontal;
this.toolStripPanel4.RowMargin = new System.Windows.Forms.Padding(3, 0, 0, 0);
this.toolStripPanel4.Size = new System.Drawing.Size(292, 25);
//
// toolStrip4
//
this.toolStrip4.Dock = System.Windows.Forms.DockStyle.None;
this.toolStrip4.Location = new System.Drawing.Point(3, 0);
this.toolStrip4.Name = "toolStrip4";
this.toolStrip4.Size = new System.Drawing.Size(109, 25);
this.toolStrip4.TabIndex = 0;
//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(292, 273);
this.Controls.Add(this.toolStripPanel3);
this.Controls.Add(this.toolStripPanel1);
this.Controls.Add(this.toolStripPanel2);
this.Controls.Add(this.menuStrip1);
this.Controls.Add(this.toolStripPanel4);
this.IsMdiContainer = true;
```

```

        this.MainMenuStrip = this.menuStrip1;
        this.Name = "Form1";
        this.Text = "Form1";
        this.menuStrip1.ResumeLayout(false);
        this.menuStrip1.PerformLayout();
        this.toolStripPanel1.ResumeLayout(false);
        this.toolStripPanel1.PerformLayout();
        this.toolStripPanel2.ResumeLayout(false);
        this.toolStripPanel2.PerformLayout();
        this.toolStripPanel3.ResumeLayout(false);
        this.toolStripPanel3.PerformLayout();
        this.toolStripPanel4.ResumeLayout(false);
        this.toolStripPanel4.PerformLayout();
        this.ResumeLayout(false);
        this.PerformLayout();

    }

    #endregion
}

public class ChildForm : Form
{
    private System.Windows.Forms.MenuStrip menuStrip1;
    private System.Windows.Forms.ToolStripItem toolStripMenuItem1;
    private System.ComponentModel.IContainer components = null;

    public ChildForm()
    {
        InitializeComponent();
    }

    protected override void Dispose(bool disposing)
    {
        if (disposing && (components != null))
        {
            components.Dispose();
        }
        base.Dispose(disposing);
    }

    #region Windows Form Designer generated code

    private void InitializeComponent()
    {
        this.menuStrip1 = new System.Windows.Forms.MenuStrip();
        this.toolStripMenuItem1 = new System.Windows.Forms.ToolStripItem();
        this.menuStrip1.SuspendLayout();
        this.SuspendLayout();
        // 
        // menuStrip1
        // 
        this.menuStrip1.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
        this.toolStripMenuItem1});
        this.menuStrip1.Location = new System.Drawing.Point(0, 0);
        this.menuStrip1.Name = "menuStrip1";
        this.menuStrip1.Size = new System.Drawing.Size(292, 24);
        this.menuStrip1.TabIndex = 0;
        this.menuStrip1.Text = "menuStrip1";
        // 
        // toolStripMenuItem1
        // 
        this.toolStripMenuItem1.Name = "toolStripMenuItem1";
        this.toolStripMenuItem1.Size = new System.Drawing.Size(90, 20);
        this.toolStripMenuItem1.Text = "ChildMenuItem";
        // 
        // ChildForm
        // 
        this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
    }
}

```

```

        this.AutoScaleDimensions = new System.Drawing.Size(80, 150);
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.ClientSize = new System.Drawing.Size(292, 273);
        this.Controls.Add(this.menuStrip1);
        this.MainMenuStrip = this.menuStrip1;
        this.Name = "ChildForm";
        this.Text = "ChildForm";
        this.menuStrip1.ResumeLayout(false);
        this.menuStrip1.PerformLayout();
        this.ResumeLayout(false);
        this.PerformLayout();
    }

}

#endregion
}

static class Program
{
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.Run(new Form1());
    }
}
}
}

```

```

Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Drawing
Imports System.Windows.Forms

' This code example demonstrates an MDI form
' that supports menu merging and moveable
' ToolStrip controls
Public Class Form1
    Inherits Form
    Private menuStrip1 As MenuStrip
    Private toolStripMenuItem1 As ToolStripMenuItem
    Private WithEvents newToolStripMenuItem As ToolStripMenuItem
    Private toolStripPanel1 As ToolStripPanel
    Private toolStrip1 As ToolStrip
    Private toolStripPanel2 As ToolStripPanel
    Private toolStrip2 As ToolStrip
    Private toolStripPanel3 As ToolStripPanel
    Private toolStrip3 As ToolStrip
    Private toolStripPanel4 As ToolStripPanel
    Private toolStrip4 As ToolStrip

    Private components As System.ComponentModel.IContainer = Nothing

    Public Sub New()
        InitializeComponent()
    End Sub

    Protected Overrides Sub Dispose(disposing As Boolean)
        If disposing AndAlso (components IsNot Nothing) Then
            components.Dispose()
        End If
        MyBase.Dispose(disposing)
    End Sub

```

```

' This method creates a new ChildForm instance
' and attaches it to the MDI parent form.
Private Sub newToolStripMenuItem_Click( _
    ByVal sender As Object, _
    ByVal e As EventArgs) _
Handles newToolStripMenuItem.Click

    Dim f As New ChildForm()
    f.MdiParent = Me
    f.Text = "Form - " + Me.MdiChildren.Length.ToString()
    f.Show()

End Sub

#Region "Windows Form Designer generated code"

Private Sub InitializeComponent()
    Me.menuStrip1 = New System.Windows.Forms.MenuStrip
    Me.toolStripMenuItem1 = New System.Windows.Forms.ToolStripItem
    Me.newToolStripMenuItem = New System.Windows.Forms.ToolStripItem
    Me.toolStripPanel1 = New System.Windows.Forms.ToolStripPanel
    Me.toolStrip1 = New System.Windows.Forms.ToolStrip
    Me.toolStripPanel2 = New System.Windows.Forms.ToolStripPanel
    Me.toolStripPanel3 = New System.Windows.Forms.ToolStripPanel
    Me.toolStripPanel4 = New System.Windows.Forms.ToolStripPanel
    Me.toolStripPanel14 = New System.Windows.Forms.ToolStripPanel
    Me.toolStrip4 = New System.Windows.Forms.ToolStrip
    Me.menuStrip1.SuspendLayout()
    Me.toolStripPanel1.SuspendLayout()
    Me.toolStripPanel2.SuspendLayout()
    Me.toolStripPanel3.SuspendLayout()
    Me.toolStripPanel14.SuspendLayout()
    Me.SuspendLayout()
    '
    'menuStrip1
    '
    Me.menuStrip1.Items.AddRange(New System.Windows.Forms.ToolStripItem() {Me.toolStripMenuItem1})
    Me.menuStrip1.Location = New System.Drawing.Point(0, 0)
    Me.menuStrip1.MdiWindowListItem = Me.toolStripMenuItem1
    Me.menuStrip1.Name = "menuStrip1"
    Me.menuStrip1.Size = New System.Drawing.Size(292, 24)
    Me.menuStrip1.TabIndex = 0
    Me.menuStrip1.Text = "menuStrip1"
    '
    'toolStripMenuItem1
    '
    Me.toolStripMenuItem1.DropDownItems.AddRange(New System.Windows.Forms.ToolStripItem()
    {Me.newToolStripMenuItem})
    Me.toolStripMenuItem1.Name = "toolStripMenuItem1"
    Me.toolStripMenuItem1.Size = New System.Drawing.Size(57, 20)
    Me.toolStripMenuItem1.Text = "Window"
    '
    'newToolStripMenuItem
    '
    Me.newToolStripMenuItem.Name = "newToolStripMenuItem"
    Me.newToolStripMenuItem.Size = New System.Drawing.Size(106, 22)
    Me.newToolStripMenuItem.Text = "New"
    '
    'toolStripPanel1
    '
    Me.toolStripPanel1.Controls.Add(Me.toolStrip1)
    Me.toolStripPanel1.Dock = System.Windows.Forms.DockStyle.Left
    Me.toolStripPanel1.Location = New System.Drawing.Point(0, 49)
    Me.toolStripPanel1.Name = "toolStripPanel1"
    Me.toolStripPanel1.Orientation = System.Windows.Forms.Orientation.Vertical
    Me.toolStripPanel1.RowMargin = New System.Windows.Forms.Padding(0, 3, 0, 0)
    Me.toolStripPanel1.Size = New System.Drawing.Size(26, 199)
    '

```

```
'toolStrip1
'

Me.toolStrip1.Dock = System.Windows.Forms.DockStyle.None
Me.toolStrip1.Location = New System.Drawing.Point(0, 3)
Me.toolStrip1.Name = "toolStrip1"
Me.toolStrip1.Size = New System.Drawing.Size(26, 109)
Me.toolStrip1.TabIndex = 0
'

'toolStripPanel2
'

Me.toolStripPanel2.Controls.Add(Me.toolStrip2)
Me.toolStripPanel2.Dock = System.Windows.Forms.DockStyle.Top
Me.toolStripPanel2.Location = New System.Drawing.Point(0, 24)
Me.toolStripPanel2.Name = "toolStripPanel2"
Me.toolStripPanel2.Orientation = System.Windows.Forms.Orientation.Horizontal
Me.toolStripPanel2.RowMargin = New System.Windows.Forms.Padding(3, 0, 0, 0)
Me.toolStripPanel2.Size = New System.Drawing.Size(292, 25)
'

'toolStrip2
'

Me.toolStrip2.Dock = System.Windows.Forms.DockStyle.None
Me.toolStrip2.Location = New System.Drawing.Point(3, 0)
Me.toolStrip2.Name = "toolStrip2"
Me.toolStrip2.Size = New System.Drawing.Size(109, 25)
Me.toolStrip2.TabIndex = 0
'

'toolStripPanel3
'

Me.toolStripPanel3.Controls.Add(Me.toolStrip3)
Me.toolStripPanel3.Dock = System.Windows.Forms.DockStyle.Right
Me.toolStripPanel3.Location = New System.Drawing.Point(266, 49)
Me.toolStripPanel3.Name = "toolStripPanel3"
Me.toolStripPanel3.Orientation = System.Windows.Forms.Orientation.Vertical
Me.toolStripPanel3.RowMargin = New System.Windows.Forms.Padding(0, 3, 0, 0)
Me.toolStripPanel3.Size = New System.Drawing.Size(26, 199)
'

'toolStrip3
'

Me.toolStrip3.Dock = System.Windows.Forms.DockStyle.None
Me.toolStrip3.Location = New System.Drawing.Point(0, 3)
Me.toolStrip3.Name = "toolStrip3"
Me.toolStrip3.Size = New System.Drawing.Size(26, 109)
Me.toolStrip3.TabIndex = 0
'

'toolStripPanel4
'

Me.toolStripPanel4.Controls.Add(Me.toolStrip4)
Me.toolStripPanel4.Dock = System.Windows.Forms.DockStyle.Bottom
Me.toolStripPanel4.Location = New System.Drawing.Point(0, 248)
Me.toolStripPanel4.Name = "toolStripPanel4"
Me.toolStripPanel4.Orientation = System.Windows.Forms.Orientation.Horizontal
Me.toolStripPanel4.RowMargin = New System.Windows.Forms.Padding(3, 0, 0, 0)
Me.toolStripPanel4.Size = New System.Drawing.Size(292, 25)
'

'toolStrip4
'

Me.toolStrip4.Dock = System.Windows.Forms.DockStyle.None
Me.toolStrip4.Location = New System.Drawing.Point(3, 0)
Me.toolStrip4.Name = "toolStrip4"
Me.toolStrip4.Size = New System.Drawing.Size(109, 25)
Me.toolStrip4.TabIndex = 0
'

'Form1
'

Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)
Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
Me.ClientSize = New System.Drawing.Size(292, 273)
Me.Controls.Add(Me.toolStripPanel3)
Me.Controls.Add(Me.toolStripPanel2)
Me.Controls.Add(Me.toolStripPanel1)
```

```

Me.Controls.Add(Me.toolStripPanel1)
Me.Controls.Add(Me.toolStripPanel2)
Me.Controls.Add(Me.menuStrip1)
Me.Controls.Add(Me.toolStripPanel4)
Me.IsMdiContainer = True
Me.MainMenuStrip = Me.menuStrip1
Me.Name = "Form1"
Me.Text = "Form1"
Me.menuStrip1.ResumeLayout(False)
Me.menuStrip1.PerformLayout()
Me.toolStripPanel1.ResumeLayout(False)
Me.toolStripPanel1.PerformLayout()
Me.toolStripPanel2.ResumeLayout(False)
Me.toolStripPanel2.PerformLayout()
Me.toolStripPanel3.ResumeLayout(False)
Me.toolStripPanel3.PerformLayout()
Me.toolStripPanel4.ResumeLayout(False)
Me.toolStripPanel4.PerformLayout()
Me.ResumeLayout(False)
Me.PerformLayout()

End Sub

#End Region
End Class

Public Class ChildForm
    Inherits Form
    Private menuStrip1 As System.Windows.Forms.MenuStrip
    Private toolStripMenuItem1 As System.Windows.Forms.ToolStripItem
    Private components As System.ComponentModel.IContainer = Nothing

    Public Sub New()
        InitializeComponent()
    End Sub

    Protected Overrides Sub Dispose(disposing As Boolean)
        If disposing AndAlso (components IsNot Nothing) Then
            components.Dispose()
        End If
        MyBase.Dispose(disposing)
    End Sub

    #Region "Windows Form Designer generated code"

    Private Sub InitializeComponent()
        Me.menuStrip1 = New System.Windows.Forms.MenuStrip()
        Me.toolStripMenuItem1 = New System.Windows.Forms.ToolStripItem()
        Me.menuStrip1.SuspendLayout()
        Me.SuspendLayout()

        ' menuStrip1
        ' 

        Me.menuStrip1.Items.AddRange(New System.Windows.Forms.ToolStripItem() {Me.toolStripMenuItem1})
        Me.menuStrip1.Location = New System.Drawing.Point(0, 0)
        Me.menuStrip1.Name = "menuStrip1"
        Me.menuStrip1.Size = New System.Drawing.Size(292, 24)
        Me.menuStrip1.TabIndex = 0
        Me.menuStrip1.Text = "menuStrip1"
        ' 

        ' toolStripMenuItem1
        ' 

        Me.toolStripMenuItem1.Name = "toolStripMenuItem1"
        Me.toolStripMenuItem1.Size = New System.Drawing.Size(90, 20)
        Me.toolStripMenuItem1.Text = "ChildMenuItem"
        '
    End Sub

```

```

' ChildForm
'

Me.AutoScaleDimensions = New System.Drawing.SizeF(6F, 13F)
Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
Me.ClientSize = New System.Drawing.Size(292, 273)
Me.Controls.Add(menuStrip1)
Me.MainMenuStrip = Me.menuStrip1
Me.Name = "ChildForm"
Me.Text = "ChildForm"
Me.menuStrip1.ResumeLayout(False)
Me.menuStrip1.PerformLayout()
Me.ResumeLayout(False)
Me.PerformLayout()

End Sub

#End Region
End Class

Public Class Program

    ' The main entry point for the application.
    <STAThread()> _
    Shared Sub Main()
        Application.EnableVisualStyles()
        Application.Run(New Form1())
    End Sub
End Class

```

Compiling the Code

This code example requires:

- References to the System.Drawing and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[ToolStrip Control](#)

How to: Create an MDI Form with ToolStripPanel Controls

5/4/2018 • 4 min to read • [Edit Online](#)

You can create a multiple document interface (MDI) form that has [ToolStrip](#) controls framing it on all four sides.

Example

The following code example demonstrates how to use docked [ToolStripPanel](#) controls to frame an MDI window with four [ToolStrip](#) controls.

In the example, the [Join](#) method attaches the [ToolStrip](#) controls to the corresponding [ToolStripPanel](#) controls.

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Drawing;
```

```
Imports System
Imports System.Collections.Generic
Imports System.Windows.Forms
Imports System.Drawing
```

```
// This code example demonstrates how to use ToolStripPanel
// controls with a multiple document interface (MDI).
public class Form1 : Form
{
    public Form1()
    {
        // Make the Form an MDI parent.
        this.IsMdiContainer = true;

        // Create ToolStripPanel controls.
        ToolStripPanel tspTop = new ToolStripPanel();
        ToolStripPanel tspBottom = new ToolStripPanel();
        ToolStripPanel tspLeft = new ToolStripPanel();
        ToolStripPanel tspRight = new ToolStripPanel();

        // Dock the ToolStripPanel controls to the edges of the form.
        tspTop.Dock = DockStyle.Top;
        tspBottom.Dock = DockStyle.Bottom;
        tspLeft.Dock = DockStyle.Left;
        tspRight.Dock = DockStyle.Right;

        // Create ToolStrip controls to move among the
        // ToolStripPanel controls.

        // Create the "Top" ToolStrip control and add
        // to the corresponding ToolStripPanel.
        ToolStrip tsTop = new ToolStrip();
        tsTop.Items.Add("Top");
        tspTop.Join(tsTop);

        // Create the "Bottom" ToolStrip control and add
        // to the corresponding ToolStripPanel.
        ToolStrip tsBottom = new ToolStrip();
```

```

        tsBottom.Items.Add("Bottom");
        tspBottom.Join(tsBottom);

        // Create the "Right" ToolStrip control and add
        // to the corresponding ToolStripPanel.
        ToolStrip tsRight = new ToolStrip();
        tsRight.Items.Add("Right");
        tspRight.Join(tsRight);

        // Create the "Left" ToolStrip control and add
        // to the corresponding ToolStripPanel.
        ToolStrip tsLeft = new ToolStrip();
        tsLeft.Items.Add("Left");
        tspLeft.Join(tsLeft);

        // Create a MenuStrip control with a new window.
        MenuStrip ms = new MenuStrip();
        ToolStripMenuItem windowMenu = new ToolStripMenuItem("Window");
        ToolStripMenuItem windowNewMenu = new ToolStripMenuItem("New", null, new
EventHandler(windowNewMenu_Click));
        windowMenu.DropDownItems.Add(windowNewMenu);
        ((ToolStripDropDownMenu)(windowMenu.DropDown)).ShowImageMargin = false;
        ((ToolStripDropDownMenu)(windowMenu.DropDown)).ShowCheckMargin = true;

        // Assign the ToolStripMenuItem that displays
        // the list of child forms.
        ms.MdiWindowListItem = windowMenu;

        // Add the window ToolStripMenuItem to the MenuStrip.
        ms.Items.Add(windowMenu);

        // Dock the MenuStrip to the top of the form.
        ms.Dock = DockStyle.Top;

        // The Form.MainMenuStrip property determines the merge target.
        this.MainMenuStrip = ms;

        // Add the ToolStripPanels to the form in reverse order.
        this.Controls.Add(tspRight);
        this.Controls.Add(tspLeft);
        this.Controls.Add(tspBottom);
        this.Controls.Add(tspTop);

        // Add the MenuStrip last.
        // This is important for correct placement in the z-order.
        this.Controls.Add(ms);
    }

    // This event handler is invoked when
    // the "New" ToolStripMenuItem is clicked.
    // It creates a new Form and sets its MdiParent
    // property to the main form.
    void windowNewMenu_Click(object sender, EventArgs e)
    {
        Form f = new Form();
        f.MdiParent = this;
        f.Text = "Form - " + this.MdiChildren.Length.ToString();
        f.Show();
    }
}

```

```

' This code example demonstrates how to use ToolStripPanel
' controls with a multiple document interface (MDI).
Public Class Form1
    Inherits Form

    Public Sub New()

```

```

' Make the Form an MDI parent.
Me.IsMdiContainer = True

' Create ToolStripPanel controls.
Dim tspTop As New ToolStripPanel()
Dim tspBottom As New ToolStripPanel()
Dim tspLeft As New ToolStripPanel()
Dim tspRight As New ToolStripPanel()

' Dock the ToolStripPanel controls to the edges of the form.
tspTop.Dock = DockStyle.Top
tspBottom.Dock = DockStyle.Bottom
tspLeft.Dock = DockStyle.Left
tspRight.Dock = DockStyle.Right

' Create ToolStrip controls to move among the
' ToolStripPanel controls.
' Create the "Top" ToolStrip control and add
' to the corresponding ToolStripPanel.
Dim tsTop As New ToolStrip()
tsTop.Items.Add("Top")
tspTop.Join(tsTop)

' Create the "Bottom" ToolStrip control and add
' to the corresponding ToolStripPanel.
Dim tsBottom As New ToolStrip()
tsBottom.Items.Add("Bottom")
tspBottom.Join(tsBottom)

' Create the "Right" ToolStrip control and add
' to the corresponding ToolStripPanel.
Dim tsRight As New ToolStrip()
tsRight.Items.Add("Right")
tspRight.Join(tsRight)

' Create the "Left" ToolStrip control and add
' to the corresponding ToolStripPanel.
Dim tsLeft As New ToolStrip()
tsLeft.Items.Add("Left")
tspLeft.Join(tsLeft)

' Create a MenuStrip control with a new window.
Dim ms As New MenuStrip()
Dim windowMenu As New ToolStripMenuItem("Window")
Dim windowNewMenu As New ToolStripMenuItem("New", Nothing, New EventHandler(AddressOf
windowNewMenu_Click))
    windowMenu.DropDownItems.Add(windowNewMenu)
    CType(windowMenu.DropDown, ToolStripDropDownMenu).ShowImageMargin = False
    CType(windowMenu.DropDown, ToolStripDropDownMenu).ShowCheckMargin = True

' Assign the ToolStripMenuItem that displays
' the list of child forms.
ms.MdiWindowListItem = windowMenu

' Add the window ToolStripMenuItem to the MenuStrip.
ms.Items.Add(windowMenu)

' Dock the MenuStrip to the top of the form.
ms.Dock = DockStyle.Top

' The Form.MainMenuStrip property determines the merge target.
Me.MainMenuStrip = ms

' Add the ToolStripPanels to the form in reverse order.
Me.Controls.Add(tspRight)
Me.Controls.Add(tspLeft)
Me.Controls.Add(tspBottom)
Me.Controls.Add(tspTop)

```

```

' Add the MenuStrip last.
' This is important for correct placement in the z-order.
Me.Controls.Add(ms)
End Sub

' This event handler is invoked when
' the "New" ToolStripMenuItem is clicked.
' It creates a new Form and sets its MdiParent
' property to the main form.
Private Sub windowNewMenu_Click(ByVal sender As Object, ByVal e As EventArgs)
    Dim f As New Form()
    f.MdiParent = Me
    f.Text = "Form - " + Me.MdiChildren.Length.ToString()
    f.Show()
End Sub
End Class

```

Compiling the Code

This example requires:

- References to the System.Drawing and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[ToolStrip](#)
[ToolStripPanel](#)
[Join](#)
[ToolStripItem](#)
[ToolStripMenuItem](#)
[ToolStrip Control](#)

How to: Change the Appearance of ToolStrip Text and Images in Windows Forms

5/4/2018 • 1 min to read • [Edit Online](#)

You can control whether text and images are displayed on a [ToolStripItem](#) and how they are aligned relative to each other and the [ToolStrip](#).

To define what is displayed on a ToolStripItem

- Set the [DisplayStyle](#) property to the desired value. The possibilities are `Image`, `ImageAndText`, `None`, and `Text`. The default is `ImageAndText`.

```
ToolStripButton2.DisplayStyle = _  
System.Windows.Forms.ToolStripItemDisplayStyle.Image
```

```
toolStripButton2.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image;
```

To align text on a ToolStripItem

- Set the [TextAlign](#) property to the desired value. The possibilities are any combination of top, middle, and bottom with left, center, and right. The default is `MiddleCenter`.

```
ToolStripSplitButton1.TextAlign = _  
System.Drawing.ContentAlignment.MiddleRight
```

```
toolStripSplitButton1.TextAlign = System.Drawing.ContentAlignment.MiddleRight;
```

To align an image on a ToolStripItem

- Set the [ImageAlign](#) property to the desired value. The possibilities are any combination of top, middle, and bottom with left, center, and right. The default is `MiddleLeft`.

```
ToolStripSplitButton1.ImageAlign = _  
System.Drawing.ContentAlignment.MiddleRight
```

```
toolStripSplitButton1.ImageAlign = System.Drawing.ContentAlignment.MiddleRight;
```

To define how ToolStripItem text and images are displayed relative to each other

- Set the [TextImageRelation](#) property to the desired value. The possibilities are `ImageAboveText`, `ImageBeforeText`, `Overlay`, `TextAboveImage`, and `TextBeforeImage`. The default is `ImageBeforeText`.

```
ToolStripButton1.TextImageRelation = _  
System.Windows.Forms.TextImageRelation.ImageAboveText
```

```
toolStripButton1.TextImageRelation = System.Windows.Forms.TextImageRelation.ImageAboveText;
```

See Also

[ToolStrip](#)

[ToolStrip Control Overview](#)

[ToolStrip Control Architecture](#)

[ToolStrip Technology Summary](#)

How to: Change the Spacing and Alignment of ToolStrip Items in Windows Forms

5/4/2018 • 1 min to read • [Edit Online](#)

The [ToolStrip](#) control fully supports layout features such as sizing, the spacing of [ToolStripItem](#) controls relative to each other, the arrangement of controls on the [ToolStrip](#), and the spacing of controls relative to the [ToolStrip](#).

Because the default value of the [AutoSize](#) property is `true`, controls are sized automatically unless you set the [AutoSize](#) property to `false`.

To manually size a ToolStripItem

1. Set the [AutoSize](#) property to `false` for the associated control.

```
ToolStripButton1.AutoSize = False
```

```
toolStripButton1.AutoSize = false;
```

2. Set the [Size](#) property the way you want for the associated [ToolStripItem](#).

To set the spacing of a ToolStripItem

1. Insert the desired values, in pixels, into the [Margin](#) property of the associated control.

The values of the [Margin](#) property specify the spacing between the item and adjacent items in this order: Left, Top, Right, and Bottom.

```
ToolStripTextBox1.Margin = New System.Windows.Forms.Padding _  
(3, 0, 3, 0)
```

```
toolStripTextBox1.Margin = new System.Windows.Forms.Padding  
(3, 0, 3, 0);
```

To align a ToolStripItem to the right side of the ToolStrip

1. Set the [Alignment](#) property to [Right](#) for the associated control. By default, [Alignment](#) is set to [Left](#), which aligns controls to the left side of the [ToolStrip](#).

```
ToolStripSplitButton1.Alignment = _  
System.Windows.Forms.ToolStripItemAlignment.Right
```

```
toolStripSplitButton1.Alignment =  
System.Windows.Forms.ToolStripItemAlignment.Right;
```

To arrange ToolStrip items on the ToolStrip

- Set the [LayoutStyle](#) property to the value of [ToolStripLayoutStyle](#) that you want.

```
ToolStripDropDown1.LayoutStyle =  
    System.Windows.Forms.ToolStripItemLayoutStyle.Flow
```

```
toolStripDropDown1.LayoutStyle =  
    System.Windows.Forms.ToolStripItemLayoutStyle.Flow;
```

See Also

[ToolStrip](#)
[Layout](#)
[LayoutCompleted](#)
[LayoutSettings](#)
[TextImageRelation](#)
[Placement](#)
[CanOverflow](#)
[ToolStrip Control Overview](#)
[ToolStrip Control Architecture](#)
[ToolStrip Technology Summary](#)

How to: Create and Set a Custom Renderer for the ToolStrip Control in Windows Forms

5/4/2018 • 1 min to read • [Edit Online](#)

ToolStrip controls give easy support to themes and styles. You can achieve completely custom appearance and behavior (look and feel) by setting either the `ToolStrip.Renderer` property or the `ToolStripManager.Renderer` property to a custom renderer.

You can assign renderers to each individual `ToolStrip`, `MenuStrip`, `ContextMenuStrip`, or `StatusStrip` control, or you can use the `Renderer` property to affect all objects by setting the `ToolStrip.RenderMode` property to `ToolStripRenderMode.ManagerRenderMode`.

NOTE

`RenderMode` returns `Custom` only if the value of `ToolStrip.Renderer` is not `null`.

To create a custom renderer

1. Extend the `ToolStripRenderer` class.
2. Implement desired custom rendering by overriding appropriate `On...` members

```
Public Class RedTextRenderer
    Inherits System.Windows.Forms.ToolStripRenderer
    Protected Overrides Sub OnRenderItemText(ByVal e As _
        ToolStripItemTextEventArgs)
        e.TextColor = Color.Red
        e.TextFont = New Font("Helvetica", 7, FontStyle.Bold)
        MyBase.OnRenderItemText(e)
    End Sub
End Class
```

```
public class RedTextRenderer : _
    System.Windows.Forms.ToolStripRenderer
{
    protected override void _
        OnRenderItemText(ToolStripItemTextEventArgs e)
    {
        e.TextColor = Color.Red;
        e.TextFont = new Font("Helvetica", 7, FontStyle.Bold);
        base.OnRenderItemText(e);
    }
}
```

To set the custom renderer to be the current renderer

1. To set the custom renderer for one `ToolStrip`, set the `ToolStrip.Renderer` property to the custom renderer.

```
toolStrip1.Renderer = New RedTextRenderer()
```

```
toolStrip1.Renderer = new RedTextRenderer();
```

2. Or to set the custom renderer for all [ToolStrip](#) classes contained in your application: Set the [ToolStripManager.Renderer](#) property to the custom renderer and set the [RenderMode](#) property to [ManagerRenderMode](#).

```
toolStrip1.RenderMode = ToolStripRenderMode.ManagerRenderMode  
ToolStripManager.Renderer = New RedTextRenderer()
```

```
toolStrip1.RenderMode = ToolStripRenderMode.ManagerRenderMode;  
ToolStripManager.Renderer = new RedTextRenderer();
```

See Also

- [Renderer](#)
- [ToolStripRenderer](#)
- [RenderMode](#)
- [ToolStrip Control Overview](#)
- [ToolStrip Control Architecture](#)
- [ToolStrip Technology Summary](#)

How to: Create Toggle Buttons in ToolStrip Controls

5/4/2018 • 1 min to read • [Edit Online](#)

When a user clicks a toggle button, it appears sunken and retains the sunken appearance until the user clicks the button again.

To create a toggling ToolStripButton

- Use code such as the following code example. This code assumes that your form contains a [ToolStrip](#) control, and that its [Items](#) collection contains a [ToolStripButton](#) called `toolStripButton1`. It also assumes that you have an event handler called `toolStripButton1_CheckedChanged`.

```
toolStripButton1.CheckOnClick = True  
toolStripButton1.CheckedChanged += new _  
EventHandler(toolStripButton1_CheckedChanged);
```

```
toolStripButton1.CheckOnClick = true;  
toolStripButton1.CheckedChanged += new _  
EventHandler(toolStripButton1_CheckedChanged);
```

See Also

[ToolStripButton](#)

[ToolStrip Control Overview](#)

How to: Custom Draw a ToolStrip Control

5/4/2018 • 1 min to read • [Edit Online](#)

The [ToolStrip](#) controls have the following associated rendering (painting) classes:

- [ToolStripSystemRenderer](#) provides the appearance and style of your operating system.
- [ToolStripProfessionalRenderer](#) provides the appearance and style of Microsoft Office.
- [ToolStripRenderer](#) is the abstract base class for the other two rendering classes.

To custom draw (also known as owner draw) a [ToolStrip](#), you can override one of the renderer classes and change an aspect of the rendering logic.

The following procedures describe various aspects of custom drawing.

To switch between the provided renderers

- Set the [RenderMode](#) property to the [ToolStripRenderMode](#) value you want.

With [ManagerRenderMode](#), the static [RenderMode](#) determines the renderer for your application. The other values of [ToolStripRenderMode](#) are [Custom](#), [Professional](#), and [System](#).

To change the Microsoft Office–style borders to straight

- Override [ToolStripProfessionalRenderer.OnRenderToolStripBorder](#), but do not call the base class.

NOTE

There is a version of this method for [ToolStripRenderer](#), [ToolStripSystemRenderer](#), and [ToolStripProfessionalRenderer](#).

To change the ProfessionalColorTable

- Override [ProfessionalColorTable](#) and change the colors you want.

```

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles Me.Load
    Dim t As MyColorTable = New MyColorTable
    ToolStrip1.Renderer = New ToolStripProfessionalRenderer(t)
End Sub

Class MyColorTable
    Inherits ProfessionalColorTable

    Public Overrides ReadOnly Property ButtonPressedGradientBegin() As Color
        Get
            Return Color.Red
        End Get
    End Property

    Public Overrides ReadOnly Property ButtonPressedGradientMiddle() _ 
As System.Drawing.Color
        Get
            Return Color.Blue
        End Get
    End Property

    Public Overrides ReadOnly Property ButtonPressedGradientEnd() _ 
As System.Drawing.Color
        Get
            Return Color.Green
        End Get
    End Property

    Public Overrides ReadOnly Property ButtonSelectedGradientBegin() _ 
As Color
        Get
            Return Color.Yellow
        End Get
    End Property

    Public Overrides ReadOnly Property ButtonSelectedGradientMiddle() As System.Drawing.Color
        Get
            Return Color.Orange
        End Get
    End Property

    Public Overrides ReadOnly Property ButtonSelectedGradientEnd() _ 
As System.Drawing.Color
        Get
            Return Color.Violet
        End Get
    End Property
End Class

```

To change the rendering for all ToolStrip controls in your application

1. Use the [ToolStripManager.RenderMode](#) property to choose one of the provided renderers.
2. Use [ToolStripManager.Renderer](#) to assign a custom renderer.
3. Ensure that [ToolStrip.RenderMode](#) is set to the default value of [ManagerRenderMode](#).

To turn off the Microsoft Office colors for the entire application

- Set [ToolStripManager.VisualStyleEnabled](#) to `false`.

To turn off the Microsoft Office colors for one ToolStrip control

- Use code similar to the following code example.

```
Dim colorTable As ProfessionalColorTable()
colorTable.UseSystemColors = True
Dim toolStrip.Renderer As ToolStripProfessionalRenderer(colorTable)
```

```
ProfessionalColorTable colorTable = new ProfessionalColorTable();
colorTable.UseSystemColors = true;
toolStrip.Renderer = new ToolStripProfessionalRenderer(colorTable);
```

See Also

[ToolStripSystemRenderer](#)

[ToolStripProfessionalRenderer](#)

[ToolStripRenderer](#)

[Controls with Built-In Owner-Drawing Support](#)

[How to: Create and Set a Custom Renderer for the ToolStrip Control in Windows Forms](#)

[ToolStrip Control Overview](#)

How to: Customize Colors in ToolStrip Applications

5/4/2018 • 2 min to read • [Edit Online](#)

You can customize the appearance of your [ToolStrip](#) by using the [ToolStripProfessionalRenderer](#) class to use customized colors.

Example

The following code example demonstrates how to use a [ToolStripProfessionalRenderer](#) to define custom colors at run time.

```
// This code example demonstrates how to use a ProfessionalRenderer
// to define custom professional colors at runtime.
class Form2 : Form
{
    public Form2()
    {
        // Create a new ToolStrip control.
        ToolStrip ts = new ToolStrip();

        // Populate the ToolStrip control.
        ts.Items.Add("Apples");
        ts.Items.Add("Oranges");
        ts.Items.Add("Pears");
        ts.Items.Add(
            "Change Colors",
            null,
            new EventHandler(ChangeColors_Click));

        // Create a new MenuStrip.
        MenuStrip ms = new MenuStrip();

        // Dock the MenuStrip control to the top of the form.
        ms.Dock = DockStyle.Top;

        // Add the top-level menu items.
        ms.Items.Add("File");
        ms.Items.Add("Edit");
        ms.Items.Add("View");
        ms.Items.Add("Window");

        // Add the ToolStrip to Controls collection.
        this.Controls.Add(ts);

        // Add the MenuStrip control last.
        // This is important for correct placement in the z-order.
        this.Controls.Add(ms);
    }

    // This event handler is invoked when the "Change colors"
    // ToolStripItem is clicked. It assigns the Renderer
    // property for the ToolStrip control.
    void ChangeColors_Click(object sender, EventArgs e)
    {
        ToolStripManager.Renderer =
            new ToolStripProfessionalRenderer(new CustomProfessionalColors());
    }
}

// This class defines the gradient colors for
// the MenuStrip and the ToolStrip
```

```
// The menustrip and the toolstrip.
class CustomProfessionalColors : ProfessionalColorTable
{
    public override Color ToolStripGradientBegin
    { get { return Color.BlueViolet; } }

    public override Color ToolStripGradientMiddle
    { get { return Color.CadetBlue; } }

    public override Color ToolStripGradientEnd
    { get { return Color.CornflowerBlue; } }

    public override Color MenuStripGradientBegin
    { get { return Color.Salmon; } }

    public override Color MenuStripGradientEnd
    { get { return Color.OrangeRed; } }
}
```

```
' This code example demonstrates how to use a ProfessionalRenderer
' to define custom professional colors at runtime.
Class Form2
    Inherits Form

    Public Sub New()
        ' Create a new ToolStrip control.
        Dim ts As New ToolStrip()

        ' Populate the ToolStrip control.
        ts.Items.Add("Apples")
        ts.Items.Add("Oranges")
        ts.Items.Add("Pears")
        ts.Items.Add("Change Colors", Nothing, New EventHandler(AddressOf ChangeColors_Click))

        ' Create a new MenuStrip.
        Dim ms As New MenuStrip()

        ' Dock the MenuStrip control to the top of the form.
        ms.Dock = DockStyle.Top

        ' Add the top-level menu items.
        ms.Items.Add("File")
        ms.Items.Add("Edit")
        ms.Items.Add("View")
        ms.Items.Add("Window")

        ' Add the ToolStrip to Controls collection.
        Me.Controls.Add(ts)

        ' Add the MenuStrip control last.
        ' This is important for correct placement in the z-order.
        Me.Controls.Add(ms)
    End Sub

    ' This event handler is invoked when the "Change colors"
    ' ToolStripItem is clicked. It assigns the Renderer
    ' property for the ToolStrip control.
    Sub ChangeColors_Click(ByVal sender As Object, ByVal e As EventArgs)
        ToolStripManager.Renderer = New ToolStripProfessionalRenderer(New CustomProfessionalColors())
    End Sub
End Class

' This class defines the gradient colors for
' the MenuStrip and the ToolStrip.
Class CustomProfessionalColors
    Inherits ProfessionalColorTable

    Public Overrides ReadOnly Property ToolStripGradientBegin() As Color
```

```

    Public Overrides ReadOnly Property ToolStripGradientBegin() As Color
        Get
            Return Color.BlueViolet
        End Get
    End Property

    Public Overrides ReadOnly Property ToolStripGradientMiddle() As Color
        Get
            Return Color.CadetBlue
        End Get
    End Property

    Public Overrides ReadOnly Property ToolStripGradientEnd() As Color
        Get
            Return Color.CornflowerBlue
        End Get
    End Property

    Public Overrides ReadOnly Property MenuStripGradientBegin() As Color
        Get
            Return Color.Salmon
        End Get
    End Property

    Public Overrides ReadOnly Property MenuStripGradientEnd() As Color
        Get
            Return Color.OrangeRed
        End Get
    End Property

End Class

```

Compiling the Code

This example requires:

- References to the System.Design, System.Drawing, and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[ToolStripManager](#)

[ProfessionalColorTable](#)

[MenuStrip](#)

[ToolStrip](#)

[ToolStripProfessionalRenderer](#)

How to: Define Z-Ordering of Docked ToolStrip Controls

5/4/2018 • 2 min to read • [Edit Online](#)

To position a [ToolStrip](#) control correctly with docking, you must position the control correctly in the form's z-order.

Example

The following code example demonstrates how to arrange a [ToolStrip](#) control and a docked [MenuStrip](#) control by specifying the z-order.

```
public Form2()
{
    // Create a new ToolStrip control.
    ToolStrip ts = new ToolStrip();

    // Populate the ToolStrip control.
    ts.Items.Add("Apples");
    ts.Items.Add("Oranges");
    ts.Items.Add("Pears");
    ts.Items.Add(
        "Change Colors",
        null,
        new EventHandler(ChangeColors_Click));

    // Create a new MenuStrip.
    MenuStrip ms = new MenuStrip();

    // Dock the MenuStrip control to the top of the form.
    ms.Dock = DockStyle.Top;

    // Add the top-level menu items.
    ms.Items.Add("File");
    ms.Items.Add("Edit");
    ms.Items.Add("View");
    ms.Items.Add("Window");

    // Add the ToolStrip to Controls collection.
    this.Controls.Add(ts);

    // Add the MenuStrip control last.
    // This is important for correct placement in the z-order.
    this.Controls.Add(ms);
}
```

```

Class Form2
    Inherits Form

    Public Sub New()
        ' Create a new ToolStrip control.
        Dim ts As New ToolStrip()

        ' Populate the ToolStrip control.
        ts.Items.Add("Apples")
        ts.Items.Add("Oranges")
        ts.Items.Add("Pears")
        ts.Items.Add("Change Colors", Nothing, New EventHandler(AddressOf ChangeColors_Click))

        ' Create a new MenuStrip.
        Dim ms As New MenuStrip()

        ' Dock the MenuStrip control to the top of the form.
        ms.Dock = DockStyle.Top

        ' Add the top-level menu items.
        ms.Items.Add("File")
        ms.Items.Add("Edit")
        ms.Items.Add("View")
        ms.Items.Add("Window")

        ' Add the ToolStrip to Controls collection.
        Me.Controls.Add(ts)

        ' Add the MenuStrip control last.
        ' This is important for correct placement in the z-order.
        Me.Controls.Add(ms)
    End Sub

    ' This event handler is invoked when the "Change colors"
    ' ToolStripItem is clicked. It assigns the Renderer
    ' property for the ToolStrip control.
    Sub ChangeColors_Click(ByVal sender As Object, ByVal e As EventArgs)
        ToolStripManager.Renderer = New ToolStripProfessionalRenderer(New CustomProfessionalColors())
    End Sub
End Class

```

The z-order is determined by the order in which the [ToolStrip](#) and [MenuStrip](#)

controls are added to the form's [Controls](#) collection.

```

// Add the ToolStrip to Controls collection.
this.Controls.Add(ts);

// Add the MenuStrip control last.
// This is important for correct placement in the z-order.
this.Controls.Add(ms);

```

```

' Add the ToolStrip to Controls collection.
Me.Controls.Add(ts)

' Add the MenuStrip control last.
' This is important for correct placement in the z-order.
Me.Controls.Add(ms)

```

Reverse the order of these calls to the [Add](#) method and view the effect on the layout.

Compiling the Code

This example requires:

- References to the System.Design, System.Drawing, and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[MenuStrip](#)

[ToolStrip](#)

[Add](#)

[Controls](#)

[Dock](#)

[ToolStrip Control](#)

How to: Detect When the Mouse Pointer Is Over a ToolStripItem

5/4/2018 • 1 min to read • [Edit Online](#)

Use the following procedure to detect when the mouse pointer is over a [ToolStripItem](#).

To detect when the pointer is over a ToolStripItem

- Use the [Selected](#) property for items in which [CanSelect](#) is `true`.

This will prevent you from having to synchronize the [MouseEnter](#) and [MouseLeave](#) events.

See Also

[ToolStripItem](#)

[Selected](#)

[ToolStrip Control Overview](#)

How to: Enable AutoComplete in ToolStrip Controls in Windows Forms

5/4/2018 • 1 min to read • [Edit Online](#)

The following procedure combines a [ToolStripLabel](#) with a [ToolStripComboBox](#) that can be dropped down to show a list of items, such as recently visited Web sites. If the user types a character that matches the first character of one of the items in the list, the item is immediately displayed.

NOTE

Automatic completion works with [ToolStrip](#) controls in the same way that it works with traditional controls such as [ComboBox](#) and [TextBox](#).

To enable AutoComplete in a ToolStrip control

1. Create a [ToolStrip](#) control and add items to it.

```
ToolStrip1 = New System.Windows.Forms.ToolStrip  
ToolStrip1.Items.AddRange(New System.Windows.Forms.ToolStripItem()_  
{ToolStripLabel1, ToolStripComboBox1})
```

```
toolStrip1 = new System.Windows.Forms.ToolStrip();  
toolStrip1.Items.AddRange(new System.Windows.Forms.ToolStripItem[]  
{toolStripLabel1, toolStripComboBox1});
```

2. Set the [Overflow](#) property of the label and the combo box to [Never](#) so that the list is always available regardless of the form's size.

```
ToolStripLabel1.Overflow = _  
    System.Windows.Forms.ToolStripItemOverflow.Never  
ToolStripComboBox1.Overflow = _  
    System.Windows.Forms.ToolStripItemOverflow.Never
```

```
toolStripLabel1.Overflow = _  
    System.Windows.Forms.ToolStripItemOverflow.Never  
toolStripComboBox1.Overflow = System.Windows.Forms.ToolStripItemOverflow.Never
```

3. Add words to the [Items](#) collection of the [ToolStripComboBox](#) control.

```
ToolStripComboBox1.Items.AddRange(New Object() {"First Item", _  
    "Second Item", "Third Item"})
```

```
toolStripComboBox1.Items.AddRange(new object[] {"First item", "Second item", "Third item"});
```

4. Set the [AutoCompleteMode](#) property of the combo box to [Append](#).

```
ToolStripComboBox1.AutoCompleteMode =  
    System.Windows.Forms.AutoCompleteMode.Append
```

```
toolStripComboBox1.AutoCompleteMode = System.Windows.Forms.AutoCompleteMode.Append;
```

5. Set the [AutoCompleteSource](#) property of the combo box to [ListItems](#).

```
ToolStripComboBox1.AutoCompleteSource =  
    System.Windows.Forms.AutoCompleteSource.ListItems
```

```
toolStripComboBox1.AutoCompleteSource = System.Windows.Forms.AutoCompleteSource.ListItems;
```

See Also

[ToolStrip](#)
[ToolStripLabel](#)
[ToolStripComboBox](#)
[AutoCompleteMode](#)
[AutoCompleteSource](#)
[ToolStrip Control Overview](#)
[ToolStrip Control Architecture](#)
[ToolStrip Technology Summary](#)

How to: Enable Reordering of ToolStrip Items at Run Time in Windows Forms

5/4/2018 • 1 min to read • [Edit Online](#)

You can enable the user to rearrange [ToolStripItem](#) controls on the [ToolStrip](#).

To enable ToolStripItem rearrangement at run time

- Set the [AllowItemReorder](#) property to `true`. By default, [AllowItemReorder](#) is `false`.

At run time, the user holds down the ALT key and the left mouse button to drag a [ToolStripItem](#) to a different location on the [ToolStrip](#).

```
toolStrip1.AllowItemReorder = True
```

```
toolStrip1.AllowItemReorder = true;
```

See Also

[ToolStrip](#)

[AllowItemReorder](#)

[ToolStrip Control Overview](#)

[ToolStrip Control Architecture](#)

[ToolStrip Technology Summary](#)

How to: Enable the TAB Key to Move Out of a ToolStrip Control

5/4/2018 • 1 min to read • [Edit Online](#)

Use the following procedure to enable the user to press the TAB key to move out of a [ToolStrip](#) to the next control in the tab order.

The [ToolStrip](#) accepts the first press of the TAB key, and the arrow keys select items within the [ToolStrip](#). When the user presses the TAB key a second time, it takes the user to the next control in the tab order.

To enable the user to press the TAB key to move out of a ToolStrip to the next control

- Set the [TabStop](#) property of the [ToolStrip](#) to `true`.

See Also

[ToolStrip](#)

[TabStop](#)

[ToolStrip Control Overview](#)

How to: Implement a Custom ToolStripRenderer

5/4/2018 • 20 min to read • [Edit Online](#)

You can customize the appearance of a [ToolStrip](#) control by implementing a class that derives from [ToolStripRenderer](#). This gives you the flexibility to create an appearance that differs from the appearance provided by the [ToolStripProfessionalRenderer](#) and [ToolStripSystemRenderer](#) classes.

Example

The following code example demonstrates how to implement a custom [ToolStripRenderer](#) class. In this example, the [GridStrip](#) control implements a sliding-tile puzzle, which allows the user to move tiles in a table layout to form an image. A custom [ToolStrip](#) control arranges its [ToolStripButton](#) controls in a grid layout. The layout contains one empty cell, into which the user can slide an adjacent tile by using a drag-and-drop operation. Tiles that the user can move are highlighted.

The [GridStripRenderer](#) class customizes three aspects of the [GridStrip](#) control's appearance:

- [GridStrip](#) border
- [ToolStripButton](#) border
- [ToolStripButton](#) image

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Drawing.Imaging;
using System.Text;
using System.Windows.Forms;
using System.Windows.Forms.Layout;

namespace GridStripLib
{
    // The following class implements a sliding-tile puzzle.
    // The GridStrip control is a custom ToolStrip that arranges
    // its ToolStripButton controls in a grid layout. There is
    // one empty cell, into which the user can slide an adjacent
    // tile with a drag-and-drop operation. Tiles that are eligible
    // for moving are highlighted.
    public class GridStrip : ToolStrip
    {
        // The button that is the drag source.
        private ToolStripButton dragButton = null;

        // Settings for the ToolStrip control's TableLayoutPanel.
        // This provides access to the cell position of each
        // ToolStripButton.
        private TableLayoutSettings tableSettings = null;

        // The empty cell. ToolStripButton controls that are
        // adjacent to this button can be moved to this button's
        // cell position.
        private ToolStripButton emptyCellButton = null;

        // The dimensions of each tile. A tile is represented
        // by a ToolStripButton controls.
        private Size tileSize = new Size(128, 128);
```

```

// The number of rows in the GridStrip control.
private readonly int rows = 5;

// The number of columns in the GridStrip control.
private readonly int columns = 5;

// The one-time initialization behavior is enforced
// with this field. For more information, see the
// OnPaint method.
private bool firstTime = false;

// This is required by the Windows Forms designer.
private System.ComponentModel.IContainer components;

// The default constructor.
public GridStrip()
{
    this.InitializeComponent();

    this.InitializeTableLayoutSettings();
}

// This property exposes the empty cell to the
// GridStripRenderer class.
internal ToolStripButton EmptyCell
{
    get
    {
        return this.emptyCellButton;
    }
}

// This utility method initializes the TableLayoutPanel
// which contains the ToolStripButton controls.
private void InitializeTableLayoutSettings()
{
    // Specify the numbers of rows and columns in the GridStrip control.
    this.tableSettings = base.LayoutSettings as TableLayoutSettings;
    this.tableSettings.ColumnCount = this.rows;
    this.tableSettings.RowCount = this.columns;

    // Create a dummy bitmap with the dimensions of each tile.
    // The GridStrip control sizes itself based on these dimensions.
    Bitmap b = new Bitmap(tileSize.Width, tileSize.Height);

    // Populate the GridStrip control with ToolStripButton controls.
    for (int i = 0; i < this.tableSettings.ColumnCount; i++)
    {
        for (int j = 0; j < this.tableSettings.RowCount; j++)
        {
            // Create a new ToolStripButton control.
            ToolStripButton btn = new ToolStripButton();
            btn.DisplayStyle = ToolStripItemDisplayStyle.Image;
            btn.Image = b;
            btn.ImageAlign = ContentAlignment.MiddleCenter;
            btn.ImageScaling = ToolStripItemImageScaling.None;
            btn.Margin = Padding.Empty;
            btn.Padding = Padding.Empty;

            // Add the new ToolStripButton control to the GridStrip.
            this.Items.Add(btn);

            // Set the cell position of the ToolStripButton control.
            TableLayoutPanelCellPosition cellPos = new TableLayoutPanelCellPosition(i, j);
            this.tableSettings.SetCellPosition(btn, cellPos);

            // If this is the ToolStripButton control at cell (0,0),
            // assign it as the empty cell button.
            if( i == 0 && j == 0 )

```

```

        {
            btn.Text = "Empty Cell";
            btn.Image = b;
            this.emptyCellButton = btn;
        }
    }
}

// This method defines the Paint event behavior.
// The GridStripRenderer requires that the GridStrip
// be fully layed out when it is renders, so this
// initialization code cannot be placed in the
// GridStrip constructor. By the time the Paint
// event is raised, the control layout has been
// completed, so the GridStripRenderer can paint
// correctly. This one-time initialization is
// implemented with the firstTime field.
protected override void OnPaint(PaintEventArgs e)
{
    base.OnPaint(e);

    if (!this.firstTime)
    {
        this.Renderer = new GridStripRenderer();

        // Comment this line to see the unscrambled image.
        this.ScrambleButtons();
        this.firstTime = true;
    }
}

// This utility method changes the ToolStripButton control
// positions in the TableLayoutPanel. This scrambles the
// buttons to initialize the puzzle.
private void ScrambleButtons()
{
    int i = 0;
    int lastElement = this.Items.Count - 1;

    while ( (i != lastElement) &&
           (lastElement - i > 1) )
    {
        TableLayoutPanelCellPosition pos1 =
            this.tableSettings.GetCellPosition(this.Items[i]);

        TableLayoutPanelCellPosition pos2 =
            this.tableSettings.GetCellPosition(this.Items[lastElement]);

        this.tableSettings.SetCellPosition(
            this.Items[i++],
            pos2);

        this.tableSettings.SetCellPosition(
            this.Items[lastElement--],
            pos1);
    }
}

// This method defines the MouseDown event behavior.
// If the user has clicked on a valid drag source,
// the drag operation starts.
protected override void OnMouseDown(MouseEventArgs mea)
{
    base.OnMouseDown(mea);

    ToolStripButton btn = this.GetItemAt(mea.Location) as ToolStripButton;
    if (btn != null)

```

```

        if (btn != null)
    {
        if (this.IsValidDragSource(btn))
        {
            this.dragButton = btn;
        }
    }

// This method defines the MouseMove event behavior.
protected override void OnMouseMove(MouseEventArgs mea)
{
    base.OnMouseMove(mea);

    // Is a drag operation pending?
    if (this.dragButton != null)
    {
        // A drag operation is pending. Call DoDragDrop to
        // determine the disposition of the operation.
        DragDropEffects dropEffect = this.DoDragDrop(
            new DataObject(this.dragButton),
            DragDropEffects.Move);
    }
}

// This method defines the DragOver event behavior.
protected override void OnDragOver(DragEventArgs dea)
{
    base.OnDragOver(dea);

    // Get the ToolStripButton control
    // at the given mouse position.
    Point p = new Point(dea.X, dea.Y);
    ToolStripButton item = this.GetItemAt(
        this.PointToClient(p)) as ToolStripButton;

    // If the ToolStripButton control is the empty cell,
    // indicate that the move operation is valid.
    if( item == this.emptyCellButton )
    {
        // Set the drag operation to indicate a valid move.
        dea.Effect = DragDropEffects.Move;
    }
}

// This method defines the DragDrop event behavior.
protected override void OnDragDrop(DragEventArgs dea)
{
    base.OnDragDrop(dea);

    // Did a valid move operation occur?
    if (dea.Effect == DragDropEffects.Move)
    {
        // The move operation is valid. Adjust the state
        // of the GridStrip control's TableLayoutPanel,
        // by swapping the positions of the source button
        // and the empty cell button.

        // Get the cell of the control to move.
        TableLayoutPanelCellPosition sourcePos =
            tableSettings.GetCellPosition(this.dragButton);

        // Get the cell of the emptyCellButton.
        TableLayoutPanelCellPosition dropPos =
            tableSettings.GetCellPosition(this.emptyCellButton);

        // Move the control to the empty cell.
        tableSettings.SetCellPosition(this.dragButton, dropPos);
    }
}

```

```

        // Set the position of the empty cell to
        // that of the previously occupied cell.
        tableSettings.SetCellPosition(this.emptyCellButton, sourcePos);

        // Reset the drag operation.
        this.dragButton = null;
    }
}

// This method defines the DragLeave event behavior.
// If the mouse leaves the client area of the GridStrip
// control, the drag operation is canceled.
protected override void OnDragLeave(EventArgs e)
{
    base.OnDragLeave(e);

    // Reset the drag operation.
    this.dragButton = null;
}

// This method defines the veryContinueDrag event behavior.
// If the mouse leaves the client area of the GridStrip
// control, the drag operation is canceled.
protected override void OnQueryContinueDrag(QueryContinueDragEventArgs qcdevent)
{
    base.OnQueryContinueDrag(qcdevent);

    // Get the current mouse position, in screen coordinates.
    Point mousePos = this.PointToClient(Control.MousePosition);

    // If the mouse position is outside the GridStrip control's
    // client area, cancel the drag operation. Be sure to
    // transform the mouse's screen coordinates to client coordinates.
    if (!this.ClientRectangle.Contains(mousePos))
    {
        qcdevent.Action = DragAction.Cancel;
    }
}

// This utility method determines if a button
// is positioned relative to the empty cell
// such that it can be dragged into the empty cell.
private bool IsValidDragSource(ToolStripButton b)
{
    TableLayoutPanelCellPosition sourcePos =
        tableSettings.GetCellPosition(b);

    TableLayoutPanelCellPosition emptyPos =
        tableSettings.GetCellPosition(this.emptyCellButton);

    return (IsValidDragSource(sourcePos, emptyPos));
}

// This utility method determines if a cell position
// is adjacent to the empty cell.
internal static bool IsValidDragSource(
    TableLayoutPanelCellPosition sourcePos,
    TableLayoutPanelCellPosition emptyPos)
{
    bool returnValue = false;

    // A cell is considered to be a valid drag source if it
    // is adjacent to the empty cell. Cells that are positioned
    // on a diagonal are not valid.
    if (((sourcePos.Column == emptyPos.Column - 1) && (sourcePos.Row == emptyPos.Row)) ||
        ((sourcePos.Column == emptyPos.Column + 1) && (sourcePos.Row == emptyPos.Row)) ||
        ((sourcePos.Column == emptyPos.Column) && (sourcePos.Row == emptyPos.Row - 1)) ||
        ((sourcePos.Column == emptyPos.Column) && (sourcePos.Row == emptyPos.Row + 1)))
    {

```

```

        returnValue = true;
    }

    return returnValue;
}

// This class implements a custom ToolStripRenderer for the
// GridStrip control. It customizes three aspects of the
// GridStrip control's appearance: GridStrip border,
// ToolStripButton border, and ToolStripButton image.
internal class GridStripRenderer : ToolStripRenderer
{
    // The style of the empty cell's text.
    private static StringFormat style = new StringFormat();

    // The thickness (width or height) of a
    // ToolStripButton control's border.
    static int borderThickness = 2;

    // The main bitmap that is the source for the
    // subimages that are assigned to individual
    // ToolStripButton controls.
    private Bitmap bmp = null;

    // The brush that paints the background of
    // the GridStrip control.
    private Brush backgroundBrush = null;

    // This is the static constructor. It initializes the
    // StringFormat for drawing the text in the empty cell.
    static GridStripRenderer()
    {
        style.Alignment = StringAlignment.Center;
        style.LineAlignment = StringAlignment.Center;
    }

    // This method initializes the GridStripRenderer by
    // creating the image that is used as the source for
    // the individual button images.
    protected override void Initialize(ToolStrip ts)
    {
        base.Initialize(ts);

        this.InitializeBitmap(ts);
    }

    // This method initializes an individual ToolStripButton
    // control. It copies a subimage from the GridStripRenderer's
    // main image, according to the position and size of
    // the ToolStripButton.
    protected override void InitializeItem(ToolStripItem item)
    {
        base.InitializeItem(item);

        GridStrip gs = item.Owner as GridStrip;

        // The empty cell does not receive a subimage.
        if ((item is ToolStripButton) &&
            (item != gs.EmptyCell))
        {
            // Copy the subimage from the appropriate
            // part of the main image.
            Bitmap subImage = bmp.Clone(
                item.Bounds,
                PixelFormat.Undefined);

            // Assign the subimage to the ToolStripButton
            // control's Image property.
            item.Image = subImage;
        }
    }
}

```

```

        }

    }

    // This utility method creates the main image that
    // is the source for the subimages of the individual
    // ToolStripButton controls.
    private void InitializeBitmap(ToolStrip toolStrip)
    {
        // Create the main bitmap, into which the image is drawn.
        this.bmp = new Bitmap(
            toolStrip.Size.Width,
            toolStrip.Size.Height);

        // Draw a fancy pattern. This could be any image or drawing.
        using (Graphics g = Graphics.FromImage(bmp))
        {
            // Draw smoothed lines.
            g.SmoothingMode = SmoothingMode.AntiAlias;

            // Draw the image. In this case, it is
            // a number of concentric ellipses.
            for (int i = 0; i < toolStrip.Size.Width; i += 8)
            {
                g.DrawEllipse(Pens.Blue, 0, 0, i, i);
            }
        }
    }

    // This method draws a border around the GridStrip control.
    protected override void OnRenderToolStripBorder(
        ToolStripRenderEventArgs e)
    {
        base.OnRenderToolStripBorder(e);

        ControlPaint.DrawFocusRectangle(
            e.Graphics,
            e.AffectedBounds,
            SystemColors.ControlDarkDark,
            SystemColors.ControlDarkDark);
    }

    // This method renders the GridStrip control's background.
    protected override void OnRenderToolStripBackground(
        ToolStripRenderEventArgs e)
    {
        base.OnRenderToolStripBackground(e);

        // This late initialization is a workaround. The gradient
        // depends on the bounds of the GridStrip control. The bounds
        // are dependent on the layout engine, which hasn't fully
        // performed layout by the time the Initialize method runs.
        if (this.backgroundBrush == null)
        {
            this.backgroundBrush = new LinearGradientBrush(
                e.ToolStrip.ClientRectangle,
                SystemColors.ControlLightLight,
                SystemColors.ControlDark,
                90,
                true);
        }

        // Paint the GridStrip control's background.
        e.Graphics.FillRectangle(
            this.backgroundBrush,
            e.AffectedBounds);
    }

    // This method draws a border around the button's image. If the background
    // to be rendered belongs to the empty cell, a string is drawn. Otherwise,

```

```

// a border is drawn at the edges of the button.
protected override void OnRenderButtonBackground(
    ToolStripItemEventArgs e)
{
    base.OnRenderButtonBackground(e);

    // Define some local variables for convenience.
    Graphics g = e.Graphics;
    GridStrip gs = e.ToolStrip as GridStrip;
    ToolStripButton gsb = e.Item as ToolStripButton;

    // Calculate the rectangle around which the border is painted.
    Rectangle imageRectangle = new Rectangle(
        borderThickness,
        borderThickness,
        e.Item.Width - 2 * borderThickness,
        e.Item.Height - 2 * borderThickness);

    // If rendering the empty cell background, draw an
    // explanatory string, centered in the ToolStripButton.
    if (gsb == gs.EmptyCell)
    {
        e.Graphics.DrawString(
            "Drag to here",
            gsb.Font,
            SystemBrushes.ControlDarkDark,
            imageRectangle, style);
    }
    else
    {
        // If the button can be a drag source, paint its border red.
        // otherwise, paint its border a dark color.
        Brush b = gs.IsValidDragSource(gsb) ? b =
            Brushes.Red : SystemBrushes.ControlDarkDark;

        // Draw the top segment of the border.
        Rectangle borderSegment = new Rectangle(
            0,
            0,
            e.Item.Width,
            imageRectangle.Top);
        g.FillRectangle(b, borderSegment);

        // Draw the right segment.
        borderSegment = new Rectangle(
            imageRectangle.Right,
            0,
            e.Item.Bounds.Right - imageRectangle.Right,
            imageRectangle.Bottom);
        g.FillRectangle(b, borderSegment);

        // Draw the left segment.
        borderSegment = new Rectangle(
            0,
            0,
            imageRectangle.Left,
            e.Item.Height);
        g.FillRectangle(b, borderSegment);

        // Draw the bottom segment.
        borderSegment = new Rectangle(
            0,
            imageRectangle.Bottom,
            e.Item.Width,
            e.Item.Bounds.Bottom - imageRectangle.Bottom);
        g.FillRectangle(b, borderSegment);
    }
}

```

```

        ' Windows Forms Designer generated code

    private void InitializeComponent()
    {
        this.components = new System.ComponentModel.Container();
        this.SuspendLayout();
        //
        // GridStrip
        //
        this.AllowDrop = true;
        this.BackgroundImageLayout = System.Windows.Forms.ImageLayout.None;
        this.CanOverflow = false;
        this.Dock = System.Windows.Forms.DockStyle.None;
        this.GripStyle = System.Windows.Forms.ToolStripGripStyle.Hidden;
        this.LayoutStyle = System.Windows.Forms.TableLayoutPanelStyle.Table;
        this.ResumeLayout(false);
    }

    #endregion

}

}

```

```

Imports System
Imports System.Collections.Generic
Imports System.Drawing
Imports System.Drawing.Drawing2D
Imports System.Drawing.Imaging
Imports System.Text
Imports System.Windows.Forms
Imports System.Windows.Forms.Layout

' The following class implements a sliding-tile puzzle.
' The GridStrip control is a custom ToolStrip that arranges
' its ToolStripButton controls in a grid layout. There is
' one empty cell, into which the user can slide an adjacent
' tile with a drag-and-drop operation. Tiles that are eligible
' for moving are highlighted.
Public Class GridStrip
    Inherits ToolStrip

    ' The button that is the drag source.
    Private dragButton As ToolStripButton = Nothing

    ' Settings for the ToolStrip control's TableLayoutPanel.
    ' This provides access to the cell position of each
    ' ToolStripButton.
    Private tableSettings As TableLayoutSettings = Nothing

    ' The empty cell. ToolStripButton controls that are
    ' adjacent to this button can be moved to this button's
    ' cell position.
    Private emptyCellButton As ToolStripButton = Nothing

    ' The dimensions of each tile. A tile is represented
    ' by a ToolStripButton controls.
    Private tileSize As New Size(128, 128)

    ' The number of rows in the GridStrip control.
    Private rows As Integer = 5

    ' The number of columns in the GridStrip control.
    Private columns As Integer = 5

```

```

' The one-time initialization behavior is enforced
' with this field. For more information, see the
' OnPaint method.
Private firstTime As Boolean = False

' This is a required by the Windows Forms designer.
Private components As System.ComponentModel.IContainer

' The default constructor.
Public Sub New()
    MyBase.New()

    Me.InitializeComponent()

    Me.InitializeTableLayoutSettings()
End Sub

' This property exposes the empty cell to the
' GridStripRenderer class.
Friend ReadOnly Property EmptyCell() As ToolStripButton
    Get
        Return Me.emptyCellButton
    End Get
End Property
End Property

' This utility method initializes the TableLayoutPanel
' which contains the ToolStripButton controls.
Private Sub InitializeTableLayoutSettings()

    ' Specify the numbers of rows and columns in the GridStrip control.
    Me.tableSettings = CType(MyBase.LayoutSettings, TableLayoutSettings)
    Me.tableSettings.ColumnCount = Me.rows
    Me.tableSettings.RowCount = Me.columns

    ' Create a dummy bitmap with the dimensions of each tile.
    ' The GridStrip control sizes itself based on these dimensions.
    Dim b As New Bitmap(tileSize.Width, tileSize.Height)

    ' Populate the GridStrip control with ToolStripButton controls.
    Dim i As Integer
    For i = 0 To (Me.tableSettings.ColumnCount) - 1
        Dim j As Integer
        For j = 0 To (Me.tableSettings.RowCount) - 1
            ' Create a new ToolStripButton control.
            Dim btn As New ToolStripButton()
            btn.DisplayStyle = ToolStripItemDisplayStyle.Image
            btn.Image = b
            btn.ImageAlign = ContentAlignment.MiddleCenter
            btn.ImageScaling = ToolStripItemImageScaling.None
            btn.Margin = System.Windows.Forms.Padding.Empty
            btn.Padding = System.Windows.Forms.Padding.Empty

            ' Add the new ToolStripButton control to the GridStrip.
            Me.Items.Add(btn)

            ' Set the cell position of the ToolStripButton control.
            Dim cellPos As New TableLayoutPanelCellPosition(i, j)
            Me.tableSettings.SetCellPosition(btn, cellPos)

            ' If this is the ToolStripButton control at cell (0,0),
            ' assign it as the empty cell button.
            If i = 0 AndAlso j = 0 Then
                btn.Text = "Empty Cell"
                btn.Image = b
                Me.emptyCellButton = btn
            End If
        Next j
    Next i
End Sub

```

```

        Next j
    Next i
End Sub

' This method defines the Paint event behavior.
' The GridStripRenderer requires that the GridStrip
' be fully layed out when it is renders, so this
' initialization code cannot be placed in the
' GridStrip constructor. By the time the Paint
' event is raised, the control layout has been
' completed, so the GridStripRenderer can paint
' correctly. This one-time initialization is
' implemented with the firstTime field.
Protected Overrides Sub OnPaint(e As PaintEventArgs)
    MyBase.OnPaint(e)

    If Not Me.firstTime Then
        Me.Renderer = New GridStripRenderer()

        ' Comment this line to see the unscrambled image.
        Me.ScrambleButtons()
        Me.firstTime = True
    End If
End Sub

' This utility method changes the ToolStripButton control
' positions in the TableLayoutPanel. This scrambles the
' buttons to initialize the puzzle.
Private Sub ScrambleButtons()
    Dim i As Integer = 0
    Dim lastElement As Integer = Me.Items.Count - 1

    While i <> lastElement AndAlso lastElement - i > 1
        Dim pos1 As TableLayoutPanelCellPosition = _
            Me.tableSettings.GetCellPosition(Me.Items(i))

        Dim pos2 As TableLayoutPanelCellPosition = _
            Me.tableSettings.GetCellPosition(Me.Items(lastElement))

        Me.tableSettings.SetCellPosition(Me.Items(i), pos2)
        i += 1

        Me.tableSettings.SetCellPosition(Me.Items(lastElement), pos1)
        lastElement -= 1
    End While
End Sub

' This method defines the MouseDown event behavior.
' If the user has clicked on a valid drag source,
' the drag operation starts.
Protected Overrides Sub OnMouseDown(mea As MouseEventArgs)
    MyBase.OnMouseDown(mea)

    Dim btn As ToolStripButton = CType(Me.GetItemAt(mea.Location), ToolStripButton)

    If (btn IsNot Nothing) Then
        If Me.IsValidDragSource(btn) Then
            Me.dragButton = btn
        End If
    End If
End Sub

' This method defines the MouseMove event behavior.
Protected Overrides Sub OnMouseMove(mea As MouseEventArgs)
    MyBase.OnMouseMove(mea)

```

```

 MyBase.OnMouseMove(m)
 
    ' Is a drag operation pending?
    If (Me.dragButton IsNot Nothing) Then
        ' A drag operation is pending. Call DoDragDrop to
        ' determine the disposition of the operation.
        Dim dropEffect As DragDropEffects = Me.DoDragDrop(New DataObject(Me.dragButton),
DragDropEffects.Move)
    End If
End Sub

' This method defines the DragOver event behavior.
Protected Overrides Sub OnDragOver(dea As DragEventArgs)
    MyBase.OnDragOver(dea)

    ' Get the ToolStripButton control
    ' at the given mouse position.
    Dim p As New Point(dea.X, dea.Y)
    Dim item As ToolStripButton = CType(Me.GetItemAt(Me.PointToClient(p)), ToolStripButton)

    ' If the ToolStripButton control is the empty cell,
    ' indicate that the move operation is valid.
    If item Is Me.emptyCellButton Then
        ' Set the drag operation to indicate a valid move.
        dea.Effect = DragDropEffects.Move
    End If
End Sub

' This method defines the DragDrop event behavior.
Protected Overrides Sub OnDragDrop(dea As DragEventArgs)
    MyBase.OnDragDrop(dea)

    ' Did a valid move operation occur?
    If dea.Effect = DragDropEffects.Move Then
        ' The move operation is valid. Adjust the state
        ' of the GridStrip control's TableLayoutPanel,
        ' by swapping the positions of the source button
        ' and the empty cell button.
        ' Get the cell of the control to move.
        Dim sourcePos As TableLayoutPanelCellPosition = tableSettings.GetCellPosition(Me.dragButton)

        ' Get the cell of the emptyCellButton.
        Dim dropPos As TableLayoutPanelCellPosition = tableSettings.GetCellPosition(Me.emptyCellButton)

        ' Move the control to the empty cell.
        tableSettings.SetCellPosition(Me.dragButton, dropPos)

        ' Set the position of the empty cell to
        ' that of the previously occupied cell.
        tableSettings.SetCellPosition(Me.emptyCellButton, sourcePos)

        ' Reset the drag operation.
        Me.dragButton = Nothing
    End If
End Sub

' This method defines the DragLeave event behavior.
' If the mouse leaves the client area of the GridStrip
' control, the drag operation is canceled.
Protected Overrides Sub OnDragLeave(e As EventArgs)
    MyBase.OnDragLeave(e)

    ' Reset the drag operation.
    Me.dragButton = Nothing
End Sub

```

```

' This method defines the veryContinueDrag event behavior.
' If the mouse leaves the client area of the GridStrip
' control, the drag operation is canceled.
Protected Overrides Sub OnQueryContinueDrag(qcdevent As QueryContinueDragEventArgs)
    MyBase.OnQueryContinueDrag(qcdevent)

    ' Get the current mouse position, in screen coordinates.
    Dim mousePos As Point = Me.PointToClient(Control.MousePosition)

    ' If the mouse position is outside the GridStrip control's
    ' client area, cancel the drag operation. Be sure to
    ' transform the mouse's screen coordinates to client coordinates.
    If Not Me.ClientRectangle.Contains(mousePos) Then
        qcdevent.Action = DragAction.Cancel
    End If
End Sub

' This utility method determines if a button
' is positioned relative to the empty cell
' such that it can be dragged into the empty cell.
Overloads Private Function IsValidDragSource(b As ToolStripButton) As Boolean
    Dim sourcePos As TableLayoutPanelCellPosition = tableSettings.GetCellPosition(b)

    Dim emptyPos As TableLayoutPanelCellPosition = tableSettings.GetCellPosition(Me.emptyCellButton)

    Return IsValidDragSource(sourcePos, emptyPos)

End Function

' This utility method determines if a cell position
' is adjacent to the empty cell.
Friend Overloads Shared Function IsValidDragSource( _
    ByVal sourcePos As TableLayoutPanelCellPosition, _
    ByVal emptyPos As TableLayoutPanelCellPosition) As Boolean
    Dim returnValue As Boolean = False

    ' A cell is considered to be a valid drag source if it
    ' is adjacent to the empty cell. Cells that are positioned
    ' on a diagonal are not valid.
    If sourcePos.Column = emptyPos.Column - 1 AndAlso sourcePos.Row = emptyPos.Row OrElse _
        (sourcePos.Column = emptyPos.Column + 1 AndAlso sourcePos.Row = emptyPos.Row) OrElse _
        (sourcePos.Column = emptyPos.Column AndAlso sourcePos.Row = emptyPos.Row - 1) OrElse _
        (sourcePos.Column = emptyPos.Column AndAlso sourcePos.Row = emptyPos.Row + 1) Then
        returnValue = True
    End If

    Return returnValue
End Function

' This class implements a custom ToolStripRenderer for the
' GridStrip control. It customizes three aspects of the
' GridStrip control's appearance: GridStrip border,
' ToolStripButton border, and ToolStripButton image.
Friend Class GridStripRenderer
    Inherits ToolStripRenderer

    ' The style of the empty cell's text.
    Private Shared style As New StringFormat()

    ' The thickness (width or height) of a
    ' ToolStripButton control's border.
    Private Shared borderThickness As Integer = 2

    ' The main bitmap that is the source for the
    ' subimages that are assigned to individual
    ' ToolStripButton controls.

```

```

Private bmp As Bitmap = Nothing

' The brush that paints the background of
' the GridStrip control.
Private backgroundBrush As Brush = Nothing

' This is the static constructor. It initializes the
' StringFormat for drawing the text in the empty cell.
Shared Sub New()
    style.Alignment = StringAlignment.Center
    style.LineAlignment = StringAlignment.Center
End Sub

' This method initializes the GridStripRenderer by
' creating the image that is used as the source for
' the individual button images.
Protected Overrides Sub Initialize(ts As ToolStrip)
    MyBase.Initialize(ts)

    Me.InitializeBitmap(ts)
End Sub

' This method initializes an individual ToolStripButton
' control. It copies a subimage from the GridStripRenderer's
' main image, according to the position and size of
' the ToolStripButton.
Protected Overrides Sub InitializeItem(item As ToolStripItem)
    MyBase.InitializeItem(item)

    Dim gs As GridStrip = item.Owner

    ' The empty cell does not receive a subimage.
    If ((TypeOf (item) Is ToolStripButton) And _
        (item IsNot gs.EmptyCell)) Then
        ' Copy the subimage from the appropriate
        ' part of the main image.
        Dim subImage As Bitmap = bmp.Clone(item.Bounds, PixelFormat.Undefined)

        ' Assign the subimage to the ToolStripButton
        ' control's Image property.
        item.Image = subImage
    End If
End Sub

' This utility method creates the main image that
' is the source for the subimages of the individual
' ToolStripButton controls.
Private Sub InitializeBitmap(toolStrip As ToolStrip)
    ' Create the main bitmap, into which the image is drawn.
    Me.bmp = New Bitmap(toolStrip.Size.Width, toolStrip.Size.Height)

    ' Draw a fancy pattern. This could be any image or drawing.
    Dim g As Graphics = Graphics.FromImage(bmp)
    Try
        ' Draw smoothed lines.
        g.SmoothingMode = SmoothingMode.AntiAlias

        ' Draw the image. In this case, it is
        ' a number of concentric ellipses.
        Dim i As Integer
        For i = 0 To toolStrip.Size.Width - 8 Step 8
            g.DrawEllipse(Pens.Blue, 0, 0, i, i)
        Next i
    Finally
        g.Dispose()
    End Try
End Sub

```

```

' This method draws a border around the GridStrip control.
Protected Overrides Sub OnRenderToolStripBorder(e As ToolStripRenderEventArgs)
    MyBase.OnRenderToolStripBorder(e)

    ControlPaint.DrawFocusRectangle(e.Graphics, e.AffectedBounds, SystemColors.ControlDarkDark,
SystemColors.ControlDarkDark)
End Sub

' This method renders the GridStrip control's background.
Protected Overrides Sub OnRenderToolStripBackground(e As ToolStripRenderEventArgs)
    MyBase.OnRenderToolStripBackground(e)

    ' This late initialization is a workaround. The gradient
    ' depends on the bounds of the GridStrip control. The bounds
    ' are dependent on the layout engine, which hasn't fully
    ' performed layout by the time the Initialize method runs.
    If Me.backgroundBrush Is Nothing Then
        Me.backgroundBrush = New LinearGradientBrush(e.ToolStrip.ClientRectangle,
SystemColors.ControlLightLight, SystemColors.ControlDark, 90, True)
    End If

    ' Paint the GridStrip control's background.
    e.Graphics.FillRectangle(Me.backgroundBrush, e.AffectedBounds)
End Sub

' This method draws a border around the button's image. If the background
' to be rendered belongs to the empty cell, a string is drawn. Otherwise,
' a border is drawn at the edges of the button.
Protected Overrides Sub OnRenderButtonBackground(e As ToolStripItemRenderEventArgs)
    MyBase.OnRenderButtonBackground(e)

    ' Define some local variables for convenience.
    Dim g As Graphics = e.Graphics
    Dim gs As GridStrip = e.ToolStrip
    Dim gsb As ToolStripButton = e.Item

    ' Calculate the rectangle around which the border is painted.
    Dim imageRectangle As New Rectangle(borderThickness, borderThickness, e.Item.Width - 2 *
borderThickness, e.Item.Height - 2 * borderThickness)

    ' If rendering the empty cell background, draw an
    ' explanatory string, centered in the ToolStripButton.
    If gsb Is gs.EmptyCell Then
        e.Graphics.DrawString("Drag to here", gsb.Font, SystemBrushes.ControlDarkDark, imageRectangle,
style)
    Else
        ' If the button can be a drag source, paint its border red.
        ' otherwise, paint its border a dark color.
        Dim b As Brush = IIf(gs.IsValidDragSource(gsb), Brushes.Red, SystemBrushes.ControlDarkDark)

        ' Draw the top segment of the border.
        Dim borderSegment As New Rectangle(0, 0, e.Item.Width, imageRectangle.Top)
        g.FillRectangle(b, borderSegment)

        ' Draw the right segment.
        borderSegment = New Rectangle(imageRectangle.Right, 0, e.Item.Bounds.Right -
imageRectangle.Right, imageRectangle.Bottom)
        g.FillRectangle(b, borderSegment)

        ' Draw the left segment.
        borderSegment = New Rectangle(0, 0, imageRectangle.Left, e.Item.Height)
        g.FillRectangle(b, borderSegment)

        ' Draw the bottom segment.
        borderSegment = New Rectangle(0, imageRectangle.Bottom, e.Item.Width, e.Item.Bounds.Bottom -
imageRectangle.Bottom)
        g.FillRectangle(b, borderSegment)
    End If
End Sub

```

```

End Class

#Region "Windows Forms Designer generated code"

Private Sub InitializeComponent()
    Me.components = New System.ComponentModel.Container()
    Me.SuspendLayout()
    '
    ' ToolStrip
    '
    Me.AllowDrop = True
    Me.BackgroundImageLayout = System.Windows.Forms.ImageLayout.None
    Me.CanOverflow = False
    Me.Dock = System.Windows.Forms.DockStyle.None
    Me.GripStyle = System.Windows.Forms.ToolStripGripStyle.Hidden
    Me.LayoutStyle = System.Windows.Forms.ToolStripLayoutStyle.Table
    Me.ResumeLayout(False)
End Sub 'InitializeComponent

#End Region

End Class

```

Compiling the Code

This example requires:

- References to the `System.Drawing` and `System.Windows.Forms` assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[MenuStrip](#)

[ToolStrip](#)

[ToolStripRenderer](#)

[ToolStripProfessionalRenderer](#)

[ToolStripSystemRenderer](#)

[StatusStrip](#)

[ToolStrip Control](#)

How to: Manage ToolStrip Overflow in Windows Forms

5/4/2018 • 1 min to read • [Edit Online](#)

When all the items on a [ToolStrip](#) control do not fit in the allotted space, you can enable overflow functionality on the [ToolStrip](#) and determine the overflow behavior of specific [ToolStripItems](#).

When you add [ToolStripItems](#) that require more space than is allotted to the [ToolStrip](#) given the form's current size, a [ToolStripOverflowButton](#) automatically appears on the [ToolStrip](#). The [ToolStripOverflowButton](#) appears, and overflow-enabled items are moved into the drop-down overflow menu. This allows you to customize and prioritize how your [ToolStrip](#) items properly adjust to different form sizes. You can also change the appearance of your items when they fall into the overflow by using the [Placement](#) and [ToolStripOverflow.DisplayedItems](#) properties and the [LayoutCompleted](#) event. If you enlarge the form at either design time or run time, more [ToolStripItems](#) can be displayed on the main [ToolStrip](#) and the [ToolStripOverflowButton](#) might even disappear until you decrease the size of the form.

To enable overflow on a ToolStrip control

- Ensure that the [CanOverflow](#) property is not set to `false` for the [ToolStrip](#). The default is `True`.

When [CanOverflow](#) is `True` (the default), a [ToolStripItem](#) is sent to the drop-down overflow menu when the content of the [ToolStripItem](#) exceeds the width of a horizontal [ToolStrip](#) or the height of a vertical [ToolStrip](#).

To specify overflow behavior of a specific ToolStripItem

- Set the [Overflow](#) property of the [ToolStripItem](#) to the desired value. The possibilities are `Always`, `Never`, and `AsNeeded`. The default is `AsNeeded`.

```
toolStripTextBox1.Overflow = _  
System.Windows.Forms.ToolStripItemOverflow.Never
```

```
toolStripTextBox1.Overflow = _  
System.Windows.Forms.ToolStripItemOverflow.Never;
```

See Also

- [ToolStrip](#)
- [ToolStripOverflowButton](#)
- [Overflow](#)
- [CanOverflow](#)
- [ToolStrip Control Overview](#)
- [ToolStrip Control Architecture](#)
- [ToolStrip Technology Summary](#)

How to: Move a ToolStrip Out of a ToolStripContainer onto a Form

5/4/2018 • 1 min to read • [Edit Online](#)

Use the following procedure to move a [ToolStrip](#) out of a [ToolStripContainer](#) onto a form.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To move a ToolStrip out of a ToolStripContainer onto a form

1. Select the [ToolStrip](#).
2. Cut the [ToolStrip](#) by pressing CTRL+X, or right-click the [ToolStrip](#) and choose **Cut** from the context menu.
3. Select the form.
4. Paste the [ToolStrip](#) by pressing CTRL+V, or choose **Paste** from the **Edit** menu.
5. Set the [Dock](#) property of the [ToolStrip](#) to **Top**.

See Also

[ToolStrip](#)

[ToolStripContainer](#)

[ToolStrip Control Overview](#)

How to: Position a ToolStripItem on a ToolStrip

5/4/2018 • 1 min to read • [Edit Online](#)

You can move or add a [ToolStripItem](#) to the left or right side of a [ToolStrip](#).

To move or add a ToolStripItem to the left side of a ToolStrip

1. Set the [ToolStripItemAlignment](#) property of the [ToolStripItem](#) to [Left](#).

To move or add a ToolStripItem to the right side of a ToolStrip

1. Set the [ToolStripItemAlignment](#) property of the [ToolStripItem](#) to [Right](#).

See Also

[ToolStripItem](#)

[ToolStripItemAlignment](#)

[Left](#)

[Right](#)

[ToolStrip Control Overview](#)

How to: Set the ToolStrip Renderer at Run Time

5/4/2018 • 4 min to read • [Edit Online](#)

You can customize the appearance of your [ToolStrip](#) control by creating a custom [ProfessionalColorTable](#) class.

Example

The following code example demonstrates how to create a custom [ProfessionalColorTable](#) class. This class defines gradients for a [MenuStrip](#) and a [ToolStrip](#) control.

To use this code example, compile and run the application, and then click **Change Colors** to apply the gradients defined in the custom [ProfessionalColorTable](#) class.

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Drawing;
```

```
Imports System
Imports System.Collections.Generic
Imports System.Windows.Forms
Imports System.Drawing
```

```
// This code example demonstrates how to use a ProfessionalRenderer
// to define custom professional colors at runtime.
class Form2 : Form
{
    public Form2()
    {
        // Create a new ToolStrip control.
        ToolStrip ts = new ToolStrip();

        // Populate the ToolStrip control.
        ts.Items.Add("Apples");
        ts.Items.Add("Oranges");
        ts.Items.Add("Pears");
        ts.Items.Add(
            "Change Colors",
            null,
            new EventHandler(ChangeColors_Click));

        // Create a new MenuStrip.
        MenuStrip ms = new MenuStrip();

        // Dock the MenuStrip control to the top of the form.
        ms.Dock = DockStyle.Top;

        // Add the top-level menu items.
        ms.Items.Add("File");
        ms.Items.Add("Edit");
        ms.Items.Add("View");
        ms.Items.Add("Window");

        // Add the ToolStrip to Controls collection.
        this.Controls.Add(ts);

        // Add the MenuStrip control last.
    }
}
```

```

    // This is important for correct placement in the z-order.
    this.Controls.Add(ms);
}

// This event handler is invoked when the "Change colors"
// ToolStripItem is clicked. It assigns the Renderer
// property for the ToolStrip control.
void ChangeColors_Click(object sender, EventArgs e)
{
    ToolStripManager.Renderer =
        new ToolStripProfessionalRenderer(new CustomProfessionalColors());
}
}

// This class defines the gradient colors for
// the MenuStrip and the ToolStrip.
class CustomProfessionalColors : ProfessionalColorTable
{
    public override Color ToolStripGradientBegin
    { get { return Color.BlueViolet; } }

    public override Color ToolStripGradientMiddle
    { get { return Color.CadetBlue; } }

    public override Color ToolStripGradientEnd
    { get { return Color.CornflowerBlue; } }

    public override Color MenuStripGradientBegin
    { get { return Color.Salmon; } }

    public override Color MenuStripGradientEnd
    { get { return Color.OrangeRed; } }
}

```

```

' This code example demonstrates how to use a ProfessionalRenderer
' to define custom professional colors at runtime.
Class Form2
    Inherits Form

    Public Sub New()
        ' Create a new ToolStrip control.
        Dim ts As New ToolStrip()

        ' Populate the ToolStrip control.
        ts.Items.Add("Apples")
        ts.Items.Add("Oranges")
        ts.Items.Add("Pears")
        ts.Items.Add("Change Colors", Nothing, New EventHandler(AddressOf ChangeColors_Click))

        ' Create a new MenuStrip.
        Dim ms As New MenuStrip()

        ' Dock the MenuStrip control to the top of the form.
        ms.Dock = DockStyle.Top

        ' Add the top-level menu items.
        ms.Items.Add("File")
        ms.Items.Add("Edit")
        ms.Items.Add("View")
        ms.Items.Add("Window")

        ' Add the ToolStrip to Controls collection.
        Me.Controls.Add(ts)

        ' Add the MenuStrip control last.
        ' This is important for correct placement in the z-order.
        Me.Controls.Add(ms)
    End Sub

```

```

End Sub

' This event handler is invoked when the "Change colors"
' ToolStripItem is clicked. It assigns the Renderer
' property for the ToolStrip control.
Sub ChangeColors_Click(ByVal sender As Object, ByVal e As EventArgs)
    ToolStripManager.Renderer = New ToolStripProfessionalRenderer(New CustomProfessionalColors())
End Sub
End Class

' This class defines the gradient colors for
' the MenuStrip and the ToolStrip.
Class CustomProfessionalColors
    Inherits ProfessionalColorTable

    Public Overrides ReadOnly Property ToolStripGradientBegin() As Color
        Get
            Return Color.BlueViolet
        End Get
    End Property

    Public Overrides ReadOnly Property ToolStripGradientMiddle() As Color
        Get
            Return Color.CadetBlue
        End Get
    End Property

    Public Overrides ReadOnly Property ToolStripGradientEnd() As Color
        Get
            Return Color.CornflowerBlue
        End Get
    End Property

    Public Overrides ReadOnly Property MenuStripGradientBegin() As Color
        Get
            Return Color.Salmon
        End Get
    End Property

    Public Overrides ReadOnly Property MenuStripGradientEnd() As Color
        Get
            Return Color.OrangeRed
        End Get
    End Property
End Class

```

Defining a Custom ProfessionalColorTable class

The custom gradients are defined in the `CustomProfessionalColors` class.

```

// This class defines the gradient colors for
// the MenuStrip and the ToolStrip.
class CustomProfessionalColors : ProfessionalColorTable
{
    public override Color ToolStripGradientBegin
    { get { return Color.BlueViolet; } }

    public override Color ToolStripGradientMiddle
    { get { return Color.CadetBlue; } }

    public override Color ToolStripGradientEnd
    { get { return Color.CornflowerBlue; } }

    public override Color MenuStripGradientBegin
    { get { return Color.Salmon; } }

    public override Color MenuStripGradientEnd
    { get { return Color.OrangeRed; } }
}

```

```

' This class defines the gradient colors for
' the MenuStrip and the ToolStrip.
Class CustomProfessionalColors
    Inherits ProfessionalColorTable

    Public Overrides ReadOnly Property ToolStripGradientBegin() As Color
        Get
            Return Color.BlueViolet
        End Get
    End Property

    Public Overrides ReadOnly Property ToolStripGradientMiddle() As Color
        Get
            Return Color.CadetBlue
        End Get
    End Property

    Public Overrides ReadOnly Property ToolStripGradientEnd() As Color
        Get
            Return Color.CornflowerBlue
        End Get
    End Property

    Public Overrides ReadOnly Property MenuStripGradientBegin() As Color
        Get
            Return Color.Salmon
        End Get
    End Property

    Public Overrides ReadOnly Property MenuStripGradientEnd() As Color
        Get
            Return Color.OrangeRed
        End Get
    End Property

End Class

```

Assigning a Custom Renderer

Create a new `ToolStripProfessionalRenderer` with a `CustomProfessionalColors` class, and assign it to the `ToolStripManager.Renderer` property.

```
// This event handler is invoked when the "Change colors"
// ToolStripItem is clicked. It assigns the Renderer
// property for the ToolStrip control.
void ChangeColors_Click(object sender, EventArgs e)
{
    ToolStripManager.Renderer =
        new ToolStripProfessionalRenderer(new CustomProfessionalColors());
}
```

```
' This event handler is invoked when the "Change colors"
' ToolStripItem is clicked. It assigns the Renderer
' property for the ToolStrip control.
Sub ChangeColors_Click(ByVal sender As Object, ByVal e As EventArgs)
    ToolStripManager.Renderer = New ToolStripProfessionalRenderer(New CustomProfessionalColors())
End Sub
```

Compiling the Code

This example requires:

- References to the System.Design, System.Drawing, and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[ToolStripManager](#)

[ProfessionalColorTable](#)

[MenuStrip](#)

[ToolStrip](#)

[ToolStripProfessionalRenderer](#)

[ToolStrip Control](#)

How to: Set the ToolStrip Renderer for an Application

5/4/2018 • 6 min to read • [Edit Online](#)

You can customize the appearance of your [ToolStrip](#) controls individually or for all the [ToolStrip](#) controls in your application.

Example

The following code example demonstrates how to selectively apply a custom renderer to a [ToolStrip](#) control and a [MenuStrip](#) control.

To use this code example, compile and run the application, and then select the scope of the custom rendering from the [ComboBox](#) control. Click **Apply** to set the renderer.

To see custom menu item rendering, select the [MenuStrip](#) option from the [ComboBox](#) control, click **Apply**, and then open the [File](#) menu item.

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Drawing;
```

```
Imports System
Imports System.Collections.Generic
Imports System.Windows.Forms
Imports System.Drawing
```

```
// This example demonstrates how to apply a
// custom professional renderer to an individual
// ToolStrip or to the application as a whole.
class Form6 : Form
{
    ComboBox targetComboBox = new ComboBox();

    public Form6()
    {
        // Alter the renderer at the top level.

        // Create and populate a new ToolStrip control.
        ToolStrip ts = new ToolStrip();
        ts.Name = "ToolStrip";
        ts.Items.Add("Apples");
        ts.Items.Add("Oranges");
        ts.Items.Add("Pears");

        // Create a new menustrip with a new window.
        MenuStrip ms = new MenuStrip();
        ms.Name = "MenuStrip";
        ms.Dock = DockStyle.Top;

        // add top level items
        ToolStripMenuItem fileMenuItem = new ToolStripMenuItem("File");
        ms.Items.Add(fileMenuItem);
        ms.Items.Add("Edit");
        ms.Items.Add("View");
        ms.Items.Add("Window");
```

```

// Add subitems to the "File" menu.
fileMenuItem.DropDownItems.Add("Open");
fileMenuItem.DropDownItems.Add("Save");
fileMenuItem.DropDownItems.Add("Save As..."); 
fileMenuItem.DropDownItems.Add("-"); 
fileMenuItem.DropDownItems.Add("Exit");

// Add a Button control to apply renderers.
Button applyButton = new Button();
applyButton.Text = "Apply Custom Renderer";
applyButton.Click += new EventHandler(applyButton_Click);

// Add the ComboBox control for choosing how
// to apply the renderers.
targetComboBox.Items.Add("All");
targetComboBox.Items.Add("MenuStrip");
targetComboBox.Items.Add("ToolStrip");
targetComboBox.Items.Add("Reset");

// Create and set up a TableLayoutPanel control.
TableLayoutPanel tlp = new TableLayoutPanel();
tlp.Dock = DockStyle.Fill;
tlp.RowCount = 1;
tlp.ColumnCount = 2;
tlp.ColumnStyles.Add(new ColumnStyle(SizeType.AutoSize));
tlp.ColumnStyles.Add(new ColumnStyle(SizeType.Percent));
tlp.Controls.Add(applyButton);
tlp.Controls.Add(targetComboBox);

// Create a GroupBox for the TableLayoutPanel control.
GroupBox gb = new GroupBox();
gb.Text = "Apply Renderers";
gb.Dock = DockStyle.Fill;
gb.Controls.Add(tlp);

// Add the GroupBox to the form.
this.Controls.Add(gb);

// Add the ToolStrip to the form's Controls collection.
this.Controls.Add(ts);

// Add the MenuStrip control last.
// This is important for correct placement in the z-order.
this.Controls.Add(ms);
}

// This event handler is invoked when
// the "Apply Renderers" button is clicked.
// Depending on the value selected in a ComboBox control,
// it applies a custom renderer selectively to
// individual MenuStrip or ToolStrip controls,
// or it applies a custom renderer to the
// application as a whole.
void applyButton_Click(object sender, EventArgs e)
{
    ToolStrip ms = ToolStripManager.FindToolStrip("MenuStrip");
    ToolStrip ts = ToolStripManager.FindToolStrip("ToolStrip");

    if (targetComboBox.SelectedItem != null)
    {
        switch (targetComboBox.SelectedItem.ToString())
        {
            case "Reset":
            {
                ms.RenderMode = ToolStripRenderMode.ManagerRenderMode;
                ts.RenderMode = ToolStripRenderMode.ManagerRenderMode;

                // Set the default RenderMode to Professional.

```

```

        ToolStripManager.RenderMode = ToolStripManagerRenderMode.Professional;

        break;
    }

    case "All":
    {
        ms.RenderMode = ToolStripRenderMode.ManagerRenderMode;
        ts.RenderMode = ToolStripRenderMode.ManagerRenderMode;

        // Assign the custom renderer at the application level.
        ToolStripManager.Renderer = new CustomProfessionalRenderer();

        break;
    }

    case "MenuStrip":
    {
        // Assign the custom renderer to the ToolStrip control only.
        ms.Renderer = new CustomProfessionalRenderer();

        break;
    }

    case "ToolStrip":
    {
        // Assign the custom renderer to the ToolStrip control only.
        ts.Renderer = new CustomProfessionalRenderer();

        break;
    }
}
}

// This type demonstrates a custom renderer. It overrides the
// OnRenderMenuItemBackground and OnRenderButtonBackground methods
// to customize the backgrounds of ToolStrip items and ToolStrip buttons.
class CustomProfessionalRenderer : ToolStripProfessionalRenderer
{
    protected override void OnRenderMenuItemBackground(ToolStripItemEventArgs e)
    {
        if (e.Item.Selected)
        {
            using (Brush b = new SolidBrush(ProfessionalColors.SeparatorLight))
            {
                e.Graphics.FillEllipse(b, e.Item.ContentRectangle);
            }
        }
        else
        {
            using (Pen p = new Pen(ProfessionalColors.SeparatorLight))
            {
                e.Graphics.DrawEllipse(p, e.Item.ContentRectangle);
            }
        }
    }

    protected override void OnRenderButtonBackground(ToolStripItemEventArgs e)
    {
        Rectangle r = Rectangle.Inflate(e.Item.ContentRectangle, -2, -2);

        if (e.Item.Selected)
        {
            using (Brush b = new SolidBrush(ProfessionalColors.SeparatorLight))
            {
                e.Graphics.FillRectangle(b, r);
            }
        }
    }
}

```

```

        }
    else
    {
        using (Pen p = new Pen(ProfessionalColors.SeparatorLight))
        {
            e.Graphics.DrawRectangle(p, r);
        }
    }
}
}

```

```

' This example demonstrates how to apply a
' custom professional renderer to an individual
' ToolStrip or to the application as a whole.
Class Form6
    Inherits Form
    Private targetComboBox As New ComboBox()

    Public Sub New()

        ' Alter the renderer at the top level.
        ' Create and populate a new ToolStrip control.
        Dim ts As New ToolStrip()
        ts.Name = "ToolStrip"
        ts.Items.Add("Apples")
        ts.Items.Add("Oranges")
        ts.Items.Add("Pears")

        ' Create a new menustrip with a new window.
        Dim ms As New MenuStrip()
        ms.Name = "MenuStrip"
        ms.Dock = DockStyle.Top

        ' add top level items
        Dim fileMenuItem As New ToolStripMenuItem("File")
        ms.Items.Add(fileMenuItem)
        ms.Items.Add("Edit")
        ms.Items.Add("View")
        ms.Items.Add("Window")

        ' Add subitems to the "File" menu.
        fileMenuItem.DropDownItems.Add("Open")
        fileMenuItem.DropDownItems.Add("Save")
        fileMenuItem.DropDownItems.Add("Save As...")
        fileMenuItem.DropDownItems.Add("-")
        fileMenuItem.DropDownItems.Add("Exit")

        ' Add a Button control to apply renderers.
        Dim applyButton As New Button()
        applyButton.Text = "Apply Custom Renderer"
        AddHandler applyButton.Click, AddressOf applyButton_Click

        ' Add the ComboBox control for choosing how
        ' to apply the renderers.
        targetComboBox.Items.Add("All")
        targetComboBox.Items.Add("MenuStrip")
        targetComboBox.Items.Add("ToolStrip")
        targetComboBox.Items.Add("Reset")

        ' Create and set up a TableLayoutPanel control.
        Dim tlp As New TableLayoutPanel()
        tlp.Dock = DockStyle.Fill
        tlp.RowCount = 1
        tlp.ColumnCount = 2
        tlp.ColumnStyles.Add(New ColumnStyle(SizeType.AutoSize))
        tlp.ColumnStyles.Add(New ColumnStyle(SizeType.Percent))
    End Sub
End Class

```

```

    tlp.Controls.Add(applyButton)
    tlp.Controls.Add(targetComboBox)

    ' Create a GroupBox for the TableLayoutPanel control.
    Dim gb As New GroupBox()
    gb.Text = "Apply Renderers"
    gb.Dock = DockStyle.Fill
    gb.Controls.Add(tlp)

    ' Add the GroupBox to the form.
    Me.Controls.Add(gb)

    ' Add the ToolStrip to the form's Controls collection.
    Me.Controls.Add(ts)

    ' Add the MenuStrip control last.
    ' This is important for correct placement in the z-order.
    Me.Controls.Add(ms)
End Sub

' This event handler is invoked when
' the "Apply Renderers" button is clicked.
' Depending on the value selected in a ComboBox
' control, it applies a custom renderer selectively
' to individual MenuStrip or ToolStrip controls,
' or it applies a custom renderer to the
' application as a whole.
Sub applyButton_Click(ByVal sender As Object, ByVal e As EventArgs)
    Dim ms As ToolStrip = ToolStripManager.FindToolStrip("MenuStrip")
    Dim ts As ToolStrip = ToolStripManager.FindToolStrip("ToolStrip")

    If targetComboBox.SelectedItem IsNot Nothing Then

        Select Case targetComboBox.SelectedItem.ToString()
            Case "Reset"
                ms.RenderMode = ToolStripRenderMode.ManagerRenderMode
                ts.RenderMode = ToolStripRenderMode.ManagerRenderMode

                ' Set the default RenderMode to Professional.
                ToolStripManager.RenderMode = ToolStripManagerRenderMode.Professional

                Exit Select

            Case "All"
                ms.RenderMode = ToolStripRenderMode.ManagerRenderMode
                ts.RenderMode = ToolStripRenderMode.ManagerRenderMode

                ' Assign the custom renderer at the application level.
                ToolStripManager.Renderer = New CustomProfessionalRenderer()

                Exit Select

            Case "MenuStrip"
                ' Assign the custom renderer to the MenuStrip control only.
                ms.Renderer = New CustomProfessionalRenderer()

                Exit Select

            Case "ToolStrip"
                ' Assign the custom renderer to the ToolStrip control only.
                ts.Renderer = New CustomProfessionalRenderer()

                Exit Select
        End Select
    End If
End Sub
End Class

```

```

' This type demonstrates a custom renderer. It overrides the
' OnRenderMenuItemBackground and OnRenderButtonBackground methods
' to customize the backgrounds of MenuStrip items and ToolStrip buttons.
Class CustomProfessionalRenderer
    Inherits ToolStripProfessionalRenderer

    Protected Overrides Sub OnRenderMenuItemBackground(e As ToolStripItemEventArgs)
        If e.Item.Selected Then
            Dim b = New SolidBrush(ProfessionalColors.SeparatorLight)
            Try
                e.Graphics.FillEllipse(b, e.Item.ContentRectangle)
            Finally
                b.Dispose()
            End Try
        Else
            Dim p As New Pen(ProfessionalColors.SeparatorLight)
            Try
                e.Graphics.DrawEllipse(p, e.Item.ContentRectangle)
            Finally
                p.Dispose()
            End Try
        End If
    End Sub

    Protected Overrides Sub OnRenderButtonBackground(e As ToolStripItemEventArgs)
        Dim r As Rectangle = Rectangle.Inflate(e.Item.ContentRectangle, - 2, - 2)

        If e.Item.Selected Then
            Dim b = New SolidBrush(ProfessionalColors.SeparatorLight)
            Try
                e.Graphics.FillRectangle(b, r)
            Finally
                b.Dispose()
            End Try
        Else
            Dim p As New Pen(ProfessionalColors.SeparatorLight)
            Try
                e.Graphics.DrawRectangle(p, r)
            Finally
                p.Dispose()
            End Try
        End If
    End Sub
End Class

```

Set the [ToolStripManager.Renderer](#) property to apply a custom renderer to all the [ToolStrip](#) controls in your application.

Set the [ToolStrip.Renderer](#) property to apply a custom renderer to an individual [ToolStrip](#) control.

Compiling the Code

This example requires:

- References to the System.Design, System.Drawing, and System.Windows.Forms assemblies.

For information about building this example from the command line for visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[ToolStripManager](#)

[MenuStrip](#)

[ToolStrip](#)

[ToolStripProfessionalRenderer](#)

[ToolStrip Control](#)

How to: Stretch a ToolStripTextBox to Fill the Remaining Width of a ToolStrip (Windows Forms)

5/4/2018 • 4 min to read • [Edit Online](#)

When you set the [Stretch](#) property of a [ToolStrip](#) control to `true`, the control fills its container from end to end, and resizes when its container resizes. In this configuration, you may find it useful to stretch an item in the control, such as a [ToolStripTextBox](#), to fill the available space and to resize when the control resizes. This stretching is useful, for example, if you want to achieve appearance and behavior similar to the address bar in Microsoft® Internet Explorer.

Example

The following code example provides a class derived from [ToolStripTextBox](#) called [ToolStripSpringTextBox](#). This class overrides the [GetPreferredSize](#) method to calculate the available width of the parent [ToolStrip](#) control after the combined width of all other items has been subtracted. This code example also provides a [Form](#) class and a [Program](#) class to demonstrate the new behavior.

```
using System;
using System.Drawing;
using System.Windows.Forms;

public class ToolStripSpringTextBox : ToolStripTextBox
{
    public override Size GetPreferredSize(Size constrainingSize)
    {
        // Use the default size if the text box is on the overflow menu
        // or is on a vertical ToolStrip.
        if (IsOnOverflow || Owner.Orientation == Orientation.Vertical)
        {
            return DefaultSize;
        }

        // Declare a variable to store the total available width as
        // it is calculated, starting with the display width of the
        // owning ToolStrip.
        Int32 width = Owner.DisplayRectangle.Width;

        // Subtract the width of the overflow button if it is displayed.
        if (Owner.OverflowButton.Visible)
        {
            width = width - Owner.OverflowButton.Width -
                Owner.OverflowButton.Margin.Horizontal;
        }

        // Declare a variable to maintain a count of ToolStripSpringTextBox
        // items currently displayed in the owning ToolStrip.
        Int32 springBoxCount = 0;

        foreach (ToolStripItem item in Owner.Items)
        {
            // Ignore items on the overflow menu.
            if (item.IsOnOverflow) continue;

            if (item is ToolStripSpringTextBox)
            {
                // For ToolStripSpringTextBox items, increment the count and
                // subtract the margin width from the total available width.
                springBoxCount++;
                width -= item.Margin.Horizontal;
            }
        }
    }
}
```

```

        }
        else
        {
            // For all other items, subtract the full width from the total
            // available width.
            width = width - item.Width - item.Margin.Horizontal;
        }
    }

    // If there are multiple ToolStripSpringTextBox items in the owning
    // ToolStrip, divide the total available width between them.
    if (springBoxCount > 1) width /= springBoxCount;

    // If the available width is less than the default width, use the
    // default width, forcing one or more items onto the overflow menu.
    if (width < DefaultSize.Width) width = DefaultSize.Width;

    // Retrieve the preferred size from the base class, but change the
    // width to the calculated width.
    Size size = base.GetPreferredSize(constrainingSize);
    size.Width = width;
    return size;
}
}

public class Form1 : Form
{
    public Form1()
    {
        ToolStrip toolStrip1 = new ToolStrip();
        toolStrip1.Dock = DockStyle.Top;
        toolStrip1.Items.Add(new ToolStripLabel("Address"));
        toolStrip1.Items.Add(new ToolStripSpringTextBox());
        toolStrip1.Items.Add(new ToolStripButton("Go"));
        Controls.Add(toolStrip1);
        Text = "ToolStripSpringTextBox demo";
    }
}

static class Program
{
    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new Form1());
    }
}

```

```

Imports System
Imports System.Drawing
Imports System.Windows.Forms

Public Class ToolStripSpringTextBox
    Inherits ToolStripTextBox

    Public Overrides Function GetPreferredSize( _
        ByVal constrainingSize As Size) As Size

        ' Use the default size if the text box is on the overflow menu
        ' or is on a vertical ToolStrip.
        If IsOnOverflow Or Owner.Orientation = Orientation.Vertical Then
            Return DefaultSize
        End If

        ' Declare a variable to store the total available width as

```

```

' it is calculated, starting with the display width of the
' owning ToolStrip.
Dim width As Int32 = Owner.DisplayRectangle.Width

' Subtract the width of the overflow button if it is displayed.
If Owner.OverflowButton.Visible Then
    width = width - Owner.OverflowButton.Width - _
        Owner.OverflowButton.Margin.Horizontal()
End If

' Declare a variable to maintain a count of ToolStripSpringTextBox
' items currently displayed in the owning ToolStrip.
Dim springBoxCount As Int32 = 0

For Each item As ToolStripItem In Owner.Items

    ' Ignore items on the overflow menu.
    If item.IsOnOverflow Then Continue For

    If TypeOf item Is ToolStripSpringTextBox Then
        ' For ToolStripSpringTextBox items, increment the count and
        ' subtract the margin width from the total available width.
        springBoxCount += 1
        width -= item.Margin.Horizontal
    Else
        ' For all other items, subtract the full width from the total
        ' available width.
        width = width - item.Width - item.Margin.Horizontal
    End If
Next

' If there are multiple ToolStripSpringTextBox items in the owning
' ToolStrip, divide the total available width between them.
If springBoxCount > 1 Then width = CInt(width / springBoxCount)

' If the available width is less than the default width, use the
' default width, forcing one or more items onto the overflow menu.
If width < DefaultSize.Width Then width = DefaultSize.Width

' Retrieve the preferred size from the base class, but change the
' width to the calculated width.
Dim preferredSize As Size = MyBase.GetPreferredSize(constrainingSize)
preferredSize.Width = width
Return preferredSize

End Function
End Class

Public Class Form1
Inherits Form

Public Sub New()
    Dim toolstrip1 As New ToolStrip()
    With toolstrip1
        .Dock = DockStyle.Top
        .Items.Add(New ToolStripLabel("Address"))
        .Items.Add(New ToolStripSpringTextBox())
        .Items.Add(New ToolStripButton("Go"))
    End With
    Controls.Add(toolstrip1)
    Text = "ToolStripSpringTextBox demo"
End Sub

End Class

Public Class Program

<STAThread()> Public Shared Sub Main()
    Application.EnableVisualStyles()

```

```
.....,--\,
Application.SetCompatibleTextRenderingDefault(False)
Application.Run(New Form1())
End Sub

End Class
```

Compiling the Code

This example requires:

- References to the System, System.Drawing, and System.Windows.Forms assemblies.

See Also

[ToolStrip](#)

[ToolStrip.Stretch](#)

[ToolStripTextBox](#)

[ToolStripTextBox.GetPreferredSize](#)

[ToolStrip Control Architecture](#)

[How to: Use the Spring Property Interactively in a StatusStrip](#)

How to: Use ToolTips in ToolStrip Controls

5/4/2018 • 1 min to read • [Edit Online](#)

You can display a **ToolTip** for the **ToolStrip** control you want by setting the control's **ShowItemToolTips** property to `true`.

To display a ToolTip

- Set the **ShowItemToolTips** property of the control to `true`.

The default value of **ToolStrip.ShowItemToolTips** is `true`, and the default value of **MenuStrip.ShowItemToolTips** and **StatusStrip.ShowItemToolTips** is `false`.

To use the **ToolTipText** property for the **ToolTip** text of a **ToolStripButton**

1. Set the **ShowItemToolTips** property of the button to `true`.
2. Set the **ToolStripButton.AutoToolTip** property of the button to `false`.

The **AutoToolTip** property is `true` by default for **ToolStripButton**, **ToolStripDropDownButton**, and **ToolStripSplitButton**.

A **ToolStripButton** uses its **Text** property for the **ToolTip** text by default. Use this procedure to display custom text in a **ToolStripButton.ToolTip**.

NOTE

If you set **ToolStripItemDisplayStyle** to **None** or **Image**, no text will appear on the button, but the tool tip still appears.

See Also

[ShowItemToolTips](#)

[ToolStripButton](#)

[ToolStripDropDownButton](#)

[ToolStripSplitButton](#)

[ToolStrip Control Overview](#)

How to: Wrap a Windows Forms Control with ToolStripControlHost

5/4/2018 • 6 min to read • [Edit Online](#)

[ToolStripControlHost](#) is designed to enable hosting of arbitrary Windows Forms controls by using the [ToolStripControlHost](#) constructor or by extending [ToolStripControlHost](#) itself. It is easier to wrap the control by extending [ToolStripControlHost](#) and implementing properties and methods that expose frequently used properties and methods of the control. You can also expose events for the control at the [ToolStripControlHost](#) level.

To host a control in a ToolStripControlHost by derivation

1. Extend [ToolStripControlHost](#). Implement a default constructor that calls the base class constructor passing in the desired control.

```
// Call the base constructor passing in a MonthCalendar instance.  
ToolStripMonthCalendar() : ToolStripControlHost( gcnew MonthCalendar ) {}
```

```
// Call the base constructor passing in a MonthCalendar instance.  
public ToolStripMonthCalendar() : base (new MonthCalendar()) { }
```

```
' Call the base constructor passing in a MonthCalendar instance.  
Public Sub New()  
    MyBase.New(New MonthCalendar())  
  
End Sub
```

2. Declare a property of the same type as the wrapped control and return `Control` as the correct type of control in the property's accessor.

```
property MonthCalendar^ MonthCalendarControl  
{  
    MonthCalendar^ get()  
    {  
        return static_cast<MonthCalendar^>(Control);  
    }  
}
```

```
public MonthCalendar MonthCalendarControl  
{  
    get  
    {  
        return Control as MonthCalendar;  
    }  
}
```

```

Public ReadOnly Property MonthCalendarControl() As MonthCalendar
    Get
        Return CType(Control, MonthCalendar)
    End Get
End Property

```

3. Expose other frequently used properties and methods of the wrapped control with properties and methods in the extended class.

```

property Day FirstDayOfWeek
{
    // Expose the MonthCalendar.FirstDayOfWeek as a property.
    Day get()
    {
        return MonthCalendarControl->FirstDayOfWeek;
    }

    void set( Day value )
    {
        MonthCalendarControl->FirstDayOfWeek = value;
    }
}

// Expose the AddBoldedDate method.
void AddBoldedDate( DateTime dateToBold )
{
    MonthCalendarControl->AddBoldedDate( dateToBold );
}

```

```

// Expose the MonthCalendar.FirstDayOfWeek as a property.
public Day FirstDayOfWeek
{
    get
    {
        return MonthCalendarControl.FirstDayOfWeek;
    }
    set { MonthCalendarControl.FirstDayOfWeek = value; }
}

// Expose the AddBoldedDate method.
public void AddBoldedDate(DateTime dateToBold)
{
    MonthCalendarControl.AddBoldedDate(dateToBold);
}

```

```

' Expose the MonthCalendar.FirstDayOfWeek as a property.
Public Property FirstDayOfWeek() As Day
    Get
        Return MonthCalendarControl.FirstDayOfWeek
    End Get
    Set
        MonthCalendarControl.FirstDayOfWeek = value
    End Set
End Property

' Expose the AddBoldedDate method.
Public Sub AddBoldedDate(ByVal dateToBold As DateTime)
    MonthCalendarControl.AddBoldedDate(dateToBold)
End Sub

```

4. Optionally, override the [OnSubscribeControlEvents](#), and [OnUnsubscribeControlEvents](#) methods and add the control events you want to expose.

```

void OnSubscribeControlEvents( System::Windows::Forms::Control^ c )
{
    // Call the base so the base events are connected.
    __super::OnSubscribeControlEvents( c );

    // Cast the control to a MonthCalendar control.
    MonthCalendar^ monthCalendarControl = (MonthCalendar^)c;

    // Add the event.
    monthCalendarControl->DateChanged += gcnew DateRangeEventHandler( this,
    &ToolStripMonthCalendar::HandleDateChanged );
}

void OnUnsubscribeControlEvents( System::Windows::Forms::Control^ c )
{

    // Call the base method so the basic events are unsubscribed.
    __super::OnUnsubscribeControlEvents( c );

    // Cast the control to a MonthCalendar control.
    MonthCalendar^ monthCalendarControl = (MonthCalendar^)c;

    // Remove the event.
    monthCalendarControl->DateChanged -= gcnew DateRangeEventHandler( this,
    &ToolStripMonthCalendar::HandleDateChanged );
}

```

```

protected override void OnSubscribeControlEvents(Control c)
{
    // Call the base so the base events are connected.
    base.OnSubscribeControlEvents(c);

    // Cast the control to a MonthCalendar control.
    MonthCalendar monthCalendarControl = (MonthCalendar) c;

    // Add the event.
    monthCalendarControl.DateChanged +=
        new DateRangeEventHandler(OnDateChanged);
}

protected override void OnUnsubscribeControlEvents(Control c)
{
    // Call the base method so the basic events are unsubscribed.
    base.OnUnsubscribeControlEvents(c);

    // Cast the control to a MonthCalendar control.
    MonthCalendar monthCalendarControl = (MonthCalendar) c;

    // Remove the event.
    monthCalendarControl.DateChanged -=
        new DateRangeEventHandler(OnDateChanged);
}

```

```

Protected Overrides Sub OnSubscribeControlEvents(ByVal c As Control)

    ' Call the base so the base events are connected.
    MyBase.OnSubscribeControlEvents(c)

    ' Cast the control to a MonthCalendar control.
    Dim monthCalendarControl As MonthCalendar = _
        CType(c, MonthCalendar)

    ' Add the event.
    AddHandler monthCalendarControl.DateChanged, _
        AddressOf HandleDateChanged

End Sub

Protected Overrides Sub OnUnsubscribeControlEvents(ByVal c As Control)
    ' Call the base method so the basic events are unsubscribed.
    MyBase.OnUnsubscribeControlEvents(c)

    ' Cast the control to a MonthCalendar control.
    Dim monthCalendarControl As MonthCalendar = _
        CType(c, MonthCalendar)

    ' Remove the event.
    RemoveHandler monthCalendarControl.DateChanged, _
        AddressOf HandleDateChanged

End Sub

```

5. Provide the necessary wrapping for the events you want to expose.

```

// Declare the DateChanged event.
// Raise the DateChanged event.
void HandleDateChanged( Object^ sender, DateRangeEventArgs^ e )
{
    if ( DateChanged != nullptr )
    {
        DateChanged( this, e );
    }
}

```

```

// Declare the DateChanged event.
public event DateRangeEventHandler DateChanged;

// Raise the DateChanged event.
private void OnDateChanged(object sender, DateRangeEventArgs e)
{
    if (DateChanged != null)
    {
        DateChanged(this, e);
    }
}

```

```

' Declare the DateChanged event.
Public Event DateChanged As DateRangeEventHandler

' Raise the DateChanged event.
Private Sub HandleDateChanged(ByVal sender As Object, _
    ByVal e As DateRangeEventArgs)

    RaiseEvent DateChanged(Me, e)
End Sub
End Class

```

Example

```

//Declare a class that inherits from ToolStripControlHost.
public ref class ToolStripMonthCalendar: public ToolStripControlHost
{
public:
    // Call the base constructor passing in a MonthCalendar instance.
    ToolStripMonthCalendar() : ToolStripControlHost( gcnew MonthCalendar ) {}

    property MonthCalendar^ MonthCalendarControl
    {
        MonthCalendar^ get()
        {
            return static_cast<MonthCalendar^>(Control);
        }
    }
    property Day FirstDayOfWeek
    {
        // Expose the MonthCalendar.FirstDayOfWeek as a property.
        Day get()
        {
            return MonthCalendarControl->FirstDayOfWeek;
        }

        void set( Day value )
        {
            MonthCalendarControl->FirstDayOfWeek = value;
        }
    }

    // Expose the AddBoldedDate method.
    void AddBoldedDate( DateTime dateToBold )
    {
        MonthCalendarControl->AddBoldedDate( dateToBold );
    }

protected:
    // Subscribe and unsubscribe the control events you wish to expose.
    void OnSubscribeControlEvents( System::Windows::Forms::Control^ c )
    {
        // Call the base so the base events are connected.
        __super::OnSubscribeControlEvents( c );

        // Cast the control to a MonthCalendar control.
        MonthCalendar^ monthCalendarControl = (MonthCalendar^)c;

        // Add the event.
        monthCalendarControl->DateChanged += gcnew DateRangeEventHandler( this,
&ToolStripMonthCalendar::HandleDateChanged );
    }

    void OnUnsubscribeControlEvents( System::Windows::Forms::Control^ c )
    {

```

```

// Call the base method so the basic events are unsubscribed.
__super::OnUnsubscribeControlEvents( c );

// Cast the control to a MonthCalendar control.
MonthCalendar^ monthCalendarControl = (MonthCalendar^)c;

// Remove the event.
monthCalendarControl->DateChanged -= gcnew DateRangeEventHandler( this,
&ToolStripMonthCalendar::HandleDateChanged );
}

public:
event DateRangeEventHandler^ DateChanged;

private:
// Declare the DateChanged event.
// Raise the DateChanged event.
void HandleDateChanged( Object^ sender, DateRangeEventArgs^ e )
{
    if ( DateChanged != nullptr )
    {
        DateChanged( this, e );
    }
}
};


```

```

//Declare a class that inherits from ToolStripControlHost.
public class ToolStripMonthCalendar : ToolStripControlHost
{
// Call the base constructor passing in a MonthCalendar instance.
public ToolStripMonthCalendar() : base (new MonthCalendar()) { }

public MonthCalendar MonthCalendarControl
{
get
{
    return Control as MonthCalendar;
}
}

// Expose the MonthCalendar.FirstDayOfWeek as a property.
public Day FirstDayOfWeek
{
get
{
    return MonthCalendarControl.FirstDayOfWeek;
}
set { MonthCalendarControl.FirstDayOfWeek = value; }
}

// Expose the AddBoldedDate method.
public void AddBoldedDate(DateTime dateToBold)
{
    MonthCalendarControl.AddBoldedDate(dateToBold);
}

// Subscribe and unsubscribe the control events you wish to expose.
protected override void OnSubscribeControlEvents(Control c)
{
// Call the base so the base events are connected.
base.OnSubscribeControlEvents(c);

// Cast the control to a MonthCalendar control.
MonthCalendar monthCalendarControl = (MonthCalendar) c;

// Add the event.
monthCalendarControl.DateChanged +=
```

```

        new DateRangeEventHandler(OnDateChanged);
    }

protected override void OnUnsubscribeControlEvents(Control c)
{
    // Call the base method so the basic events are unsubscribed.
    base.OnUnsubscribeControlEvents(c);

    // Cast the control to a MonthCalendar control.
    MonthCalendar monthCalendarControl = (MonthCalendar) c;

    // Remove the event.
    monthCalendarControl.DateChanged -=
        new DateRangeEventHandler(OnDateChanged);
}

// Declare the DateChanged event.
public event DateRangeEventHandler DateChanged;

// Raise the DateChanged event.
private void OnDateChanged(object sender, DateRangeEventArgs e)
{
    if (DateChanged != null)
    {
        DateChanged(this, e);
    }
}
}

```

'Declare a class that inherits from ToolStripControlHost.

```

Public Class ToolStripMonthCalendar
    Inherits ToolStripControlHost

    ' Call the base constructor passing in a MonthCalendar instance.
    Public Sub New()
        MyBase.New(New MonthCalendar())

    End Sub

    Public ReadOnly Property MonthCalendarControl() As MonthCalendar
        Get
            Return CType(Control, MonthCalendar)
        End Get
    End Property

    ' Expose the MonthCalendar.FirstDayOfWeek as a property.
    Public Property FirstDayOfWeek() As Day
        Get
            Return MonthCalendarControl.FirstDayOfWeek
        End Get
        Set
            MonthCalendarControl.FirstDayOfWeek = value
        End Set
    End Property

    ' Expose the AddBoldedDate method.
    Public Sub AddBoldedDate(ByVal dateToBold As DateTime)
        MonthCalendarControl.AddBoldedDate(dateToBold)

    End Sub

    ' Subscribe and unsubscribe the control events you wish to expose.
    Protected Overrides Sub OnSubscribeControlEvents(ByVal c As Control)

        ' Call the base so the base events are connected.
        MyBase.OnSubscribeControlEvents(c)

```

```

' Cast the control to a MonthCalendar control.
Dim monthCalendarControl As MonthCalendar = _
    CType(c, MonthCalendar)

' Add the event.
AddHandler monthCalendarControl.DateChanged, _
    AddressOf HandleDateChanged

End Sub

Protected Overrides Sub OnUnsubscribeControlEvents(ByVal c As Control)
    ' Call the base method so the basic events are unsubscribed.
    MyBase.OnUnsubscribeControlEvents(c)

    ' Cast the control to a MonthCalendar control.
    Dim monthCalendarControl As MonthCalendar = _
        CType(c, MonthCalendar)

    ' Remove the event.
    RemoveHandler monthCalendarControl.DateChanged, _
        AddressOf HandleDateChanged

End Sub

' Declare the DateChanged event.
Public Event DateChanged As DateRangeEventHandler

' Raise the DateChanged event.
Private Sub HandleDateChanged(ByVal sender As Object, _
    ByVal e As DateRangeEventArgs)

    RaiseEvent DateChanged(Me, e)
End Sub
End Class

```

Compiling the Code

- This example requires:
- References to the System and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

- [ToolStripControlHost](#)
[ToolStrip Control Overview](#)
[ToolStrip Control Architecture](#)
[ToolStrip Technology Summary](#)

Walkthrough: Creating a Professionally Styled ToolStrip Control

5/4/2018 • 15 min to read • [Edit Online](#)

You can give your application's [ToolStrip](#) controls a professional appearance and behavior by writing your own class derived from the [ToolStripProfessionalRenderer](#) type.

This walkthrough demonstrates how to use [ToolStrip](#) controls to create a composite control that resembles the **Navigation Pane** provided by Microsoft® Outlook®. The following tasks are illustrated in this walkthrough:

- Creating a Windows Control Library project.
- Designing the StackView Control.
- Implementing a Custom Renderer.

When you are finished, you will have a reusable custom client control with the professional appearance of a Microsoft Office® XP control.

To copy the code in this topic as a single listing, see [How to: Create a Professionally Styled ToolStrip Control](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

Prerequisites

In order to complete this walkthrough, you will need:

- Sufficient permissions to be able to create and run Windows Forms application projects on the computer where Visual Studio is installed.

Creating a Windows Control Library Project

The first step is to create the control library project.

To create the control library project

1. Create a new Windows Control Library project named `StackViewLibrary`.
2. In **Solution Explorer**, delete the project's default control by deleting the source file named "UserControl1.cs" or "UserControl1.vb", depending on your language of choice.

For more information, see [NIB:How to: Remove, Delete, and Exclude Items](#).

3. Add a new [UserControl](#) item to the **StackViewLibrary** project. Give the new source file a base name of `StackView`.

Designing the StackView Control

The `StackView` control is a composite control with one child [ToolStrip](#) control. For more information about composite controls, see [Varieties of Custom Controls](#).

To design the StackView control

1. From the **Toolbox**, drag a **ToolStrip** control to the design surface.
2. In the **Properties** window, set the **ToolStrip** control's properties according to the following table.

PROPERTY	VALUE
Name	stackStrip
CanOverflow	false
Dock	Bottom
Font	Tahoma, 10pt, style=Bold
GripStyle	Hidden
LayoutStyle	VerticalStackWithOverflow
Padding	0, 7, 0, 0
RenderMode	Professional

3. In the Windows Forms Designer, click the **ToolStrip** control's **Add** button and add a **ToolStripButton** to the **stackStrip** control.
4. In the **Properties** window, set the **ToolStripButton** control's properties according to the following table.

PROPERTY	VALUE
Name	mailStackButton
CheckOnClick	true
CheckState	Checked
DisplayStyle	ImageAndText
ImageAlign	MiddleLeft
ImageScaling	None
ImageTransparentColor	238, 238, 238
Margin	0, 0, 0, 0
Padding	3, 3, 3, 3
Text	Mail
TextAlign	MiddleLeft

5. Repeat step 7 for three more **ToolStripButton** controls.

Name the controls `calendarStackButton`, `contactsStackButton`, and `tasksStackButton`. Set the value of the `Text` property to **Calendar**, **Contacts**, and **Tasks**, respectively.

Handling Events

Two events are important to make the `StackView` control behave correctly. Handle the `Load` event to position the control correctly. Handle the `Click` event for each `ToolStripButton` to give the `stackView` control state behavior similar to the `RadioButton` control.

To handle events

1. In the Windows Forms Designer, select the `stackView` control.
2. In the **Properties** window, click **Events**.
3. Double-click the Load event to generate the `StackView_Load` event handler.
4. In the `StackView_Load` event handler, copy and paste the following code.

```
// This method handles the Load event for the UserControl.  
private void StackView_Load(object sender, EventArgs e)  
{  
    // Dock bottom.  
    this.Dock = DockStyle.Bottom;  
  
    // Set AutoSize.  
    this.AutoSize = true;  
}
```

```
' This method handles the Load event for the UserControl.  
Private Sub StackView_Load(sender As Object, e As EventArgs) Handles MyBase.Load  
    ' Dock bottom.  
    Me.Dock = DockStyle.Bottom  
  
    ' Set AutoSize.  
    Me.AutoSize = True  
End Sub
```

5. In the Windows Forms Designer, select the `mailStackButton` control.

6. In the **Properties** window, click **Events**.

7. Double-click the Click event.

The Windows Forms Designer generates the `mailStackButton_Click` event handler.

8. Rename the `mailStackButton_Click` event handler to `stackButton_Click`.

For more information, see [How to: Rename an Identifier \(Visual Basic\)](#).

9. Insert the following code into the `stackButton_Click` event handler.

```
// This method handles the Click event for all
// the ToolStripButton controls in the StackView.
private void stackButton_Click(object sender, EventArgs e)
{
    // Define a "one of many" state, similar to
    // the logic of a RadioButton control.
    foreach (ToolStripItem item in this.stackStrip.Items)
    {
        if ((item != sender) &&
            (item is ToolStripButton))
        {
            ((ToolStripButton)item).Checked = false;
        }
    }
}
```

```
' This method handles the Click event for all
' the ToolStripButton controls in the StackView.

Private Sub stackButton_Click(sender As Object, e As EventArgs) Handles mailStackButton.Click,
calendarStackButton.Click, contactsStackButton.Click, tasksStackButton.Click

    ' Define a "one of many" state, similar to
    ' the logic of a RadioButton control.

    Dim item As ToolStripItem
    For Each item In Me.stackStrip.Items
        If item IsNot sender AndAlso TypeOf item Is ToolStripButton Then
            CType(item, ToolStripButton).Checked = False
        End If
    Next item
End Sub
```

10. In the Windows Forms Designer, select the `calendarStackButton` control.
 11. In the **Properties** window, set the Click event to the `stackButton_Click` event handler.
 12. Repeat steps 10 and 11 for the `contactsStackButton` and `tasksStackButton` controls.

Defining Icons

Each `StackView` button has an associated icon. For convenience, each icon is represented as a Base64-encoded string, which is deserialized before a `Bitmap` is created from it. In a production environment, you store bitmap data as a resource, and your icons appear in the Windows Forms Designer. For more information, see [How to: Add Background Images to Windows Forms](#).

To define icons

1. In the Code Editor, insert the following code into the `StackView` class definition. This code initializes the bitmaps for the `ToolStripButton` icons.


```

}

// This utility method creates a bitmap from
// a Base64-encoded string.
internal static Bitmap DeserializeFromBase64(string data)
{
    // Decode the string and create a memory stream
    // on the decoded string data.
    MemoryStream stream =
        new MemoryStream(Convert.FromBase64String(data));

    // Create a new bitmap from the stream.
    Bitmap b = new Bitmap(stream);

    return b;
}

```

```

Private Shared mailBmp As Bitmap
Private Shared calendarBmp As Bitmap
Private Shared contactsBmp As Bitmap
Private Shared tasksBmp As Bitmap

Private Shared mailBmpEnc As String = "Qk32BgAAAAAAADYAAAAoAAAA" + _
    "GAAAAABgAAAABABgAAAAAAAAADEdGAxA4AAAAAAAAAAA7u7u7u7u" + _
    "7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u" + _
    "vYmDu4eBuYV/t4N9tYB7s355sXt3tnxsnhwsHZurXNsrfHrp21pomln" + _
    "oGdlnmVjnWNi7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u" + _
    "980S9bqI87F/87KA8qt68KRy76Bu7ph17ZVi54xb0ntSoGVj7u7u" + _
    "7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u" + _
    "+tKo+cyg+Mic9b+S9LqM8a135ZZmoXBSoWdk7u7u7u7u7u7u7u7u" + _
    "xZSN9ejaya0Yuvvh//js/vXo/e/a/evs/OfL/OTF+927+9u1+tSq+tk1" + _
    "+cqW8buFqHZa7JvDn2hm7u7u7u7u7u7u7u7u7u7u7u7u7u7u" + _
    "+07k//nw//fu//fr/vTl/u/c/uvU/eLB/N+7+tev8b6Hq3pe+Lh6/69q" + _
    "okpp7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u" + _
    "//jx//bs/+7z/erQ/eTC8syiqntg+M6a/90h/8CFom1s7u7u7u7u7u" + _
    "7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u" + _
    "6NO/poVyt4xx5b2T/9ap/8ybpXFw7u7u7u7u7u7u7u7u7u7u7u" + _
    "7uPbzaye/fv6/fv5v6GTvKCQvJ+Nu5qIupmJt5uKsZWE+u3e+unYo4Jq" + _
    "5720/9KjqHd17u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u" + _
    "5df05dfN5dfM5NbM49TI3Mq+3Mm93Mm8382/+unbrIJp57WGrH597u7u" + _
    "7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u" + _
    "7uPb9e719evk9uvj9+zh/PHlzs61r4VtsolC7u7u7u7u7u7u7u7u" + _
    "0aKXfdb/8uzn//////////+/37/Pn3+fXv9fHp8+v1" + _
    "8enk7uXe1si/L6f3tYeC7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u" + _
    "//////////fz8/Pv69fPx7vDw70/w20fvxN/swt/thMXnK6ft" + _
    "r4SC7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u" + _
    "RsT/RsT/PsP/Obz7NLT2MbL0K6vuJ6XpKKfrLqrvr4J/7u7u7u7u7u" + _
    "7u7u7u7u28jc0rSo0036y0v7x+r7v+f8suP9o97+jtj+idn+htX/e83/" + _
    "a8f/XcD/Tbn/SLF/R7b/rJKJ7u7u7u7u7u7u7u7u7u7u7u7u7u" + _
    "07ew29zd1e/610/60e77y+z8vun9reH+mNn+gdH/ccj/ZMP/Vr3/t7/C" + _
    "ukCX7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u" + _
    "3fL61/L7y+/8u+f9pt7+jNT+cMn/XcH/rZyW1c3L7u7u7u7u7u7u" + _
    "7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u" + _
    "r+H9k9X+kr/cv6Wb7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u" + _
    "7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u" + _
    "7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u" + _
    "7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u" + _
    "7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u" + _
    "7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u7u"

```

```
Private Shared calendarBmpEnc As String = "Qk32BgAAAAAAADYAAAAo" + _
```



```

End Sub

Public Sub New()
    Me.InitializeComponent()

    ' Assign icons to ToolStripButton controls.
    Me.InitializeImages()

    ' Set up renderers.
    Me.stackStrip.Renderer = New StackRenderer()
End Sub

' This utility method assigns icons to each
' ToolStripButton control.
Private Sub InitializeImages()
    Me.mailStackButton.Image = mailBmp
    Me.calendarStackButton.Image = calendarBmp
    Me.contactsStackButton.Image = contactsBmp
    Me.tasksStackButton.Image = tasksBmp
End Sub

' This utility method creates bitmaps for all the icons.
' It uses a utility method called DeserializeFromBase64
' to decode the Base64 image data.
Private Shared Sub CreateBitmaps()
    mailBmp = DeserializeFromBase64(mailBmpEnc)
    calendarBmp = DeserializeFromBase64(calendarBmpEnc)
    contactsBmp = DeserializeFromBase64(contactsBmpEnc)
    tasksBmp = DeserializeFromBase64(tasksBmpEnc)
End Sub

' This utility method creates a bitmap from
' a Base64-encoded string.
Friend Shared Function DeserializeFromBase64(data As String) As Bitmap
    ' Decode the string and create a memory stream
    ' on the decoded string data.
    Dim stream As New MemoryStream(Convert.FromBase64String(data))

    ' Create a new bitmap from the stream.
    Dim b As New Bitmap(stream)

    Return b
End Function

```

2. Add a call to the `InitializeImages` method in the `StackView` class constructor.

```

public StackView()
{
    this.InitializeComponent();

    // Assign icons to ToolStripButton controls.
    this.InitializeImages();

    // Set up renderers.
    this.stackStrip.Renderer = new StackRenderer();
}

```

```

Public Sub New()
    Me.InitializeComponent()

    ' Assign icons to ToolStripButton controls.
    Me.InitializeImages()

    ' Set up renderers.
    Me.stackStrip.Renderer = New StackRenderer()
End Sub

```

Implementing a Custom Renderer

You can customize most elements of the `StackView` control by implementing a class that derives from the `ToolStripRenderer` class. In this procedure, you will implement a `ToolStripProfessionalRenderer` class that customizes the grip and draws gradient backgrounds for the `ToolStripButton` controls.

To implement a custom renderer

1. Insert the following code into the `StackView` control definition.

This is the definition for the `StackRenderer` class, which overrides the `RenderGrip`, `RenderToolStripBorder`, and `RenderButtonBackground` methods to produce a custom appearance.

```

internal class StackRenderer : ToolStripProfessionalRenderer
{
    private static Bitmap titleBarGripBmp;
    private static string titleBarGripEnc =
@"Qk16AQAAAAAAADYAAAoAAAAIwAAAAMAAAABgAAAAAAAAADEDgAAxA4AAAAAAAAAUgMy+/n5+/n5uGMyuGMy+/n5+/n5uG
MyuGMy+/n5+/n5uGMyuGMy+/n5+/n5uGMyuGMy+/n5+/n5uGMyuGMy+/n5+/n5uGMyuGMy+/n5+/n5uGMyuGMy+/
n5+/n5ANj+RzIomHRh+/n5wm8/RzIomHRh+/n5wm8/RzIomHRh+/n5wm8/RzIomHRh+/n5wm8/RzIomHRh+/n5wm
8/RzIomHRh+/n5wm8/RzIomHRh+/n5wm8/RzIomHRh+/n5ANj+RzIozHtMzHtMRzIozHtMzHtMRzIozHtMzHtMRzIoz
HtMzHtMRzIozHtMzHtMRzIozHtMzHtMRzIozHtMzHtMRzIozHtMzHtMRzIozHtMzHtMRzIozHtMANj+";

    // Define titlebar colors.
    private static Color titlebarColor1 = Color.FromArgb(89, 135, 214);
    private static Color titlebarColor2 = Color.FromArgb(76, 123, 204);
    private static Color titlebarColor3 = Color.FromArgb(63, 111, 194);
    private static Color titlebarColor4 = Color.FromArgb(50, 99, 184);
    private static Color titlebarColor5 = Color.FromArgb(38, 88, 174);
    private static Color titlebarColor6 = Color.FromArgb(25, 76, 164);
    private static Color titlebarColor7 = Color.FromArgb(12, 64, 154);
    private static Color borderColor = Color.FromArgb(0, 0, 128);

    static StackRenderer()
    {
        titleBarGripBmp = StackView.DeserializeFromBase64(titleBarGripEnc);
    }

    public StackRenderer()
    {
    }

    private void DrawTitleBar(Graphics g, Rectangle rect)
    {
        // Assign the image for the grip.
        Image titlebarGrip = titleBarGripBmp;

        // Fill the titlebar.
        // This produces the gradient and the rounded-corner effect.
        g.DrawLine(new Pen(titlebarColor1), rect.X, rect.Y, rect.X + rect.Width, rect.Y);
        g.DrawLine(new Pen(titlebarColor2), rect.X, rect.Y + 1, rect.X + rect.Width, rect.Y + 1);
        g.DrawLine(new Pen(titlebarColor3), rect.X, rect.Y + 2, rect.X + rect.Width, rect.Y + 2);
        g.DrawLine(new Pen(titlebarColor4), rect.X, rect.Y + 3, rect.X + rect.Width, rect.Y + 3);
        g.DrawLine(new Pen(titlebarColor5), rect.X, rect.Y + 4, rect.X + rect.Width, rect.Y + 4);
    }
}

```

```

g.DrawLine(new Pen(titlebarColor5), rect.X, rect.Y + 4, rect.X + rect.Width, rect.Y + 4);
g.DrawLine(new Pen(titlebarColor6), rect.X, rect.Y + 5, rect.X + rect.Width, rect.Y + 5);
g.DrawLine(new Pen(titlebarColor7), rect.X, rect.Y + 6, rect.X + rect.Width, rect.Y + 6);

// Center the titlebar grip.
g.DrawImage(
    titlebarGrip,
    new Point(rect.X + ((rect.Width / 2) - (titlebarGrip.Width / 2)),
    rect.Y + 1));
}

// This method handles the RenderGrip event.
protected override void OnRenderGrip(ToolStripGripRenderEventArgs e)
{
    DrawTitleBar(
        e.Graphics,
        new Rectangle(0, 0, e.ToolStrip.Width, 7));
}

// This method handles the RenderToolStripBorder event.
protected override void OnRenderToolStripBorder(ToolStripRenderEventArgs e)
{
    DrawTitleBar(
        e.Graphics,
        new Rectangle(0, 0, e.ToolStrip.Width, 7));
}

// This method handles the RenderButtonBackground event.
protected override void OnRenderButtonBackground(ToolStripItemRenderEventArgs e)
{
    Graphics g = e.Graphics;
    Rectangle bounds = new Rectangle(Point.Empty, e.Item.Size);

    Color gradientBegin = Color.FromArgb(203, 225, 252);
    Color gradientEnd = Color.FromArgb(125, 165, 224);

    ToolStripButton button = e.Item as ToolStripButton;
    if (button.Pressed || button.Checked)
    {
        gradientBegin = Color.FromArgb(254, 128, 62);
        gradientEnd = Color.FromArgb(255, 223, 154);
    }
    else if (button.Selected)
    {
        gradientBegin = Color.FromArgb(255, 255, 222);
        gradientEnd = Color.FromArgb(255, 203, 136);
    }

    using (Brush b = new LinearGradientBrush(
        bounds,
        gradientBegin,
        gradientEnd,
        LinearGradientMode.Vertical))
    {
        g.FillRectangle(b, bounds);
    }

    e.Graphics.DrawRectangle(
        SystemPens.ControlDarkDark,
        bounds);

    g.DrawLine(
        SystemPens.ControlDarkDark,
        bounds.X,
        bounds.Y,
        bounds.Width - 1,
        bounds.Y);

    g.DrawLine(
        SystemPens.ControlDarkDark,

```

```

        SystemPens.ControlDarkDark,
        bounds.X,
        bounds.Y,
        bounds.X,
        bounds.Height - 1);

    ToolStrip toolstrip = button.Owner;
    ToolStripButton nextItem = button.Owner.GetItemAt(
        button.Bounds.X,
        button.Bounds.Bottom + 1) as ToolStripButton;

    if (nextItem == null)
    {
        g.DrawLine(
            SystemPens.ControlDarkDark,
            bounds.X,
            bounds.Height - 1,
            bounds.X + bounds.Width - 1,
            bounds.Height - 1);
    }
}
}
}

```

```

Friend Class StackRenderer
    Inherits ToolStripProfessionalRenderer
    Private Shared titleBarGripBmp As Bitmap
    Private Shared titleBarGripEnc As String =
"Qk16AQAAAAAADYAAAoAAAAIwAAAAAAABAgAAAAAAAAAADEDgAAxA4AAAAAAAAAAuGMy+/n5+/n5uGMyuGMy+/n5+/n5uGM
yuGMy+/n5+/n5uGMyuGMy+/n5+/n5uGMyuGMy+/n5+/n5uGMyuGMy+/n5+/n5uGMyuGMy+/n5+/n5uGMyuGMy+/n
5+/n5ANj+RzIomHRh+/n5wm8/RzIomHRh+/n5wm8/RzIomHRh+/n5wm8/RzIomHRh+/n5wm8/RzIomHRh+/n5wm8
/RzIomHRh+/n5wm8/RzIomHRh+/n5wm8/RzIomHRh+/n5ANj+RzIozHtMzHtMRzIozHtMzHtMRzIozHtMzHtMRzI
ozHtMzHtMRzIozHtMzHtMRzIozHtMzHtMRzIozHtMzHtMRzIozHtMzHtMRzIozHtMANj+"

    ' Define titlebar colors.
    Private Shared titlebarColor1 As Color = Color.FromArgb(89, 135, 214)
    Private Shared titlebarColor2 As Color = Color.FromArgb(76, 123, 204)
    Private Shared titlebarColor3 As Color = Color.FromArgb(63, 111, 194)
    Private Shared titlebarColor4 As Color = Color.FromArgb(50, 99, 184)
    Private Shared titlebarColor5 As Color = Color.FromArgb(38, 88, 174)
    Private Shared titlebarColor6 As Color = Color.FromArgb(25, 76, 164)
    Private Shared titlebarColor7 As Color = Color.FromArgb(12, 64, 154)
    Private Shared borderColor As Color = Color.FromArgb(0, 0, 128)

    Shared Sub New()
        titleBarGripBmp = StackView.DeserializeFromBase64(titleBarGripEnc)
    End Sub

    Public Sub New()
    End Sub

    Private Sub DrawTitleBar(ByVal g As Graphics, ByVal rect As Rectangle)

        ' Assign the image for the grip.
        Dim titlebarGrip As Image = titleBarGripBmp

        ' Fill the titlebar.
        ' This produces the gradient and the rounded-corner effect.
        g.DrawLine(New Pen(titlebarColor1), rect.X, rect.Y, rect.X + rect.Width, rect.Y)
        g.DrawLine(New Pen(titlebarColor2), rect.X, rect.Y + 1, rect.X + rect.Width, rect.Y + 1)
        g.DrawLine(New Pen(titlebarColor3), rect.X, rect.Y + 2, rect.X + rect.Width, rect.Y + 2)
        g.DrawLine(New Pen(titlebarColor4), rect.X, rect.Y + 3, rect.X + rect.Width, rect.Y + 3)
        g.DrawLine(New Pen(titlebarColor5), rect.X, rect.Y + 4, rect.X + rect.Width, rect.Y + 4)
        g.DrawLine(New Pen(titlebarColor6), rect.X, rect.Y + 5, rect.X + rect.Width, rect.Y + 5)
        g.DrawLine(New Pen(titlebarColor7), rect.X, rect.Y + 6, rect.X + rect.Width, rect.Y + 6)

        ' Center the titlebar grip.
        g.DrawImage(titlebarGrip, New Point(rect.X + (rect.Width / 2 - titlebarGrip.Width / 2), rect.Y
+ 111

```

```

    End Sub

    ' This method handles the RenderGrip event.
Protected Overrides Sub OnRenderGrip(e As ToolStripGripRenderEventArgs)
    DrawTitleBar(e.Graphics, New Rectangle(0, 0, e.ToolStrip.Width, 7))
End Sub

    ' This method handles the RenderToolStripBorder event.
Protected Overrides Sub OnRenderToolStripBorder(e As ToolStripRenderEventArgs)
    DrawTitleBar(e.Graphics, New Rectangle(0, 0, e.ToolStrip.Width, 7))
End Sub

    ' This method handles the RenderButtonBackground event.
Protected Overrides Sub OnRenderButtonBackground(e As ToolStripItemRenderEventArgs)
    Dim g As Graphics = e.Graphics
    Dim bounds As New Rectangle(Point.Empty, e.Item.Size)

    Dim gradientBegin As Color = Color.FromArgb(203, 225, 252)
    Dim gradientEnd As Color = Color.FromArgb(125, 165, 224)

    Dim button As ToolStripButton = CType(e.Item, ToolStripButton)

    If button.Pressed OrElse button.Checked Then
        gradientBegin = Color.FromArgb(254, 128, 62)
        gradientEnd = Color.FromArgb(255, 223, 154)
    ElseIf button.Selected Then
        gradientBegin = Color.FromArgb(255, 255, 222)
        gradientEnd = Color.FromArgb(255, 203, 136)
    End If

    Dim b = New LinearGradientBrush(bounds, gradientBegin, gradientEnd, LinearGradientMode.Vertical)
    Try
        g.FillRectangle(b, bounds)
    Finally
        b.Dispose()
    End Try

    e.Graphics.DrawRectangle(SystemPens.ControlDarkDark, bounds)

    g.DrawLine(SystemPens.ControlDarkDark, bounds.X, bounds.Y, bounds.Width - 1, bounds.Y)

    g.DrawLine(SystemPens.ControlDarkDark, bounds.X, bounds.Y, bounds.X, bounds.Height - 1)

    Dim toolStrip As ToolStrip = button.Owner
    Dim nextItem As ToolStripButton = CType(button.Owner.GetItemAt(button.Bounds.X,
button.Bounds.Bottom + 1), ToolStripButton)

    If nextItem Is Nothing Then
        g.DrawLine(SystemPens.ControlDarkDark, bounds.X, bounds.Height - 1, bounds.X + bounds.Width -
1, bounds.Height - 1)
    End If
    End Sub
End Class

```

2. In the `StackView` control's constructor, create a new instance of the `StackRenderer` class and assign this instance to the `stackStrip` control's `Renderer` property.

```
public StackView()
{
    this.InitializeComponent();

    // Assign icons to ToolStripButton controls.
    this.InitializeImages();

    // Set up renderers.
    this.stackStrip.Renderer = new StackRenderer();
}
```

```
Public Sub New()
    Me.InitializeComponent()

    ' Assign icons to ToolStripButton controls.
    Me.InitializeImages()

    ' Set up renderers.
    Me.stackStrip.Renderer = New StackRenderer()
End Sub
```

Testing the StackView Control

The `StackView` control derives from the [UserControl](#) class. Therefore, you can test the control with the **UserControl Test Container**. For more information, see [How to: Test the Run-Time Behavior of a UserControl](#).

To test the StackView control

1. Press F5 to build the project and start the **UserControl Test Container**.
2. Move the pointer over the buttons of the `StackView` control, and then click a button to see the appearance of its selected state.

Next Steps

In this walkthrough, you have created a reusable custom client control with the professional appearance of an Office XP control. You can use the [ToolStrip](#) family of controls for many other purposes:

- Create shortcut menus for your controls with [ContextMenuStrip](#). For more information, see [ContextMenu Component Overview](#).
- Create a form with an automatically populated standard menu. For more information, see [Walkthrough: Providing Standard Menu Items to a Form](#).
- Create a multiple document interface (MDI) form with docking [ToolStrip](#) controls. For more information, see [How to: Create an MDI Form with Menu Merging and ToolStrip Controls](#).

See Also

[MenuStrip](#)

[ToolStrip](#)

[StatusStrip](#)

[ToolStrip Control](#)

[How to: Provide Standard Menu Items to a Form](#)

Walkthrough: Creating an MDI Form with Menu Merging and ToolStrip Controls

5/4/2018 • 6 min to read • [Edit Online](#)

The `System.Windows.Forms` namespace supports multiple document interface (MDI) applications, and the `MenuStrip` control supports menu merging. MDI forms can also `ToolStrip` controls.

This walkthrough demonstrates how to use `ToolStripPanel` controls with an MDI form. The form also supports menu merging with child menus. The following tasks are illustrated in this walkthrough:

- Creating a Windows Forms project.
- Creating the main menu for your form. The actual name of the menu will vary.
- Adding the `ToolStripPanel` control to the **Toolbox**.
- Creating a child form.
- Arranging `ToolStripPanel` controls by z-order.

When you are finished, you will have an MDI form that supports menu merging and movable `ToolStrip` controls.

To copy the code in this topic as a single listing, see [How to: Create an MDI Form with Menu Merging and ToolStrip Controls](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

Prerequisites

In order to complete this walkthrough, you will need:

- Sufficient permissions to be able to create and run Windows Forms application projects on the computer where Visual Studio is installed.

Creating the Project

The first step is to create the project and set up the form.

To create the project

1. Create a Windows Application project called **MdiForm**.

For more information, see [How to: Create a Windows Application Project](#).

2. In the Windows Forms Designer, select the form.
3. In the Properties window, set the value of the `IsMdiContainer` to `true`.

Creating the Main Menu

The parent MDI form contains the main menu. The main menu has one menu item named **Window**. With the

Window menu item, you can create child forms. Menu items from child forms are merged into the main menu.

To create the Main menu

1. From the **Toolbox**, drag a **MenuStrip** control onto the form.
2. Add a **ToolStripMenuItem** to the **MenuStrip** control and name it **Window**.
3. Select the **MenuStrip** control.
4. In the Properties window, set the value of the **MdiWindowListItem** property to **ToolStripMenuItem1**.
5. Add a subitem to the **Window** menu item, and then name the subitem **New**.
6. In the Properties window, click **Events**.
7. Double-click the **Click** event.

The Windows Forms Designer generates an event handler for the **Click** event.

8. Insert the following code into the event handler.

```
// This method creates a new ChildForm instance
// and attaches it to the MDI parent form.
private void newToolStripMenuItem_Click(object sender, EventArgs e)
{
    ChildForm f = new ChildForm();
    f.MdiParent = this;
    f.Text = "Form - " + this.MdiChildren.Length.ToString();
    f.Show();
}
```

```
' This method creates a new ChildForm instance
' and attaches it to the MDI parent form.
Private Sub newToolStripMenuItem_Click(
    ByVal sender As Object, _
    ByVal e As EventArgs) _
Handles newToolStripMenuItem.Click

    Dim f As New ChildForm()
    f.MdiParent = Me
    f.Text = "Form - " + Me.MdiChildren.Length.ToString()
    f.Show()

End Sub
```

Adding the ToolStripPanel Control to the Toolbox

When you use **MenuStrip** controls with an MDI form you must have the **ToolStripPanel** control. You must add the **ToolStripPanel** control to the **Toolbox** to build your MDI form in the Windows Forms Designer.

To add the ToolStripPanel control to the Toolbox

1. Open the **Toolbox**, and then click the **All Windows Forms** tab to show the available Windows Forms controls.
2. Right-click to open the shortcut menu, and select **Choose Items**.
3. In the **Choose Toolbox Items** dialog box, scroll down the **Name** column until you find **ToolStripPanel**.
4. Select the check box by **ToolStripPanel**, and then click **OK**.

The **ToolStripPanel** control appears in the **Toolbox**.

Creating a Child Form

In this procedure, you will define a separate child form class that has its own [MenuStrip](#) control. The menu items for this form are merged with those of the parent form.

To define a child form

1. Add a new form named `ChildForm` to the project.

For more information, see [How to: Add Windows Forms to a Project](#).

2. From the **Toolbox**, drag a [MenuStrip](#) control onto the child form.
3. Click the [MenuStrip](#) control's smart tag glyph (□), and then select **Edit Items**.
4. In the **Items Collection Editor** dialog box, add a new [ToolStripMenuItem](#) named **ChildMenuItem** to the child menu.

For more information, see [ToolStrip Items Collection Editor](#).

Testing the Form

To test your form

1. Press F5 to compile and run your form.
2. Click the **Window** menu item to open the menu, and then click **New**.

A new child form is created in the form's MDI client area. The child form's menu is merged with the main menu.

3. Close the child form.

The child form's menu is removed from the main menu.

4. Click **New** several times.

The child forms are automatically listed under the **Window** menu item because the [MenuStrip](#) control's [MdiWindowListItem](#) property is assigned.

Adding ToolStrip Support

In this procedure, you will add four [ToolStrip](#) controls to the MDI parent form. Each [ToolStrip](#) control is added inside a [ToolStripPanel](#) control, which is docked to the edge of the form.

To add ToolStrip controls to the MDI parent form

1. From the **Toolbox**, drag a [ToolStripPanel](#) control onto the form.
2. With the [ToolStripPanel](#) control selected, double-click the [ToolStrip](#) control in the **Toolbox**.

A [ToolStrip](#) control is created in the [ToolStripPanel](#) control.

3. Select the [ToolStripPanel](#) control.
4. In the Properties window, change the value of the control's [Dock](#) property to **Left**.

The [ToolStripPanel](#) control docks to the left side of the form, underneath the main menu. The MDI client area resizes to fit the [ToolStripPanel](#) control.

5. Repeat steps 1 through 4.

Dock the new [ToolStripPanel](#) control to the top of the form.

The [ToolStripPanel](#) control is docked underneath the main menu, but to the right of the first [ToolStripPanel](#)

control. This step illustrates the importance of z-order in correctly positioning [ToolStripPanel](#) controls.

6. Repeat steps 1 through 4 for two more [ToolStripPanel](#) controls.

Dock the new [ToolStripPanel](#) controls to the right and bottom of the form.

Arranging ToolStripPanel Controls by Z-order

The position of a docked [ToolStripPanel](#) control on your MDI form is determined by the control's position in the z-order. You can easily arrange the z-order of your controls in the Document Outline window.

To arrange ToolStripPanel controls by Z-order

1. In the **View** menu, click **Other Windows**, and then click **Document Outline**.

The arrangement of your [ToolStripPanel](#) controls from the previous procedure is nonstandard. This is because the z-order is not correct. Use the Document Outline window to change the z-order of the controls.

2. In the Document Outline window, select **ToolStripPanel4**.

3. Click the down-arrow button repeatedly, until **ToolStripPanel4** is at the bottom of the list.

The **ToolStripPanel4** control is docked to the bottom of the form, underneath the other controls.

4. Select **ToolStripPanel2**.

5. Click the down-arrow button one time to position the control third in the list.

The **ToolStripPanel2** control is docked to the top of the form, underneath the main menu and above the other controls.

6. Select various controls in the **Document Outline** window and move them to different positions in the z-order. Note the effect of the z-order on the placement of docked controls. Use **CTRL-Z** or **Undo** on the **Edit** menu to undo your changes.

Checkpoint

To test your form

1. Press F5 to compile and run your form.
2. Click the grip of a [ToolStrip](#) control and drag the control to different positions on the form.

You can drag a [ToolStrip](#) control from one [ToolStripPanel](#) control to another.

Next Steps

In this walkthrough, you have created an MDI parent form with [ToolStrip](#) controls and menu merging. You can use the [ToolStrip](#) family of controls for many other purposes:

- Create shortcut menus for your controls with [ContextMenuStrip](#). For more information, see [ContextMenu Component Overview](#).
- Created a form with an automatically populated standard menu. For more information, see [Walkthrough: Providing Standard Menu Items to a Form](#).
- Give your [ToolStrip](#) controls a professional appearance. For more information, see [How to: Set the ToolStrip Renderer for an Application](#).

See Also

[MenuStrip](#)

[ToolStrip](#)

[StatusStrip](#)

[How to: Create MDI Parent Forms](#)

[How to: Create MDI Child Forms](#)

[How to: Insert a ToolStrip into an MDI Drop-Down Menu](#)

[ToolStrip Control](#)

ToolStripContainer Control

5/4/2018 • 1 min to read • [Edit Online](#)

[ToolStrip](#) controls feature built-in rafting (sharing of horizontal or vertical space within the tool area when docked) by using the [ToolStripContainer](#).

The topics in this section describe the concepts and techniques that you can use to build [ToolStripContainer](#) features into your applications.

In This Section

[ToolStripContainer Control Overview](#)

Provides topics that describe the purpose and main concepts of the Windows Forms [ToolStripContainer](#) control.

[How to: Add a ToolStripContainer to a Form](#)

Demonstrates adding a [ToolStripContainer](#) to an application and adding a control to a specific panel of the [ToolStripContainer](#).

[How to: Add a Control to a ToolStripContentPanel](#)

Demonstrates adding a control to the [ToolStripContentPanel](#).

Reference

[ToolStripContainer](#)

Provides reference documentation for the [ToolStripContainer](#) control.

[ToolStripContentPanel](#)

Provides reference documentation for the [ToolStripContentPanel](#) of a [ToolStripContainer](#) control.

Also see [ToolStripContainer Tasks Dialog Box](#).

Related Sections

[ToolStripPanel](#)

Provides reference documentation for the [ToolStripPanel](#) control.

See Also

[Controls to Use on Windows Forms](#)

ToolStripContainer Control Overview

5/4/2018 • 1 min to read • [Edit Online](#)

A [ToolStripContainer](#) has panels on its left, right, top, and bottom sides for positioning and hosting [ToolStrip](#), [MenuStrip](#), and [StatusStrip](#) controls. Multiple [ToolStrip](#) controls stack vertically if you put them in the left or right [ToolStripContainer](#). They stack horizontally if you put them in the top or bottom [ToolStripContainer](#). You can use the central [ToolStripContentPanel](#) of the [ToolStripContainer](#) to position traditional controls on the form.

Any or all [ToolStripContainer](#) controls are directly selectable at design time and can be deleted. Each panel of a [ToolStripContainer](#) is expandable and collapsible, and resizes with the controls that it contains.

Important ToolStripContainer Members

NAME	DESCRIPTION
BottomToolStripPanel	Gets the bottom panel of the ToolStripContainer .
BottomToolStripPanelVisible	Gets or sets a value indicating whether the bottom panel of the ToolStripContainer is visible.
LeftToolStripPanel	Gets the left panel of the ToolStripContainer .
LeftToolStripPanelVisible	Gets or sets a value indicating whether the left panel of the ToolStripContainer is visible.
RightToolStripPanel	Gets the right panel of the ToolStripContainer .
RightToolStripPanelVisible	Gets or sets a value indicating whether the right panel of the ToolStripContainer is visible.
TopToolStripPanel	Gets the top panel of the ToolStripContainer .
TopToolStripPanelVisible	Gets or sets a value indicating whether the top panel of the ToolStripContainer is visible.

See Also

[ToolStripContainer](#)

[ToolStripContentPanel](#)

How to: Add a ToolStripContainer to a Form

5/4/2018 • 1 min to read • [Edit Online](#)

You can programmatically add a [ToolStripContainer](#) to a Windows Form and populate it with controls.

Example

The following code example demonstrates how to add a [ToolStripContainer](#) and a [ToolStrip](#) to a Windows Forms, how to add items to the [ToolStrip](#), and how to add the [ToolStrip](#) to the [TopToolStripPanel](#) of the [ToolStripContainer](#).

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

public class Form1 : Form
{
    private ToolStripContainer toolStripContainer1;
    private ToolStrip toolStrip1;

    public Form1()
    {
        InitializeComponent();
    }

    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.Run(new Form1());
    }

    private void InitializeComponent()
    {
        toolStripContainer1 = new System.Windows.Forms.ToolStripContainer();
        toolStrip1 = new System.Windows.Forms.ToolStrip();
        // Add items to the ToolStrip.
        toolStrip1.Items.Add("One");
        toolStrip1.Items.Add("Two");
        toolStrip1.Items.Add("Three");
        // Add the ToolStrip to the top panel of the ToolStripContainer.
        toolStripContainer1.TopToolStripPanel.Controls.Add(toolStrip1);
        // Add the ToolStripContainer to the form.
        Controls.Add(toolStripContainer1);

    }
}
```

```

Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Text
Imports System.Windows.Forms

Public Class Form1
    Inherits Form
    Private toolStripContainer1 As ToolStripContainer
    Private toolStrip1 As ToolStrip

    Public Sub New()
        InitializeComponent()
    End Sub 'New

    <STAThread()> _
    Shared Sub Main()
        Application.EnableVisualStyles()
        Application.Run(New Form1())
    End Sub 'Main

    Private Sub InitializeComponent()
        toolStripContainer1 = New System.Windows.Forms.ToolStripContainer()
        toolStrip1 = New System.Windows.Forms.ToolStrip()
        ' Add items to the ToolStrip.
        toolStrip1.Items.Add("One")
        toolStrip1.Items.Add("Two")
        toolStrip1.Items.Add("Three")
        ' Add the ToolStrip to the top panel of the ToolStripContainer.
        toolStripContainer1.TopToolStripPanel.Controls.Add(toolStrip1)
        ' Add the ToolStripContainer to the form.
        Controls.Add(toolStripContainer1)
    End Sub 'InitializeComponent
End Class 'Form1

```

Compiling the Code

This code example requires:

- References to the System.Drawing, System.Text, and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. See also [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#) or [ToolStripContainer Tasks Dialog Box](#).

See Also

[ToolStripContainer](#)

[ToolStripContainer Control](#)

[ToolStrip Control](#)

How to: Add a Control to a ToolStripContentPanel

5/4/2018 • 1 min to read • [Edit Online](#)

You can programmatically add one or more controls to a [ToolStripContentPanel](#).

Example

The following code example demonstrates how to add a [RichTextBox](#) to a [ToolStripContentPanel](#).

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

public class Form1 : Form
{
    private ToolStripContainer tsc;
    private RichTextBox rtb;

    public Form1()
    {
        InitializeComponent();
    }

    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.Run(new Form1());
    }

    private void InitializeComponent()
    {
        this.tsc = new System.Windows.Forms.ToolStripContainer();
        this.rtb = new System.Windows.Forms.RichTextBox();
        this.tsc.ContentPanel.Controls.Add(this.rtb);
        this.Controls.Add(this.tsc);

    }
}
```

```
Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Text
Imports System.Windows.Forms

Public Class Form1
    Inherits Form
    Private tsc As ToolStripContainer
    Private rtb As RichTextBox
    Public Sub New()
        InitializeComponent()
    End Sub

    <STAThread()> _
    Shared Sub Main()
        Application.EnableVisualStyles()
        Application.Run(New Form1())
    End Sub

    Private Sub InitializeComponent()
        Me.tsc = New System.Windows.Forms.ToolStripContainer()
        Me.rtb = New System.Windows.Forms.RichTextBox()
        Me.tsc.ContentPanel.Controls.Add(Me.rtb)
        Me.Controls.Add(tsc)
    End Sub
End Class
```

Compiling the Code

This code example requires:

- References to the System, System.Data and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. See also [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#) or [ToolStripContainer Tasks Dialog Box](#).

See Also

[ToolStripContentPanel](#)

[ToolStripContainer](#)

[ToolStripContainer Control](#)

[ToolStrip Control](#)

ToolStripPanel Control

5/4/2018 • 1 min to read • [Edit Online](#)

`ToolStripPanel` control enables the sharing of horizontal or vertical space within the tool area when docked and arranging of `ToolStrip` controls when you do not need the four panels and central panel of a `ToolStripContainer`.

The topics in this section describe the concepts and techniques that you can use to build `ToolStripPanel` features into your applications.

In This Section

[ToolStripPanel Control Overview](#)

Provides topics that describe the purpose and main concepts of the Windows Forms `ToolStripContainer` control.

[How to: Join ToolStripPanels](#)

Demonstrates adding `ToolStrip` controls to a `ToolStripPanel`.

[How to: Use ToolStripPanels for MDI](#)

Demonstrates the flexibility afforded by `ToolStripPanel` controls in a Multiple Document Interface application.

Reference

[ToolStripPanel](#)

Provides reference documentation for the `ToolStripPanel` control.

See Also

[Controls to Use on Windows Forms](#)

ToolStripPanel Control Overview

5/4/2018 • 1 min to read • [Edit Online](#)

A [ToolStripPanel](#) provides a single area for positioning and hosting [ToolStrip](#), [MenuStrip](#), and [StatusStrip](#) controls. Multiple [ToolStrip](#) controls stack vertically or horizontally depending on the [Orientation](#) of the [ToolStripPanel](#).

Important ToolStripPanel Members

NAME	DESCRIPTION
Orientation	Gets or sets a value indicating the horizontal or vertical orientation of the ToolStripPanel .
Renderer	Gets or sets a ToolStripRenderer used to customize the appearance of a ToolStripPanel .
RenderMode	Gets or sets the painting styles to be applied to the ToolStripPanel .
RowMargin	Gets or sets the spacing, in pixels, between the ToolStripPanelRow and the ToolStripPanel .
Rows	Gets the ToolStripPanelRow in this ToolStripPanel .
Join	Adds a ToolStrip to a ToolStripPanel .

See Also

[ToolStripContainer](#)
[ToolStripContentPanel](#)
[ToolStrip Sample](#)

How to: Join ToolStripPanels

5/4/2018 • 1 min to read • [Edit Online](#)

You can join [ToolStrip](#) controls to a [ToolStripPanel](#) at run time, which provides the flexibility of multiple-document interface (MDI) applications.

Example

The following code example demonstrates how to join [ToolStrip](#) controls to a [ToolStripPanel](#).

```
// Create ToolStripPanel controls.  
ToolStripPanel tspTop = new ToolStripPanel();  
ToolStripPanel tspBottom = new ToolStripPanel();  
ToolStripPanel tspLeft = new ToolStripPanel();  
ToolStripPanel tspRight = new ToolStripPanel();  
  
// Dock the ToolStripPanel controls to the edges of the form.  
tspTop.Dock = DockStyle.Top;  
tspBottom.Dock = DockStyle.Bottom;  
tspLeft.Dock = DockStyle.Left;  
tspRight.Dock = DockStyle.Right;  
  
// Create ToolStrip controls to move among the  
// ToolStripPanel controls.  
  
// Create the "Top" ToolStrip control and add  
// to the corresponding ToolStripPanel.  
ToolStrip tsTop = new ToolStrip();  
tsTop.Items.Add("Top");  
tspTop.Join(tsTop);  
  
// Create the "Bottom" ToolStrip control and add  
// to the corresponding ToolStripPanel.  
ToolStrip tsBottom = new ToolStrip();  
tsBottom.Items.Add("Bottom");  
tspBottom.Join(tsBottom);  
  
// Create the "Right" ToolStrip control and add  
// to the corresponding ToolStripPanel.  
ToolStrip tsRight = new ToolStrip();  
tsRight.Items.Add("Right");  
tspRight.Join(tsRight);  
  
// Create the "Left" ToolStrip control and add  
// to the corresponding ToolStripPanel.  
ToolStrip tsLeft = new ToolStrip();  
tsLeft.Items.Add("Left");  
tspLeft.Join(tsLeft);
```

```

' Create ToolStripPanel controls.
Dim tspTop As New ToolStripPanel()
Dim tspBottom As New ToolStripPanel()
Dim tspLeft As New ToolStripPanel()
Dim tspRight As New ToolStripPanel()

' Dock the ToolStripPanel controls to the edges of the form.
tspTop.Dock = DockStyle.Top
tspBottom.Dock = DockStyle.Bottom
tspLeft.Dock = DockStyle.Left
tspRight.Dock = DockStyle.Right

' Create ToolStrip controls to move among the
' ToolStripPanel controls.
' Create the "Top" ToolStrip control and add
' to the corresponding ToolStripPanel.
Dim tsTop As New ToolStrip()
tsTop.Items.Add("Top")
tspTop.Join(tsTop)

' Create the "Bottom" ToolStrip control and add
' to the corresponding ToolStripPanel.
Dim tsBottom As New ToolStrip()
tsBottom.Items.Add("Bottom")
tspBottom.Join(tsBottom)

' Create the "Right" ToolStrip control and add
' to the corresponding ToolStripPanel.
Dim tsRight As New ToolStrip()
tsRight.Items.Add("Right")
tspRight.Join(tsRight)

' Create the "Left" ToolStrip control and add
' to the corresponding ToolStripPanel.
Dim tsLeft As New ToolStrip()
tsLeft.Items.Add("Left")
tspLeft.Join(tsLeft)

```

Compiling the Code

This example requires:

- References to the System.Design, System.Drawing, and System.Windows.Forms assemblies.

For information about building this example from the command line for visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[ToolStrip](#)

[ToolStripPanel](#)

[How to: Use ToolStripPanels for MDI](#)

How to: Use ToolStripPanels for MDI

5/4/2018 • 3 min to read • [Edit Online](#)

The [ToolStripPanel](#) provides flexibility for multiple-document interface (MDI) applications by using the [Join](#) method.

Example

The following code example demonstrates how to use [ToolStripPanel](#) controls for MDI.

```
// This code example demonstrates how to use ToolStripPanel
// controls with a multiple document interface (MDI).
public class Form1 : Form
{
    public Form1()
    {
        // Make the Form an MDI parent.
        this.IsMdiContainer = true;

        // Create ToolStripPanel controls.
        ToolStripPanel tspTop = new ToolStripPanel();
        ToolStripPanel tspBottom = new ToolStripPanel();
        ToolStripPanel tspLeft = new ToolStripPanel();
        ToolStripPanel tspRight = new ToolStripPanel();

        // Dock the ToolStripPanel controls to the edges of the form.
        tspTop.Dock = DockStyle.Top;
        tspBottom.Dock = DockStyle.Bottom;
        tspLeft.Dock = DockStyle.Left;
        tspRight.Dock = DockStyle.Right;

        // Create ToolStrip controls to move among the
        // ToolStripPanel controls.

        // Create the "Top" ToolStrip control and add
        // to the corresponding ToolStripPanel.
        ToolStrip tsTop = new ToolStrip();
        tsTop.Items.Add("Top");
        tspTop.Join(tsTop);

        // Create the "Bottom" ToolStrip control and add
        // to the corresponding ToolStripPanel.
        ToolStrip tsBottom = new ToolStrip();
        tsBottom.Items.Add("Bottom");
        tspBottom.Join(tsBottom);

        // Create the "Right" ToolStrip control and add
        // to the corresponding ToolStripPanel.
        ToolStrip tsRight = new ToolStrip();
        tsRight.Items.Add("Right");
        tspRight.Join(tsRight);

        // Create the "Left" ToolStrip control and add
        // to the corresponding ToolStripPanel.
        ToolStrip tsLeft = new ToolStrip();
        tsLeft.Items.Add("Left");
        tspLeft.Join(tsLeft);

        // Create a MenuStrip control with a new window.
        MenuStrip ms = new MenuStrip();
        ToolStripMenuItem windowMenu = new ToolStripMenuItem("Window");
    }
}
```

```

    ToolStripMenuItem windowNewMenu = new ToolStripMenuItem("New", null, new
EventHandler(windowNewMenu_Click));
    windowMenu.DropDownItems.Add(windowNewMenu);
    ((ToolStripDropDownMenu)(windowMenu.DropDown)).ShowImageMargin = false;
    ((ToolStripDropDownMenu)(windowMenu.DropDown)).ShowCheckMargin = true;

    // Assign the ToolStripMenuItem that displays
    // the list of child forms.
    ms.MdiWindowListItem = windowMenu;

    // Add the window ToolStripMenuItem to the MenuStrip.
    ms.Items.Add(windowMenu);

    // Dock the MenuStrip to the top of the form.
    ms.Dock = DockStyle.Top;

    // The Form.MainMenuStrip property determines the merge target.
    this.MainMenuStrip = ms;

    // Add the ToolStripPanels to the form in reverse order.
    this.Controls.Add(tspRight);
    this.Controls.Add(tspLeft);
    this.Controls.Add(tspBottom);
    this.Controls.Add(tspTop);

    // Add the MenuStrip last.
    // This is important for correct placement in the z-order.
    this.Controls.Add(ms);
}

// This event handler is invoked when
// the "New" ToolStripMenuItem is clicked.
// It creates a new Form and sets its MdiParent
// property to the main form.
void windowNewMenu_Click(object sender, EventArgs e)
{
    Form f = new Form();
    f.MdiParent = this;
    f.Text = "Form - " + this.MdiChildren.Length.ToString();
    f.Show();
}
}

```

```

' This code example demonstrates how to use ToolStripPanel
' controls with a multiple document interface (MDI).
Public Class Form1
    Inherits Form

    Public Sub New()
        ' Make the Form an MDI parent.
        Me.IsMdiContainer = True

        ' Create ToolStripPanel controls.
        Dim tspTop As New ToolStripPanel()
        Dim tspBottom As New ToolStripPanel()
        Dim tspLeft As New ToolStripPanel()
        Dim tspRight As New ToolStripPanel()

        ' Dock the ToolStripPanel controls to the edges of the form.
        tspTop.Dock = DockStyle.Top
        tspBottom.Dock = DockStyle.Bottom
        tspLeft.Dock = DockStyle.Left
        tspRight.Dock = DockStyle.Right

        ' Create ToolStrip controls to move among the
        ' ToolStripPanel controls.
        ' Create the "Top" ToolStrip control and add

```

```

' to the corresponding ToolStripPanel.
Dim tsTop As New ToolStrip()
tsTop.Items.Add("Top")
tspTop.Join(tsTop)

' Create the "Bottom" ToolStrip control and add
' to the corresponding ToolStripPanel.
Dim tsBottom As New ToolStrip()
tsBottom.Items.Add("Bottom")
tspBottom.Join(tsBottom)

' Create the "Right" ToolStrip control and add
' to the corresponding ToolStripPanel.
Dim tsRight As New ToolStrip()
tsRight.Items.Add("Right")
tspRight.Join(tsRight)

' Create the "Left" ToolStrip control and add
' to the corresponding ToolStripPanel.
Dim tsLeft As New ToolStrip()
tsLeft.Items.Add("Left")
tspLeft.Join(tsLeft)

' Create a MenuStrip control with a new window.
Dim ms As New MenuStrip()
Dim windowMenu As New ToolStripMenuItem("Window")
Dim windowNewMenu As New ToolStripMenuItem("New", Nothing, New EventHandler(AddressOf
windowNewMenu_Click))
windowMenu.DropDownItems.Add(windowNewMenu)
 CType(windowMenu.DropDown, ToolStripDropDownMenu).ShowImageMargin = False
 CType(windowMenu.DropDown, ToolStripDropDownMenu).ShowCheckMargin = True

' Assign the ToolStripMenuItem that displays
' the list of child forms.
ms.MdiWindowListItem = windowMenu

' Add the window ToolStripMenuItem to the MenuStrip.
ms.Items.Add(windowMenu)

' Dock the MenuStrip to the top of the form.
ms.Dock = DockStyle.Top

' The Form.MainMenuStrip property determines the merge target.
Me.MainMenuStrip = ms

' Add the ToolStripPanels to the form in reverse order.
Me.Controls.Add(tspRight)
Me.Controls.Add(tspLeft)
Me.Controls.Add(tspBottom)
Me.Controls.Add(tspTop)

' Add the MenuStrip last.
' This is important for correct placement in the z-order.
Me.Controls.Add(ms)
End Sub

' This event handler is invoked when
' the "New" ToolStripMenuItem is clicked.
' It creates a new Form and sets its MdiParent
' property to the main form.
Private Sub windowNewMenu_Click(ByVal sender As Object, ByVal e As EventArgs)
    Dim f As New Form()
    f.MdiParent = Me
    f.Text = "Form - " + Me.MdiChildren.Length.ToString()
    f.Show()
End Sub
End Class

```

Compiling the Code

This example requires:

- References to the System.Design, System.Drawing, and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[ToolStripPanel](#)

[How to: Join ToolStripPanels](#)

ToolStripProgressBar Control

5/4/2018 • 1 min to read • [Edit Online](#)

The [ToolStripProgressBar](#) combines [ToolStrip](#) rendering and rafting features with its typical process-tracking functionality.

In This Section

[ToolStripProgressBar Control Overview](#)

Provides topics that describe the purpose and main concepts of the Windows Forms [ToolStripProgressBar](#) control.

Reference

[ToolStripPanel](#)

Provides reference documentation for the [ToolStripPanel](#) control.

[ToolStripProgressBar](#)

Provides reference documentation for the [ToolStripProgressBar](#) control.

See Also

[Controls to Use on Windows Forms](#)

ToolStripProgressBar Control Overview

5/4/2018 • 1 min to read • [Edit Online](#)

The [ToolStripProgressBar](#) combines the rafting and rendering functionality of all [ToolStrip](#) controls with its typical process-tracking functionality. A [ToolStripProgressBar](#) is most usually hosted by [StatusStrip](#), and less frequently by a [ToolStrip](#).

Although [ToolStripProgressBar](#) replaces and adds functionality to the control in previous versions, [ToolStripProgressBar](#) is retained for both backward compatibility and future use if desired.

Important ToolStripProgressBar Members

NAME	DESCRIPTION
MarqueeAnimationSpeed	Gets or sets a value representing the delay between each Marquee display update, in milliseconds.
Maximum	Gets or sets the upper bound of the range that is defined for this ToolStripProgressBar .
Minimum	Gets or sets the lower bound of the range that is defined for this ToolStripProgressBar .
Style	Gets or sets the style that the ToolStripProgressBar uses to display the progress of an operation.
Value	Gets or sets the current value of the ToolStripProgressBar .
PerformStep	Advances the current position of the progress bar by the amount of the Step property.

See Also

[ToolStripProgressBar](#)

ToolStripStatusLabel Control

5/4/2018 • 1 min to read • [Edit Online](#)

The [ToolStripStatusLabel](#) provides a display area in a [StatusStrip](#) for text, images, or both.

In This Section

[ToolStripStatusLabel Control Overview](#)

Provides topics that describe the purpose and main concepts of the Windows Forms [ToolStripStatusLabel](#) control.

Reference

[ToolStripStatusLabel](#)

Provides reference documentation for the [ToolStripStatusLabel](#) control.

[StatusStrip](#)

Provides reference documentation for the [StatusStrip](#) control.

[ToolStripProgressBar](#)

Provides reference documentation for the [ToolStripProgressBar](#) control.

See Also

[Controls to Use on Windows Forms](#)

ToolStripStatusLabel Control Overview

5/4/2018 • 1 min to read • [Edit Online](#)

The [ToolStripStatusLabel](#) is a label for a [StatusStrip](#). Like the [Label](#) or [ToolStripLabel](#), the [ToolStripStatusLabel](#) provides a non-clickable display area for text, images, or both. The [ToolStripStatusLabel](#) is hosted by a [StatusStrip](#).

Important ToolStripStatusLabel Members

NAME	DESCRIPTION
Spring	Gets or sets a value indicating whether the ToolStripStatusLabel automatically fills the available space on the StatusStrip as the form is resized
BorderSides	Gets or sets a value that indicates which sides of the ToolStripStatusLabel show borders.
BorderStyle	Gets or sets the border style of the ToolStripStatusLabel .

See Also

[ToolStripStatusLabel](#)

ToolTip Component (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [ToolTip](#) component displays text when the user points at controls. A ToolTip can be associated with any control. An example use of this control: In order to save space on a form, you can display a small icon on a button and use a ToolTip to explain the button's function.

In This Section

[ToolTip Component Overview](#)

Introduces the general concepts of the [ToolTip](#) component, which allows users to see text when they point the mouse at a control.

[How to: Set ToolTips for Controls on a Windows Form at Design Time](#)

Describes how to set Tooltips in code or in the designer.

[How to: Change the Delay of the Windows Forms ToolTip Component](#)

Explains how to set values that control how long a Tooltip takes to appear and the length of time for which it is shown.

Reference

[ToolTip class](#)

Provides reference information on the class and its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

[Control Help Using ToolTips](#)

Discusses Tooltips as a way to make brief, specialized Help messages for individual controls on Windows Forms.

ToolTip Component Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [ToolTip](#) component displays text when the user points at controls. A ToolTip can be associated with any control. An example use of this component: to save space on a form, you can display a small icon on a button and use a ToolTip to explain the button's function.

Working with the ToolTip Component

A [ToolTip](#) component provides a `ToolTip` property to multiple controls on a Windows Form or other container. For example, if you place one [ToolTip](#) component on a form, you can display "Type your name here" for a [TextBox](#) control and "Click here to save changes" for a [Button](#) control.

The key methods of the [ToolTip](#) component are [SetToolTip](#) and [GetToolTip](#). You can use the [SetToolTip](#) method to set the ToolTips displayed for controls. For more information, see [How to: Set ToolTips for Controls on a Windows Form at Design Time](#). The key properties are [Active](#), which must be set to `true` for the ToolTip to appear, and [AutomaticDelay](#), which sets the length of time that the ToolTip string is shown, how long the user must point at the control for the ToolTip to appear, and how long it takes for subsequent ToolTip windows to appear. For more information, see [How to: Change the Delay of the Windows Forms ToolTip Component](#).

See Also

[ToolTip](#)

[How to: Set ToolTips for Controls on a Windows Form at Design Time](#)

[How to: Change the Delay of the Windows Forms ToolTip Component](#)

How to: Set ToolTips for Controls on a Windows Form at Design Time

5/4/2018 • 1 min to read • [Edit Online](#)

You can set a [ToolTip](#) string in code or in the Windows Forms Designer. For more information about the [ToolTip](#) component, see [ToolTip Component Overview](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To set a ToolTip programmatically

1. Add the control that will display the ToolTip.
2. Use the [SetToolTip](#) method of the [ToolTip](#) component.

```
' In this example, Button1 is the control to display the ToolTip.  
ToolTip1.SetToolTip(Button1, "Save changes")
```

```
// In this example, button1 is the control to display the ToolTip.  
toolTip1.SetToolTip(button1, "Save changes");
```

```
// In this example, button1 is the control to display the ToolTip.  
toolTip1->SetToolTip(button1, "Save changes");
```

To set a ToolTip in the designer

1. Add a [ToolTip](#) component to the form.
2. Select the control that will display the ToolTip, or add it to the form.
3. In the **Properties** window, set the **ToolTip on ToolTip1** value to an appropriate string of text.

See Also

[ToolTip Component Overview](#)

[How to: Change the Delay of the Windows Forms ToolTip Component](#)

[ToolTip Component](#)

How to: Change the Delay of the Windows Forms ToolTip Component

5/4/2018 • 1 min to read • [Edit Online](#)

There are multiple delay values that you can set for a Windows Forms `ToolTip` component. The unit of measure for all these properties is milliseconds. The `InitialDelay` property determines how long the user must point at the associated control for the ToolTip string to appear. The `ReshowDelay` property sets the number of milliseconds it takes for subsequent ToolTip strings to appear as the mouse moves from one ToolTip-associated control to another. The `AutoPopDelay` property determines the length of time the ToolTip string is shown. You can set these values individually, or by setting the value of the `AutomaticDelay` property; the other delay properties are set based on the value assigned to the `AutomaticDelay` property. For example, when `AutomaticDelay` is set to a value N , `InitialDelay` is set to N , `ReshowDelay` is set to the value of `AutomaticDelay` divided by five (or $N/5$), and `AutoPopDelay` is set to a value that is five times the value of the `AutomaticDelay` property (or $5N$).

To set the delay

1. Set the following properties as shown in this example.

```
ToolTip1.InitialDelay = 500  
ToolTip1.ReshowDelay = 100  
ToolTip1.AutoPopDelay = 5000
```

```
ToolTip1.InitialDelay = 500;  
ToolTip1.ReshowDelay = 100;  
ToolTip1.AutoPopDelay = 5000;
```

```
toolTip1->InitialDelay = 500;  
toolTip1->ReshowDelay = 100;  
toolTip1->AutoPopDelay = 5000;
```

See Also

[ToolTip Component Overview](#)

[How to: Set ToolTips for Controls on a Windows Form at Design Time](#)

[ToolTip Component](#)

TrackBar Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `TrackBar` control (also sometimes called a "slider" control) is used for navigating through a large amount of information or for visually adjusting a numeric setting. The `TrackBar` control has two parts: the thumb, also known as a slider, and the tick marks. The thumb is the part that can be adjusted. Its position corresponds to the `Value` property. The tick marks are visual indicators that are spaced at regular intervals. The track bar moves in increments that you specify and can be aligned horizontally or vertically. An example use of a track bar would be for setting cursor blink rate or mouse speed.

In This Section

[TrackBar Control Overview](#)

Introduces the general concepts of the `TrackBar` control, which allows users to navigate through information by visually adjusting a numeric setting.

Reference

[TrackBar class](#)

Provides reference information on the class and its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

TrackBar Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [TrackBar](#) control (also sometimes called a "slider" control) is used for navigating through a large amount of information or for visually adjusting a numeric setting. The [TrackBar](#) control has two parts: the thumb, also known as a slider, and the tick marks. The thumb is the part that can be adjusted. Its position corresponds to the [Value](#) property. The tick marks are visual indicators that are spaced at regular intervals. The trackbar moves in increments that you specify and can be aligned horizontally or vertically. For example, you might use the track bar to control the cursor blink rate or mouse speed for a system.

Key Properties

The key properties of the [TrackBar](#) control are [Value](#), [TickFrequency](#), [Minimum](#), and [Maximum](#). [TickFrequency](#) is the spacing of the ticks. [Minimum](#) and [Maximum](#) are the smallest and largest values that can be represented on the track bar.

Two other important properties are [SmallChange](#) and [LargeChange](#). The value of the [SmallChange](#) property is the number of positions the thumb moves in response to having the LEFT or RIGHT ARROW key pressed. The value of the [LargeChange](#) property is the number of positions the thumb moves in response to having the PAGE UP or PAGE DOWN key pressed, or in response to mouse clicks on the track bar on either side of the thumb.

See Also

[TrackBar](#)

[TrackBar Control](#)

TreeView Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `TreeView` control displays a hierarchy of nodes, like the way files and folders are displayed in the left pane of the Windows Explorer feature in Windows operating systems.

In This Section

[TreeView Control Overview](#)

Explains what the control is and its key features and properties.

[How to: Add and Remove Nodes with the Windows Forms TreeView Control](#)

Gives instructions for adding and remove nodes from a tree view.

[How to: Add Custom Information to a TreeView or ListView Control \(Windows Forms\)](#)

Gives instructions for deriving an item in a list view or a node in a tree view to add any fields, methods, or constructors you need.

[How to: Determine Which TreeView Node Was Clicked](#)

Gives instructions for determining which node in a tree view was clicked, so the application can respond appropriately.

[How to: Iterate Through All Nodes of a Windows Forms TreeView Control](#)

Gives instructions for examining every node in a tree view.

[How to: Set Icons for the Windows Forms TreeView Control](#)

Gives instructions for displaying icons for the nodes of a tree view.

[How to: Attach a ShortCut Menu to a TreeView Node](#)

Demonstrates how to add a shortcut menu to a tree view node.

Also see [How to: Add and Remove Nodes with the Windows Forms TreeView Control Using the Designer](#), [How to: Attach a Shortcut Menu to a TreeNode Using the Designer](#).

Reference

[TreeView class](#)

Describes this class and has links to all its members.

Related Sections

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

TreeView Control Overview (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

With the Windows Forms [TreeView](#) control, you can display a hierarchy of nodes to users, like the way files and folders are displayed in the left pane of the Windows Explorer feature of the Windows operating system. Each node in the tree view might contain other nodes, called *child nodes*. You can display parent nodes, or nodes that contain child nodes, as expanded or collapsed. You can also display a tree view with check boxes next to the nodes by setting the tree view's [Checkboxes](#) property to `true`. You can then programmatically select or clear nodes by setting the node's [Checked](#) property to `true` or `false`.

Key Properties

The key properties of the [TreeView](#) control are [Nodes](#) and [SelectedNode](#). The [Nodes](#) property contains the list of top-level nodes in the tree view. The [SelectedNode](#) property sets the currently selected node. You can display icons next to the nodes. The control uses images from the [ImageList](#) named in the tree view's [ImageList](#) property. The [ImageIndex](#) property sets the default image for nodes in the tree view. For more information about displaying images, see [How to: Set Icons for the Windows Forms TreeView Control](#). If you are using Visual Studio 2005, you have access to a large library of standard images that you can use with the [TreeView](#) control.

See Also

[TreeView](#)

[TreeView Control](#)

[How to: Set Icons for the Windows Forms TreeView Control](#)

[How to: Add and Remove Nodes with the Windows Forms TreeView Control](#)

[How to: Iterate Through All Nodes of a Windows Forms TreeView Control](#)

[How to: Determine Which TreeView Node Was Clicked](#)

[How to: Add Custom Information to a TreeView or ListView Control \(Windows Forms\)](#)

How to: Add and Remove Nodes with the Windows Forms TreeView Control

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [TreeView](#) control stores the top-level nodes in its [Nodes](#) collection. Each [TreeNode](#) also has its own [Nodes](#) collection to store its child nodes. Both collection properties are of type [TreeNodeCollection](#), which provides standard collection members that enable you to add, remove, and rearrange the nodes at a single level of the node hierarchy.

To add nodes programmatically

1. Use the [Add](#) method of the tree view's [Nodes](#) property.

```
' Adds new node as a child node of the currently selected node.  
Dim newNode As TreeNode = New TreeNode("Text for new node")  
TreeView1.SelectedNode.Nodes.Add(newNode)
```

```
// Adds new node as a child node of the currently selected node.  
TreeNode newNode = new TreeNode("Text for new node");  
treeView1.SelectedNode.Nodes.Add(newNode);
```

```
// Adds new node as a child node of the currently selected node.  
TreeNode ^ newNode = new TreeNode("Text for new node");  
treeView1->SelectedNode->Nodes->Add(newNode);
```

To remove nodes programmatically

1. Use the [Remove](#) method of the tree view's [Nodes](#) property to remove a single node, or the [Clear](#) method to clear all nodes.

```
' Removes currently selected node, or root if nothing is selected.  
TreeView1.Nodes.Remove(TreeView1.SelectedNode)  
' Clears all nodes.  
TreeView1.Nodes.Clear()
```

```
// Removes currently selected node, or root if nothing  
// is selected.  
treeView1.Nodes.Remove(treeView1.SelectedNode);  
// Clears all nodes.  
TreeView1.Nodes.Clear();
```

```
// Removes currently selected node, or root if nothing  
// is selected.  
treeView1->Nodes->Remove(treeView1->SelectedNode);  
// Clears all nodes.  
treeView1->Nodes->Clear();
```

See Also

[TreeView Control](#)

[TreeView Control Overview](#)

[How to: Set Icons for the Windows Forms TreeView Control](#)

[How to: Iterate Through All Nodes of a Windows Forms TreeView Control](#)

[How to: Determine Which TreeView Node Was Clicked](#)

[How to: Add Custom Information to a TreeView or ListView Control \(Windows Forms\)](#)

How to: Add and Remove Nodes with the Windows Forms TreeView Control Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

Because the Windows Forms [TreeView](#) control displays nodes in a hierarchical manner, when adding a node you must pay attention to what its parent node is.

The following procedure requires a **Windows Application** project with a form containing a [TreeView](#) control. For information about setting up such a project, see [How to: Create a Windows Application Project](#) and [How to: Add Controls to Windows Forms](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To add or remove nodes in the designer

1. Select the [TreeView](#) control.
2. In the **Properties** window, click the **Ellipsis (...**) button next to the [Nodes](#) property.

The **TreeNode Editor** appears.

3. To add nodes, a root node must exist; if one does not exist, you must first add a root by clicking the **Add Root** button. You can then add child nodes by selecting the root or any other node and clicking the **Add Child** button.
4. To delete nodes, select the node to delete and then click the **Delete** button.

See Also

[TreeView Control](#)

[TreeView Control Overview](#)

[How to: Set Icons for the Windows Forms TreeView Control](#)

[How to: Iterate Through All Nodes of a Windows Forms TreeView Control](#)

[How to: Determine Which TreeView Node Was Clicked](#)

[How to: Add Custom Information to a TreeView or ListView Control \(Windows Forms\)](#)

How to: Attach a ShortCut Menu to a TreeView Node

5/4/2018 • 2 min to read • [Edit Online](#)

The Windows Forms [TreeView](#) control displays a hierarchy of nodes, similar to the files and folders displayed in the left pane of Windows Explorer. By setting the [ContextMenuStrip](#) property, you can provide context-sensitive operations to the user when they right-click the [TreeView](#) control. By associating a [ContextMenuStrip](#) component with individual [TreeNode](#) items, you can add a customized level of shortcut menu functionality to your [TreeView](#) controls.

To associate a shortcut menu with a TreeNode programmatically

1. Instantiate a [TreeView](#) control with the appropriate property settings, create a root [TreeNode](#), and then add subnodes.
2. Instantiate a [ContextMenuStrip](#) component, and then add a [ToolStripMenuItem](#) for each operation you want to make available at run time.
3. Set the [ContextMenuStrip](#) property of the appropriate [TreeNode](#) to the shortcut menu you create.
4. When this property is set, the shortcut menu will be displayed when you right-click the node.

The following code example creates a basic [TreeView](#) and [ContextMenuStrip](#) associated with the root [TreeNode](#) of the [TreeView](#). You will need to customize the menu choices to those that fit the [TreeView](#) you are developing. Additionally, you will want to write code to handle the [Click](#) events for these menu items.

```
// Declare the TreeView and ContextMenuStrip
private:
    TreeView^ menuTreeView;
private:
    System::Windows::Forms::ContextMenuStrip^ docMenu;

public:
    void InitializeMenuTreeView()
    {
        // Create the TreeView.
        menuTreeView = gcnew TreeView();
        menuTreeView->Size = System::Drawing::Size(200, 200);

        // Create the root node.
        TreeNode^ docNode = gcnew TreeNode("Documents");

        // Add some additional nodes.
        docNode->Nodes->Add("phoneList.doc");
        docNode->Nodes->Add("resume.doc");

        // Add the root nodes to the TreeView.
        menuTreeView->Nodes->Add(docNode);

        // Create the ContextMenuStrip.
        docMenu = gcnew System::Windows::Forms::ContextMenuStrip();

        //Create some menu items.
        ToolStripMenuItem^ openLabel = gcnew ToolStripMenuItem();
        openLabel->Text = "Open";
        ToolStripMenuItem^ deleteLabel = gcnew ToolStripMenuItem();
        deleteLabel->Text = "Delete";
        ToolStripMenuItem^ renameLabel = gcnew ToolStripMenuItem();
        renameLabel->Text = "Rename";

        //Add the menu items to the menu.
        docMenu->Items->AddRange(gcnew array<ToolStripMenuItem^>{openLabel,
            deleteLabel, renameLabel});

        // Set the ContextMenuStrip property to the ContextMenuStrip.
        docNode->ContextMenuStrip = docMenu;

        // Add the TreeView to the form.
        this->Controls->Add(menuTreeView);
    }
}
```

```
// Declare the TreeView and ContextMenuStrip
private TreeView menuTreeView;
private ContextMenuStrip docMenu;

public void InitializeMenuTreeView()
{
    // Create the TreeView.
    menuTreeView = new TreeView();
    menuTreeView.Size = new Size(200, 200);

    // Create the root node.
    TreeNode docNode = new TreeNode("Documents");

    // Add some additional nodes.
    docNode.Nodes.Add("phoneList.doc");
    docNode.Nodes.Add("resume.doc");

    // Add the root nodes to the TreeView.
    menuTreeView.Nodes.Add(docNode);

    // Create the ContextMenuStrip.
    docMenu = new ContextMenuStrip();

    //Create some menu items.
    ToolStripMenuItem openLabel = new ToolStripMenuItem();
    openLabel.Text = "Open";
    ToolStripMenuItem deleteLabel = new ToolStripMenuItem();
    deleteLabel.Text = "Delete";
    ToolStripMenuItem renameLabel = new ToolStripMenuItem();
    renameLabel.Text = "Rename";

    //Add the menu items to the menu.
    docMenu.Items.AddRange(new ToolStripMenuItem[]{openLabel,
        deleteLabel, renameLabel});

    // Set the ContextMenuStrip property to the ContextMenuStrip.
    docNode.ContextMenuStrip = docMenu;

    // Add the TreeView to the form.
    this.Controls.Add(menuTreeView);
}
```

```
' Declare the TreeView and ContextMenuStrip
Private menuTreeView As TreeView
Private docMenu As ContextMenuStrip

Public Sub InitializeMenuTreeView()

    ' Create the TreeView.
    menuTreeView = New TreeView()
    menuTreeView.Size = New Size(200, 200)

    ' Create the root node.
    Dim docNode As New TreeNode("Documents")

    ' Add some additional nodes.
    docNode.Nodes.Add("phoneList.doc")
    docNode.Nodes.Add("resume.doc")

    ' Add the root nodes to the TreeView.
    menuTreeView.Nodes.Add(docNode)

    ' Create the ContextMenuStrip.
    docMenu = New ContextMenuStrip()

    'Create some menu items.
    Dim openLabel As New ToolStripMenuItem()
    openLabel.Text = "Open"
    Dim deleteLabel As New ToolStripMenuItem()
    deleteLabel.Text = "Delete"
    Dim renameLabel As New ToolStripMenuItem()
    renameLabel.Text = "Rename"

    'Add the menu items to the menu.
    docMenu.Items.AddRange(New ToolStripMenuItem() _
        {openLabel, deleteLabel, renameLabel})

    ' Set the ContextMenuStrip property to the ContextMenuStrip.
    docNode.ContextMenuStrip = docMenu

    ' Add the TreeView to the form.
    Me.Controls.Add(menuTreeView)

End Sub
```

See Also

[ContextMenuStrip](#)
[TreeView Control](#)

How to: Attach a Shortcut Menu to a TreeNode Using the Designer

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [TreeView](#) control displays a hierarchy of nodes, similar to the files and folders displayed in the left pane of the Windows Explorer feature in Windows operating systems. By setting the [ContextMenuStrip](#) property, you can provide context-sensitive operations to the user when they right-click the [TreeView](#) control. By associating a [ContextMenuStrip](#) component with individual [TreeNode](#) items, you can add a customized level of shortcut menu functionality to your [TreeView](#) controls.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To associate a shortcut menu with a TreeNode at design time

1. Add a [TreeView](#) control to your form, and then add nodes to the [TreeView](#) as needed. For more information, see [How to: Add and Remove Nodes with the Windows Forms TreeView Control](#).
2. Add a [ContextMenuStrip](#) component to your form, and then add menu items to the shortcut menu that represent node-level operations you wish to make available at run time. For more information, see [How to: Add Menu Items to a ContextMenuStrip](#).
3. Reopen the **TreeNodeEditor** dialog box for the [TreeView](#) control, select the node to edit, and set its [ContextMenuStrip](#) property to the shortcut menu that you added.
4. When this property is set, the shortcut menu will be displayed when you right-click the node.

Additionally, you will want to write code to handle the [Click](#) events for these menu items.

See Also

- [TreeView Control](#)
- [TreeView Control Overview](#)
- [ContextMenuStrip Control](#)

How to: Add Custom Information to a TreeView or ListView Control (Windows Forms)

5/4/2018 • 2 min to read • [Edit Online](#)

You can create a derived node in a Windows Forms [TreeView](#) control or a derived item in a [ListView](#) control. Derivation allows you to add any fields you require, as well as custom methods and constructors for handling them. One use of this feature is to attach a Customer object to each tree node or list item. The examples here are for a [TreeView](#) control, but the same approach can be used for a [ListView](#) control.

To derive a tree node

- Create a new node class, derived from the [TreeNode](#) class, which has a custom field to record a file path.

```
Class myTreeNode
    Inherits TreeNode

    Public FilePath As String

    Sub New(ByVal fp As String)
        MyBase.New()
        FilePath = fp
        Me.Text = fp.Substring(fp.LastIndexOf("\"))
    End Sub
End Class
```

```
class myTreeNode : TreeNode
{
    public string FilePath;

    public myTreeNode(string fp)
    {
        FilePath = fp;
        this.Text = fp.Substring(fp.LastIndexOf("\\"));
    }
}
```

```
ref class myTreeNode : public TreeNode
{
public:
    System::String ^ FilePath;

    myTreeNode(System::String ^ fp)
    {
        FilePath = fp;
        this->Text = fp->Substring(fp->LastIndexOf("\\"));
    }
};
```

To use a derived tree node

1. You can use the new derived tree node as a parameter to function calls.

In the example below, the path set for the location of the text file is the My Documents folder. This is done because you can assume that most computers running the Windows operating system will include this directory. This also allows users with minimal system access levels to safely run the application.

```

' You should replace the bold text file
' in the sample below with a text file of your own choosing.
TreeView1.Nodes.Add(New myTreeNode (System.Environment.GetFolderPath _
(System.Environment.SpecialFolder.Personal) _
& "\ TextFile.txt" ) )

```

```

// You should replace the bold text file
// in the sample below with a text file of your own choosing.
// Note the escape character used (@) when specifying the path.
treeView1.Nodes.Add(new myTreeNode (System.Environment.GetFolderPath _
(System.Environment.SpecialFolder.Personal) _
+ @"\TextFile.txt" ) );

```

```

// You should replace the bold text file
// in the sample below with a text file of your own choosing.
treeView1->Nodes->Add(new myTreeNode(String::Concat(
    System::Environment::GetFolderPath
    (System::Environment::SpecialFolder::Personal),
    "\TextFile.txt")));

```

2. If you are passed the tree node and it is typed as a [TreeNode](#) class, then you will need to cast to your derived class. Casting is an explicit conversion from one type of object to another. For more information on casting, see [Implicit and Explicit Conversions](#) (Visual Basic), [\(\) Operator](#) (Visual C#), or [Cast Operator: \(\)](#) (Visual C++).

```

Public Sub TreeView1_AfterSelect(ByVal sender As Object, ByVal e As
System.Windows.Forms.TreeViewEventArgs) Handles TreeView1.AfterSelect
    Dim mynode As myTreeNode
    mynode = CType(e.node, myTreeNode)
    MessageBox.Show("Node selected is " & mynode.filepath)
End Sub

```

```

protected void treeView1_AfterSelect (object sender,
System.Windows.Forms.TreeViewEventArgs e)
{
    myTreeNode myNode = (myTreeNode)e.Node;
    MessageBox.Show("Node selected is " + myNode.FilePath);
}

```

```

private:
    System::Void treeView1_AfterSelect(System::Object ^ sender,
        System::Windows::Forms::TreeViewEventArgs ^ e)
    {
        myTreeNode ^ myNode = safe_cast<myTreeNode^>(e->Node);
        MessageBox::Show(String::Concat("Node selected is ",
            myNode->FilePath));
    }

```

See Also

[TreeView Control](#)

[ListView Control](#)

How to: Determine Which TreeView Node Was Clicked (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

When working with the Windows Forms [TreeView](#) control, a common task is to determine which node was clicked, and respond appropriately.

To determine which TreeView node was clicked

1. Use the [EventArgs](#) object to return a reference to the clicked node object.
2. Determine which node was clicked by checking the [TreeViewEventArgs](#) class, which contains data related to the event.

```
Private Sub TreeView1_AfterSelect(ByVal sender As System.Object, _  
    ByVal e As System.Windows.Forms.TreeViewEventArgs) Handles TreeView1.AfterSelect  
    ' Determine by checking the Node property of the TreeViewEventArgs.  
    MessageBox.Show(e.Node.Text)  
End Sub
```

```
protected void treeView1_AfterSelect (object sender,  
System.Windows.Forms.TreeViewEventArgs e)  
{  
    // Determine by checking the Text property.  
    MessageBox.Show(e.Node.Text);  
}
```

```
private:  
    void treeView1_AfterSelect(System::Object ^ sender,  
        System::Windows::Forms::TreeViewEventArgs ^ e)  
    {  
        // Determine by checking the Text property.  
        MessageBox::Show(e->Node->Text);  
    }
```

NOTE

As an alternative, you can use the [MouseEventArgs](#) of the [MouseDown](#) or [MouseUp](#) event to get the [X](#) and [Y](#) coordinate values of the [Point](#) where the click occurred. Then, use the [TreeView](#) control's [GetNodeAt](#) method to determine which node was clicked.

See Also

[TreeView Control](#)

How to: Iterate Through All Nodes of a Windows Forms TreeView Control

5/4/2018 • 1 min to read • [Edit Online](#)

It is sometimes useful to examine every node in a Windows Forms `TreeView` control in order to perform some calculation on the node values. This operation can be done using a recursive procedure (recursive method in C# and C++) that iterates through each node in each collection of the tree.

Each `TreeNode` object in a tree view has properties that you can use to navigate the tree view: `FirstNode`, `LastNode`, `NextNode`, `PrevNode`, and `Parent`. The value of the `Parent` property is the parent node of the current node. The child nodes of the current node, if there are any, are listed in its `Nodes` property. The `TreeView` control itself has the `TopNode` property, which is the root node of the entire tree view.

To iterate through all nodes of the `TreeView` control

1. Create a recursive procedure (recursive method in C# and C++) that tests each node.
2. Call the procedure.

The following example shows how to print each `TreeNode` object's `Text` property:

```
Private Sub PrintRecursive(ByVal n As TreeNode)
    System.Diagnostics.Debug.WriteLine(n.Text)
    MessageBox.Show(n.Text)
    Dim aNode As TreeNode
    For Each aNode In n.Nodes
        PrintRecursive(aNode)
    Next
End Sub

' Call the procedure using the top nodes of the treeview.
Private Sub CallRecursive(ByVal aTreeView As TreeView)
    Dim n As TreeNode
    For Each n In aTreeView.Nodes
        PrintRecursive(n)
    Next
End Sub
```

```
private void PrintRecursive(TreeNode treeNode)
{
    // Print the node.
    System.Diagnostics.Debug.WriteLine(treeNode.Text);
    MessageBox.Show(treeNode.Text);
    // Print each node recursively.
    foreach (TreeNode tn in treeNode.Nodes)
    {
        PrintRecursive(tn);
    }
}

// Call the procedure using the TreeView.
private void CallRecursive(TreeView treeView)
{
    // Print each node recursively.
    TreeNodeCollection nodes = treeView.Nodes;
    foreach (TreeNode n in nodes)
    {
        PrintRecursive(n);
    }
}
```

```

private:
    void PrintRecursive( TreeNode^ treeNode )
    {
        // Print the node.
        System::Diagnostics::Debug::WriteLine( treeNode->Text );
        MessageBox::Show( treeNode->Text );

        // Print each node recursively.
        System::Collections::IEnumerator^ myNodes = (safe_cast<System::Collections::IEnumerable^>
(treeNode->Nodes))->GetEnumerator();
        try
        {
            while ( myNodes->MoveNext() )
            {
                TreeNode^ tn = safe_cast<TreeNode^>(myNodes->Current);
                PrintRecursive( tn );
            }
        }
        finally
        {
            IDisposable^ disposable = dynamic_cast<System::IDisposable^>(myNodes);
            if ( disposable != nullptr )
                disposable->Dispose();
        }
    }

    // Call the procedure using the TreeView.
    void CallRecursive( TreeView^ treeView )
    {
        // Print each node recursively.
        TreeNodeCollection^ nodes = treeView->Nodes;
        System::Collections::IEnumerator^ myNodes = (safe_cast<System::Collections::IEnumerable^>
(nodes))->GetEnumerator();
        try
        {
            while ( myNodes->MoveNext() )
            {
                TreeNode^ n = safe_cast<TreeNode^>(myNodes->Current);
                PrintRecursive( n );
            }
        }
        finally
        {
            IDisposable^ disposable = dynamic_cast<System::IDisposable^>(myNodes);
            if ( disposable != nullptr )
                disposable->Dispose();
        }
    }
}

```

See Also

[TreeView Control](#)

[Recursive Procedures](#)

How to: Set Icons for the Windows Forms TreeView Control

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms [TreeView](#) control can display icons next to each node. The icons are positioned to the immediate left of the node text. To display these icons, you must associate the tree view with an [ImageList](#) control. For more information about image lists, see [ImageList Component](#) and [How to: Add or Remove Images with the Windows Forms ImageList Component](#).

NOTE

A bug in Microsoft .NET Framework version 1.1 prevents images from appearing on [TreeView](#) nodes when your application calls [Application.EnableVisualStyles](#). To work around this bug, call [Application.DoEvents](#) in your `Main` method immediately after calling [EnableVisualStyles](#). This bug is fixed in .NET Framework 2.0.

To display images in a tree view

1. Set the [TreeView](#) control's [ImageList](#) property to the existing [ImageList](#) control you wish to use.

These properties can be set in the designer with the Properties window, or in code.

```
TreeView1.ImageList = ImageList1
```

```
treeView1.ImageList = imageList1;
```

```
treeView1->ImageList = imageList1;
```

2. Set the node's [ImageIndex](#) and [SelectedImageIndex](#) properties. The [ImageIndex](#) property determines the image displayed for the node's normal and expanded states, and the [SelectedImageIndex](#) property determines the image displayed for the node's selected state.

These properties can be set in code, or within the TreeNode Editor. To open the TreeNode Editor, click the ellipsis button () next to the [Nodes](#) property on the Properties window.

```
' (Assumes that ImageList1 contains at least two images and  
' the TreeView control contains a selected image.)  
TreeView1.SelectedNode.ImageIndex = 0  
TreeView1.SelectedNode.SelectedImageIndex = 1
```

```
// (Assumes that imageList1 contains at least two images and  
// the TreeView control contains a selected image.)  
treeView1.SelectedNode.ImageIndex = 0;  
treeView1.SelectedNode.SelectedImageIndex = 1;
```

```
// (Assumes that imageList1 contains at least two images and  
// the TreeView control contains a selected image.)  
treeView1->SelectedNode->ImageIndex = 0;  
treeView1->SelectedNode->SelectedImageIndex = 1;
```

See Also

[TreeView Control Overview](#)

[How to: Add and Remove Nodes with the Windows Forms TreeView Control](#)

[How to: Iterate Through All Nodes of a Windows Forms TreeView Control](#)

[How to: Determine Which TreeView Node Was Clicked](#)

[How to: Add Custom Information to a TreeView or ListView Control \(Windows Forms\)](#)

WebBrowser Control (Windows Forms)

5/4/2018 • 1 min to read • [Edit Online](#)

The Windows Forms `WebBrowser` control hosts Web pages and provides Web browsing capabilities to your application.

In This Section

[WebBrowser Control Overview](#)

Explains what this control is and its key features and properties.

[WebBrowser Security](#)

Explains security issues related to the control.

[How to: Navigate to a URL with the WebBrowser Control](#)

Demonstrates how to use the control to navigate to a specific URL.

[How to: Print with a WebBrowser Control](#)

Demonstrates how to print a Web page without displaying it.

[How to: Add Web Browser Capabilities to a Windows Forms Application](#)

Describes how to initialize the control for use as a Web browser.

[How to: Create an HTML Document Viewer in a Windows Forms Application](#)

Describes how to initialize the control for use as an HTML viewer.

[How to: Implement Two-Way Communication Between DHTML Code and Client Application Code](#)

Describes how to set up two-way communication between your application code and DHTML in a Web page hosted by the control.

[Using the Managed HTML Document Object Model](#)

Provides topics that describe how to manipulate or create HTML pages hosted by the `WebBrowser` control.

Reference

[WebBrowser class](#)

Describes this class and has links to all its members.

[WebBrowserDocumentCompletedEventArgs](#)

Describes this class and has links to all its members.

[WebBrowserDocumentCompletedEventHandler](#)

Describes this delegate.

[WebBrowserEncryptionLevel](#)

Describes this enumeration and all its values.

[WebBrowserNavigatedEventArgs](#)

Describes this class and has links to all its members.

[WebBrowserNavigatedEventHandler](#)

Describes this delegate.

[WebBrowserNavigatingEventArgs](#)

Describes this class and has links to all its members.

[WebBrowserNavigatingEventHandler](#)

Describes this delegate.

[WebBrowserProgressChangedEventArgs](#)

Describes this class and has links to all its members.

[WebBrowserProgressChangedEventHandler](#)

Describes this delegate.

[WebBrowserReadyState](#)

Describes this enumeration and all its values.

[WebBrowserRefreshOption](#)

Describes this enumeration and all its values.

See Also

[Controls to Use on Windows Forms](#)

WebBrowser Control Overview

5/4/2018 • 2 min to read • [Edit Online](#)

The [WebBrowser](#) control provides a managed wrapper for the WebBrowser ActiveX control. The managed wrapper lets you display Web pages in your Windows Forms client applications. You can use the [WebBrowser](#) control to duplicate Internet Explorer Web browsing functionality in your application or you can disable default Internet Explorer functionality and use the control as a simple HTML document viewer. You can also use the control to add DHTML-based user interface elements to your form and hide the fact that they are hosted in the [WebBrowser](#) control. This approach lets you seamlessly combine Web controls with Windows Forms controls in a single application.

Frequently Used Properties, Methods, and Events

The [WebBrowser](#) control has several properties, methods, and events that you can use to implement controls found in Internet Explorer. For example, you can use the `Navigate` method to implement an address bar, and the `GoBack`, `GoForward`, `Stop`, and `Refresh` methods to implement navigation buttons on a toolbar. You can handle the `Navigated` event to update the address bar with the value of the `Url` property and the title bar with the value of the `DocumentTitle` property.

If you want to generate your own page content within your application, you can set the `DocumentText` property. If you are familiar with the HTML document object model (DOM), you can also manipulate the contents of the current Web page through the `Document` property. With this property, you can store and modify documents in memory instead of navigating among files.

The `Document` property also lets you call methods implemented in Web page scripting code from your client application code. To access your client application code from your scripting code, set the `ObjectForScripting` property. The object that you specify can be accessed by your script code as the `window.external` object.

NAME	DESCRIPTION
Document property	Gets an object that provides managed access to the HTML document object model (DOM) of the current Web page.
DocumentCompleted event	Occurs when a Web page finishes loading.
DocumentText property	Gets or sets the HTML content of the current Web page.
DocumentTitle property	Gets the title of the current Web page.
GoBack method	Navigates to the previous page in history.
GoForward method	Navigates to the next page in history.
Navigate method	Navigates to the specified URL.
Navigating event	Occurs before navigation begins, enabling the action to be canceled.
ObjectForScripting property	Gets or sets an object that Web page scripting code can use to communicate with your application.

NAME	DESCRIPTION
Print method	Prints the current Web page.
Refresh method	Reloads the current Web page.
Stop method	Halts the current navigation and stops dynamic page elements such as sounds and animation.
Url property	Gets or sets the URL of the current Web page. Setting this property navigates the control to the new URL.

See Also

[WebBrowser](#)

[WebBrowserDocumentCompletedEventArgs](#)

[WebBrowserDocumentCompletedEventHandler](#)

[WebBrowserEncryptionLevel](#)

[WebBrowserNavigatedEventArgs](#)

[WebBrowserNavigatedEventHandler](#)

[WebBrowserNavigatingEventArgs](#)

[WebBrowserNavigatingEventHandler](#)

[WebBrowserProgressChangedEventArgs](#)

[WebBrowserReadyState](#)

[WebBrowserRefreshOption](#)

[How to: Navigate to a URL with the WebBrowser Control](#)

[How to: Print with a WebBrowser Control](#)

[How to: Add Web Browser Capabilities to a Windows Forms Application](#)

[How to: Create an HTML Document Viewer in a Windows Forms Application](#)

[How to: Implement Two-Way Communication Between DHTML Code and Client Application Code](#)

[WebBrowser Security](#)

WebBrowser Security

5/4/2018 • 1 min to read • [Edit Online](#)

The [WebBrowser](#) control is designed to work in full trust only. The HTML content displayed in the control can come from external Web servers and may contain unmanaged code in the form of scripts or Web controls. If you use the [WebBrowser](#) control in this situation, the control is no less secure than Internet Explorer would be, but the managed [WebBrowser](#) control does not prevent such unmanaged code from running.

For more information about security issues relating to the underlying ActiveX `WebBrowser` control, see [WebBrowser Control](#).

See Also

[WebBrowser](#)

[WebBrowser Control Overview](#)

[WebBrowser Control](#)

How to: Navigate to a URL with the WebBrowser Control

5/4/2018 • 1 min to read • [Edit Online](#)

The following code example demonstrates how to navigate the [WebBrowser](#) control to a specific URL.

To determine when the new document is fully loaded, handle the [DocumentCompleted](#) event. For a demonstration of this event, see [How to: Print with a WebBrowser Control](#).

Example

```
Me.webBrowser1.Navigate("http://www.microsoft.com")
```

```
this.webBrowser1.Navigate("http://www.microsoft.com");
```

Compiling the Code

This example requires:

- A [WebBrowser](#) control named `webBrowser1`.
- References to the `System` and `System.Windows.Forms` assemblies.

See Also

[WebBrowser](#)

[WebBrowser.DocumentCompleted](#)

[WebBrowser.Navigating](#)

[WebBrowser.Navigated](#)

[WebBrowser Control](#)

[How to: Print with a WebBrowser Control](#)

How to: Print with a WebBrowser Control

5/4/2018 • 1 min to read • [Edit Online](#)

The following code example demonstrates how to use the [WebBrowser](#) control to print a Web page without displaying it.

Example

```
private void PrintHelpPage()
{
    // Create a WebBrowser instance.
    WebBrowser webBrowserForPrinting = new WebBrowser();

    // Add an event handler that prints the document after it loads.
    webBrowserForPrinting.DocumentCompleted +=
        new WebBrowserDocumentCompletedEventHandler(PrintDocument);

    // Set the Url property to load the document.
    webBrowserForPrinting.Url = new Uri(@"\\myshare\help.html");
}

private void PrintDocument(object sender,
    WebBrowserDocumentCompletedEventArgs e)
{
    // Print the document now that it is fully loaded.
    ((WebBrowser)sender).Print();

    // Dispose the WebBrowser now that the task is complete.
    ((WebBrowser)sender).Dispose();
}
```

```
Private Sub PrintHelpPage()

    ' Create a WebBrowser instance.
    Dim webBrowserForPrinting As New WebBrowser()

    ' Add an event handler that prints the document after it loads.
    AddHandler webBrowserForPrinting.DocumentCompleted, New _
        WebBrowserDocumentCompletedEventHandler(AddressOf PrintDocument)

    ' Set the Url property to load the document.
    webBrowserForPrinting.Url = New Uri("\\myshare\help.html")

End Sub

Private Sub PrintDocument(ByVal sender As Object, _
    ByVal e As WebBrowserDocumentCompletedEventArgs)

    Dim webBrowserForPrinting As WebBrowser = CType(sender, WebBrowser)

    ' Print the document now that it is fully loaded.
    webBrowserForPrinting.Print()
    MessageBox.Show("print")

    ' Dispose the WebBrowser now that the task is complete.
    webBrowserForPrinting.Dispose()

End Sub
```

Compiling the Code

This example requires:

- References to the `System` and `System.Windows.Forms` assemblies.

See Also

[WebBrowser](#)

[Print](#)

[Url](#)

[How to: Navigate to a URL with the WebBrowser Control](#)

[How to: Add Web Browser Capabilities to a Windows Forms Application](#)

[How to: Create an HTML Document Viewer in a Windows Forms Application](#)

[WebBrowser Control Overview](#)

[WebBrowser Security](#)

How to: Add Web Browser Capabilities to a Windows Forms Application

5/4/2018 • 17 min to read • [Edit Online](#)

With the [WebBrowser](#) control, you can add Web browser functionality to your application. The control works like a Web browser by default. After you load an initial URL by setting the [Url](#) property, you can navigate by clicking hyperlinks or by using keyboard shortcuts to move backward and forward through navigation history. By default, you can access additional browser functionality through the right-click shortcut menu. You can also open new documents by dropping them onto the control. The [WebBrowser](#) control also has several properties, methods, and events that you can use to implement user interface features similar to those found in Internet Explorer.

The following code example implements an address bar, typical browser buttons, a **File** menu, a status bar, and a title bar that displays the current page title.

Example

```
#using <System.Drawing.dll>
#using <System.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Security::Permissions;

public ref class Form1: public System::Windows::Forms::Form
{
public:
    Form1()
    {
        //This call is required by the Windows Form Designer.
        InitializeComponent();

        //Add any initialization after the InitializeComponent() call
    }

private:
    //NOTE: The following procedure is required by the Windows Form Designer
    //It can be modified using the Windows Form Designer.
    //Do not modify it using the code editor.
    System::Windows::Forms::MainMenu^ MainMenu1;
    System::Windows::Forms::MenuItem^ MenuItemFile;
    System::Windows::Forms::MenuItem^ MenuItemFileSaveAs;
    System::Windows::Forms::MenuItem^ MenuItemFilePageSetup;
    System::Windows::Forms::MenuItem^ MenuItemFilePrint;
    System::Windows::Forms::MenuItem^ MenuItemFilePrintPreview;
    System::Windows::Forms::MenuItem^ MenuItemFileProperties;
    System::Windows::Forms::TextBox^ TextBoxAddress;
    System::Windows::Forms::Button^ ButtonGo;
    System::Windows::Forms::Button^ backButton;
    System::Windows::Forms::Button^ ButtonForward;
    System::Windows::Forms::Button^ ButtonStop;
    System::Windows::Forms::Button^ ButtonRefresh;
    System::Windows::Forms::Button^ ButtonHome;
    System::Windows::Forms::Button^ ButtonSearch;
    System::Windows::Forms::Panel^ Panel1;
    System::Windows::Forms::WebBrowser ^ WebBrowser1;
    System::Windows::Forms::StatusBar^ StatusBar1;
```

```

System::Windows::Forms::MenuItem^ MenuItem1;
System::Windows::Forms::MenuItem^ MenuItem2;
System::Windows::Button^ ButtonPrint;

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
[SecurityPermission(SecurityAction::Demand, Flags=SecurityPermissionFlag::UnmanagedCode)]
void InitializeComponent()
{
    this->MainMenu1 = gcnew System::Windows::Forms::MainMenu;
    this->MenuItemFile = gcnew System::Windows::Forms::MenuItem;
    this->MenuItemFileSaveAs = gcnew System::Windows::Forms::MenuItem;
    this->MenuItem1 = gcnew System::Windows::Forms::MenuItem;
    this->MenuItemFilePageSetup = gcnew System::Windows::Forms::MenuItem;
    this->MenuItemFilePrint = gcnew System::Windows::Forms::MenuItem;
    this->MenuItemFilePrintPreview = gcnew System::Windows::Forms::MenuItem;
    this->MenuItem2 = gcnew System::Windows::Forms::MenuItem;
    this->MenuItemFileProperties = gcnew System::Windows::Forms::MenuItem;
    this->TextBoxAddress = gcnew System::Windows::Forms::TextBox;
    this->ButtonGo = gcnew System::Windows::Forms::Button;
    this->backButton = gcnew System::Windows::Forms::Button;
    this->ButtonForward = gcnew System::Windows::Forms::Button;
    this->ButtonStop = gcnew System::Windows::Forms::Button;
    this->ButtonRefresh = gcnew System::Windows::Forms::Button;
    this->ButtonHome = gcnew System::Windows::Forms::Button;
    this->ButtonSearch = gcnew System::Windows::Forms::Button;
    this->ButtonPrint = gcnew System::Windows::Forms::Button;
    this->Panel1 = gcnew System::Windows::Forms::Panel;
    this->WebBrowser1 = gcnew System::Windows::Forms::WebBrowser;
    this->StatusBar1 = gcnew System::Windows::Forms::StatusBar;
    this->Panel1->SuspendLayout();
    this->SuspendLayout();

    //
    // MainMenu1
    //
    array<System::Windows::Forms::MenuItem^>^temp0 = {this->MenuItemFile};
    this->MainMenu1->MenuItems->AddRange( temp0 );
    this->MainMenu1->Name = "MainMenu1";

    //
    // MenuItemFile
    //
    this->MenuItemFile->Index = 0;
    array<System::Windows::Forms::MenuItem^>^temp1 = {this->MenuItemFileSaveAs,this->MenuItem1,this->MenuItemFilePageSetup,this->MenuItemFilePrint,this->MenuItemFilePrintPreview,this->MenuItem2,this->MenuItemFileProperties};
    this->MenuItemFile->MenuItems->AddRange( temp1 );
    this->MenuItemFile->Name = "MenuItemFile";
    this->MenuItemFile->Text = "&File";

    //
    // MenuItemFileSaveAs
    //
    this->MenuItemFileSaveAs->Index = 0;
    this->MenuItemFileSaveAs->Name = "MenuItemFileSaveAs";
    this->MenuItemFileSaveAs->Text = " Save &As";
    this->MenuItemFileSaveAs->Click += gcnew System::EventHandler( this, &Form1::MenuItemFileSaveAs_Click );

    //
    // MenuItem1
    //
    this->MenuItem1->Index = 1;
    this->MenuItem1->Name = "MenuItem1";
    this->MenuItem1->Text = "-";

    //

```

```

// MenuItemFilePageSetup
//
this->MenuItemFilePageSetup->Index = 2;
this->MenuItemFilePageSetup->Name = "MenuItemFilePageSetup";
this->MenuItemFilePageSetup->Text = "Page Set&up...";
this->MenuItemFilePageSetup->Click += gcnew System::EventHandler( this,
&Form1::MenuItemFilePageSetup_Click );

//
// MenuItemFilePrint
//
this->MenuItemFilePrint->Index = 3;
this->MenuItemFilePrint->Name = "MenuItemFilePrint";
this->MenuItemFilePrint->Text = "&Print...";
this->MenuItemFilePrint->Click += gcnew System::EventHandler( this, &Form1::MenuItemFilePrint_Click );

//
// MenuItemFilePrintPreview
//
this->MenuItemFilePrintPreview->Index = 4;
this->MenuItemFilePrintPreview->Name = "MenuItemFilePrintPreview";
this->MenuItemFilePrintPreview->Text = "Print Pre&view...";
this->MenuItemFilePrintPreview->Click += gcnew System::EventHandler( this,
&Form1::MenuItemFilePrintPreview_Click );

//
// MenuItem2
//
this->MenuItem2->Index = 5;
this->MenuItem2->Name = "MenuItem2";
this->MenuItem2->Text = "-";

//
// MenuItemFileProperties
//
this->MenuItemFileProperties->Index = 6;
this->MenuItemFileProperties->Name = "MenuItemFileProperties";
this->MenuItemFileProperties->Text = "P&roperties";
this->MenuItemFileProperties->Click += gcnew System::EventHandler( this,
&Form1::MenuItemFileProperties_Click );

//
// TextBoxAddress
//
this->TextBoxAddress->Location = System::Drawing::Point( 0, 0 );
this->TextBoxAddress->Name = "TextBoxAddress";
this->TextBoxAddress->Size = System::Drawing::Size( 240, 20 );
this->TextBoxAddress->TabIndex = 1;
this->TextBoxAddress->Text = "";
this->TextBoxAddress->KeyDown += gcnew System::Windows::Forms::KeyEventHandler( this,
&Form1::TextBoxAddress_KeyDown );

//
// ButtonGo
//
this->ButtonGo->Location = System::Drawing::Point( 240, 0 );
this->ButtonGo->Name = "ButtonGo";
this->ButtonGo->Size = System::Drawing::Size( 48, 24 );
this->ButtonGo->TabIndex = 2;
this->ButtonGo->Text = "Go";
this->ButtonGo->Click += gcnew System::EventHandler( this, &Form1::ButtonGo_Click );

//
// backButton
//
this->backButton->Location = System::Drawing::Point( 288, 0 );
this->backButton->Name = "backButton";
this->backButton->Size = System::Drawing::Size( 48, 24 );
this->backButton->TabIndex = 3.

```

```

this->backButton->TabIndex = 3;
this->backButton->Text = "Back";
this->backButton->Click += gcnew System::EventHandler( this, &Form1::backButton_Click );

//
// ButtonForward
//
this->ButtonForward->Location = System::Drawing::Point( 336, 0 );
this->ButtonForward->Name = "ButtonForward";
this->ButtonForward->Size = System::Drawing::Size( 48, 24 );
this->ButtonForward->TabIndex = 4;
this->ButtonForward->Text = "Forward";
this->ButtonForward->Click += gcnew System::EventHandler( this, &Form1::ButtonForward_Click );

//
// ButtonStop
//
this->ButtonStop->Location = System::Drawing::Point( 384, 0 );
this->ButtonStop->Name = "ButtonStop";
this->ButtonStop->Size = System::Drawing::Size( 48, 24 );
this->ButtonStop->TabIndex = 5;
this->ButtonStop->Text = "Stop";
this->ButtonStop->Click += gcnew System::EventHandler( this, &Form1::ButtonStop_Click );

//
// ButtonRefresh
//
this->ButtonRefresh->Location = System::Drawing::Point( 432, 0 );
this->ButtonRefresh->Name = "ButtonRefresh";
this->ButtonRefresh->Size = System::Drawing::Size( 48, 24 );
this->ButtonRefresh->TabIndex = 6;
this->ButtonRefresh->Text = "Refresh";
this->ButtonRefresh->Click += gcnew System::EventHandler( this, &Form1::ButtonRefresh_Click );

//
// ButtonHome
//
this->ButtonHome->Location = System::Drawing::Point( 480, 0 );
this->ButtonHome->Name = "ButtonHome";
this->ButtonHome->Size = System::Drawing::Size( 48, 24 );
this->ButtonHome->TabIndex = 7;
this->ButtonHome->Text = "Home";
this->ButtonHome->Click += gcnew System::EventHandler( this, &Form1::ButtonHome_Click );

//
// ButtonSearch
//
this->ButtonSearch->Location = System::Drawing::Point( 528, 0 );
this->ButtonSearch->Name = "ButtonSearch";
this->ButtonSearch->Size = System::Drawing::Size( 48, 24 );
this->ButtonSearch->TabIndex = 8;
this->ButtonSearch->Text = "Search";
this->ButtonSearch->Click += gcnew System::EventHandler( this, &Form1::ButtonSearch_Click );

//
// ButtonPrint
//
this->ButtonPrint->Location = System::Drawing::Point( 576, 0 );
this->ButtonPrint->Name = "ButtonPrint";
this->ButtonPrint->Size = System::Drawing::Size( 48, 24 );
this->ButtonPrint->TabIndex = 9;
this->ButtonPrint->Text = "Print";
this->ButtonPrint->Click += gcnew System::EventHandler( this, &Form1::ButtonPrint_Click );

//
// Panel1
//
this->Panel1->Controls->Add( this->ButtonPrint );
this->Panel1->Controls->Add( this->TextBoxAddress );

```

```

this->Panel1->Controls->Add( this->ButtonHome );
this->Panel1->Controls->Add( this->backButton );
this->Panel1->Controls->Add( this->ButtonForward );
this->Panel1->Controls->Add( this->ButtonStop );
this->Panel1->Controls->Add( this->ButtonRefresh );
this->Panel1->Controls->Add( this->ButtonSearch );
this->Panel1->Dock = System::Windows::Forms::DockStyle::Top;
this->Panel1->Location = System::Drawing::Point( 0, 0 );
this->Panel1->Name = "Panel1";
this->Panel1->Size = System::Drawing::Size( 624, 24 );
this->Panel1->TabIndex = 11;

//
// WebBrowser1
//
this->WebBrowser1->AllowWebBrowserDrop = false;
this->WebBrowser1->ScriptErrorsSuppressed = true;
this->WebBrowser1->WebBrowserShortcutsEnabled = false;
this->WebBrowser1->Dock = System::Windows::Forms::DockStyle::Fill;
this->WebBrowser1->IsWebBrowserContextMenuEnabled = false;

this->WebBrowser1->Location = System::Drawing::Point( 0, 24 );
this->WebBrowser1->Name = "WebBrowser1";
this->WebBrowser1->Size = System::Drawing::Size( 624, 389 );
this->WebBrowser1->TabIndex = 10;
this->WebBrowser1->StatusTextChanged += gcnew System::EventHandler( this,
&Form1::WebBrowser1_StatusTextChanged );
this->WebBrowser1->CanGoBackChanged += gcnew System::EventHandler( this,
&Form1::WebBrowser1_CanGoBackChanged );
this->WebBrowser1->Navigated += gcnew System::Windows::Forms::WebBrowserNavigatedEventHandler( this,
&Form1::WebBrowser1_Navigated );
this->WebBrowser1->CanGoForwardChanged += gcnew System::EventHandler( this,
&Form1::WebBrowser1_CanGoForwardChanged );
this->WebBrowser1->DocumentTitleChanged += gcnew System::EventHandler( this,
&Form1::WebBrowser1_DocumentTitleChanged );

//
// StatusBar1
//
this->StatusBar1->Location = System::Drawing::Point( 0, 413 );
this->StatusBar1->Name = "StatusBar1";
this->StatusBar1->Size = System::Drawing::Size( 624, 16 );
this->StatusBar1->TabIndex = 12;

//
// Form1
//
this->ClientSize = System::Drawing::Size( 624, 429 );
this->Controls->Add( this->WebBrowser1 );
this->Controls->Add( this->Panel1 );
this->Controls->Add( this->StatusBar1 );
this->Menu = this->MainMenu1;
this->Name = "Form1";
this->Text = "WebBrowser Example";
this->Panel1->ResumeLayout( false );
this->ResumeLayout( false );
}

internal:

static property Form1^ GetInstance
{
    Form1^ get()
    {
        if ( m_DefaultInstance == nullptr || m_DefaultInstance->IsDisposed )
        {
            System::Threading::Monitor::Enter( Form1::typeid );

```

```

        try
    {
        if ( m_DefaultInstance == nullptr || m_DefaultInstance->IsDisposed )
        {
            m_DefaultInstance = gcnew Form1;
        }
    }
    finally
    {
        System::Threading::Monitor::Exit( Form1::typeid );
    }
}

return m_DefaultInstance;
}
}

private:
static Form1^ m_DefaultInstance;

// Displays the Save dialog box.
void MenuItemFileSaveAs_Click( System::Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    this->WebBrowser1->ShowSaveAsDialog();
}

// Displays the Page Setup dialog box.
void MenuItemFilePageSetup_Click( System::Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    this->WebBrowser1->ShowPageSetupDialog();
}

// Displays the Print dialog box.
void MenuItemFilePrint_Click( System::Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    this->WebBrowser1->ShowPrintDialog();
}

// Displays the Print Preview dialog box.
void MenuItemFilePrintPreview_Click( System::Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    this->WebBrowser1->ShowPrintPreviewDialog();
}

// Displays the Properties dialog box.
void MenuItemFileProperties_Click( System::Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    this->WebBrowser1->ShowPropertiesDialog();
}

// Navigates to the URL in the address text box when
// the ENTER key is pressed while the text box has focus.
void TextBoxAddress_KeyDown( Object^ /*sender*/, System::Windows::Forms::KeyEventArgs^ e )
{
    if ( e->KeyCode == System::Windows::Forms::Keys::Enter && !this->TextBoxAddress->Text->Equals( "" ) )
    {
        this->WebBrowser1->Navigate( this->TextBoxAddress->Text );
    }
}

// Navigates to the URL in the address text box when
// the Go button is clicked.
void ButtonGo_Click( System::Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    if ( !this->TextBoxAddress->Text->Equals( "" ) )
    {
        this->WebBrowser1->Navigate( this->TextBoxAddress->Text );
    }
}

```

```

// Updates the URL in TextBoxAddress upon navigation.
void WebBrowser1_Navigated( Object^ /*sender*/, System::Windows::Forms::WebBrowserEventArgs^ /*e*/ )
{
{
    this->TextBoxAddress->Text = this->WebBrowser1->Url->ToString();
}

// Navigates WebBrowser1 to the previous page in the history.
void backButton_Click( System::Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    this->WebBrowser1->GoBack();
}

// Disables the Back button at the beginning of the navigation history.
void WebBrowser1_CanGoBackChanged( System::Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    this->backButton->Enabled = this->WebBrowser1->CanGoBack;
}

// Navigates WebBrowser1 to the next page in history.
void ButtonForward_Click( System::Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    this->WebBrowser1->GoForward();
}

// Disables the Forward button at the end of navigation history.
void WebBrowser1_CanGoForwardChanged( System::Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    this->ButtonForward->Enabled = this->WebBrowser1->CanGoForward;
}

// Halts the current navigation and any sounds or animations on
// the page.
void ButtonStop_Click( System::Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    this->WebBrowser1->Stop();
}

// Reloads the current page.
void ButtonRefresh_Click( System::Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    // Skip refresh if about:blank is loaded to avoid removing
    // content specified by the DocumentText property.
    if ( !this->WebBrowser1->Url->Equals( "about:blank" ) )
    {
        this->WebBrowser1->Refresh();
    }
}

// Navigates WebBrowser1 to the home page of the current user.
void ButtonHome_Click( System::Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    this->WebBrowser1->GoHome();
}

// Navigates WebBrowser1 to the search page of the current user.
void ButtonSearch_Click( System::Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    this->WebBrowser1->GoSearch();
}

// Prints the current document using the current print settings.
void ButtonPrint_Click( System::Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    this->WebBrowser1->Print();
}

```

```

// Updates StatusBar1 with the current browser status text.
void WebBrowser1_StatusTextChanged( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    this->StatusBar1->Text = WebBrowser1->StatusText;
}

// Updates the title bar with the current document title.
void WebBrowser1_DocumentTitleChanged( Object^ /*sender*/, System::EventArgs^ /*e*/ )
{
    this->Text = WebBrowser1->DocumentTitle;
}

/// <summary>
/// The main entry point for the application.
/// </summary>

[STAThread]
int main()
{
    System::Windows::Forms::Application::Run( gcnew Form1 );
}

```

```

using System;
using System.Windows.Forms;
using System.Security.Permissions;

[PermissionSet(SecurityAction.Demand, Name = "FullTrust")]
public class Form1 : Form
{
    public Form1()
    {
        // Create the form layout. If you are using Visual Studio,
        // you can replace this code with code generated by the designer.
        InitializeForm();

        // The following events are not visible in the designer, so
        // you must associate them with their event-handlers in code.
        webBrowser1.CanGoBackChanged +=
            new EventHandler(webBrowser1_CanGoBackChanged);
        webBrowser1.CanGoForwardChanged +=
            new EventHandler(webBrowser1_CanGoForwardChanged);
        webBrowser1.DocumentTitleChanged +=
            new EventHandler(webBrowser1_DocumentTitleChanged);
        webBrowser1.StatusTextChanged +=
            new EventHandler(webBrowser1_StatusTextChanged);

        // Load the user's home page.
        webBrowser1.GoHome();
    }

    // Displays the Save dialog box.
    private void saveAsToolStripMenuItem_Click(object sender, EventArgs e)
    {
        webBrowser1.ShowSaveAsDialog();
    }

    // Displays the Page Setup dialog box.
    private void pageSetupToolStripMenuItem_Click(object sender, EventArgs e)
    {
        webBrowser1.ShowPageSetupDialog();
    }

    // Displays the Print dialog box.
    private void printToolStripMenuItem_Click(object sender, EventArgs e)
    {

```

```

        webBrowser1.ShowPrintDialog();
    }

    // Displays the Print Preview dialog box.
    private void printPreviewToolStripMenuItem_Click(
        object sender, EventArgs e)
    {
        webBrowser1.ShowPrintPreviewDialog();
    }

    // Displays the Properties dialog box.
    private void propertiesToolStripMenuItem_Click(
        object sender, EventArgs e)
    {
        webBrowser1.ShowPropertiesDialog();
    }

    // Selects all the text in the text box when the user clicks it.
    private void toolStripTextBox1_Click(object sender, EventArgs e)
    {
        toolStripTextBox1.SelectAll();
    }

    // Navigates to the URL in the address box when
    // the ENTER key is pressed while the ToolStripTextBox has focus.
    private void toolStripTextBox1_KeyDown(object sender, KeyEventArgs e)
    {
        if (e.KeyCode == Keys.Enter)
        {
            Navigate(toolStripTextBox1.Text);
        }
    }

    // Navigates to the URL in the address box when
    // the Go button is clicked.
    private void goButton_Click(object sender, EventArgs e)
    {
        Navigate(toolStripTextBox1.Text);
    }

    // Navigates to the given URL if it is valid.
    private void Navigate(String address)
    {
        if (String.IsNullOrEmpty(address)) return;
        if (address.Equals("about:blank")) return;
        if (!address.StartsWith("http://") &&
            !address.StartsWith("https://"))
        {
            address = "http://" + address;
        }
        try
        {
            webBrowser1.Navigate(new Uri(address));
        }
        catch (System.UriFormatException)
        {
            return;
        }
    }

    // Updates the URL in TextBoxAddress upon navigation.
    private void webBrowser1_Navigated(object sender,
        WebBrowserNavigatedEventArgs e)
    {
        toolStripTextBox1.Text = webBrowser1.Url.ToString();
    }

    // Navigates webBrowser1 to the previous page in the history.
    private void backButton_Click(object sender, EventArgs e)

```

```
{  
    webBrowser1.GoBack();  
}  
  
// Disables the Back button at the beginning of the navigation history.  
private void webBrowser1_CanGoBackChanged(object sender, EventArgs e)  
{  
    backButton.Enabled = webBrowser1.CanGoBack;  
}  
  
// Navigates webBrowser1 to the next page in history.  
private void forwardButton_Click(object sender, EventArgs e)  
{  
    webBrowser1.GoForward();  
}  
  
// Disables the Forward button at the end of navigation history.  
private void webBrowser1_CanGoForwardChanged(object sender, EventArgs e)  
{  
    forwardButton.Enabled = webBrowser1.CanGoForward;  
}  
  
// Halts the current navigation and any sounds or animations on  
// the page.  
private void stopButton_Click(object sender, EventArgs e)  
{  
    webBrowser1.Stop();  
}  
  
// Reloads the current page.  
private void refreshButton_Click(object sender, EventArgs e)  
{  
    // Skip refresh if about:blank is loaded to avoid removing  
    // content specified by the DocumentText property.  
    if (!webBrowser1.Url.Equals("about:blank"))  
    {  
        webBrowser1.Refresh();  
    }  
}  
  
// Navigates webBrowser1 to the home page of the current user.  
private void homeButton_Click(object sender, EventArgs e)  
{  
    webBrowser1.GoHome();  
}  
  
// Navigates webBrowser1 to the search page of the current user.  
private void searchButton_Click(object sender, EventArgs e)  
{  
    webBrowser1.GoSearch();  
}  
  
// Prints the current document using the current print settings.  
private void printButton_Click(object sender, EventArgs e)  
{  
    webBrowser1.Print();  
}  
  
// Updates the status bar with the current browser status text.  
private void webBrowser1_StatusTextChanged(object sender, EventArgs e)  
{  
    toolStripStatusLabel1.Text = webBrowser1.StatusText;  
}  
  
// Updates the title bar with the current document title.  
private void webBrowser1_DocumentTitleChanged(object sender, EventArgs e)  
{  
    this.Text = webBrowser1.DocumentTitle;  
}
```

```
// Exits the application.
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    Application.Exit();
}

// The remaining code in this file provides basic form initialization and
// includes a Main method. If you use the Visual Studio designer to create
// your form, you can use the designer generated code instead of this code,
// but be sure to use the names shown in the variable declarations here,
// and be sure to attach the event handlers to the associated events.

private WebBrowser webBrowser1;

private MenuStrip menuStrip1;
private ToolStripMenuItem fileToolStripMenuItem,
    saveAsToolStripMenuItem, printToolStripMenuItem,
    printPreviewToolStripMenuItem, exitToolStripMenuItem,
    pageSetupToolStripMenuItem, propertiesToolStripMenuItem;
private ToolStripSeparator toolStripSeparator1, toolStripSeparator2;

private ToolStrip toolStrip1, toolStrip2;
private ToolStripTextBox toolStripTextBox1;
private ToolStripButton goButton, backButton,
    forwardButton, stopButton, refreshButton,
    homeButton, searchButton, printButton;

private StatusStrip statusStrip1;
private ToolStripStatusLabel toolStripStatusLabel1;

private void InitializeForm()
{
    webBrowser1 = new WebBrowser();

    menuStrip1 = new MenuStrip();
    fileToolStripMenuItem = new ToolStripMenuItem();
    saveAsToolStripMenuItem = new ToolStripMenuItem();
    toolStripSeparator1 = new ToolStripSeparator();
    printToolStripMenuItem = new ToolStripMenuItem();
    printPreviewToolStripMenuItem = new ToolStripMenuItem();
    toolStripSeparator2 = new ToolStripSeparator();
    exitToolStripMenuItem = new ToolStripMenuItem();
    pageSetupToolStripMenuItem = new ToolStripMenuItem();
    propertiesToolStripMenuItem = new ToolStripMenuItem();

    toolStrip1 = new ToolStrip();
    goButton = new ToolStripButton();
    backButton = new ToolStripButton();
    forwardButton = new ToolStripButton();
    stopButton = new ToolStripButton();
    refreshButton = new ToolStripButton();
    homeButton = new ToolStripButton();
    searchButton = new ToolStripButton();
    printButton = new ToolStripButton();

    toolStrip2 = new ToolStrip();
    toolStripTextBox1 = new ToolStripTextBox();

    statusStrip1 = new StatusStrip();
    toolStripStatusLabel1 = new ToolStripStatusLabel();

    menuStrip1.Items.Add(fileToolStripMenuItem);

    fileToolStripMenuItem.DropDownItems.AddRange(
        new ToolStripItem[] {
            saveAsToolStripMenuItem, toolStripSeparator1,
            pageSetupToolStripMenuItem, printToolStripMenuItem,
            printPreviewToolStripMenuItem, toolStripSeparator2
        }
    );
}
```

```

        printPreviewToolStripMenuItem, toolStripSeparator2,
        propertiesToolStripMenuItem, exitToolStripMenuItem
    });

fileToolStripMenuItem.Text = "&File";
saveAsToolStripMenuItem.Text = "Save &As...";
pageSetupToolStripMenuItem.Text = "Page Set&up...";
printToolStripMenuItem.Text = "&Print...";
printPreviewToolStripMenuItem.Text = "Print Pre&view...";
propertiesToolStripMenuItem.Text = "Properties";
exitToolStripMenuItem.Text = "E&xit";

printToolStripMenuItem.ShortcutKeys = Keys.Control | Keys.P;

saveAsToolStripMenuItem.Click +=
    new System.EventHandler(saveAsToolStripMenuItem_Click);
pageSetupToolStripMenuItem.Click +=
    new System.EventHandler(pageSetupToolStripMenuItem_Click);
printToolStripMenuItem.Click +=
    new System.EventHandler(printToolStripMenuItem_Click);
printPreviewToolStripMenuItem.Click +=
    new System.EventHandler(printPreviewToolStripMenuItem_Click);
propertiesToolStripMenuItem.Click +=
    new System.EventHandler(propertiesToolStripMenuItem_Click);
exitToolStripMenuItem.Click +=
    new System.EventHandler(exitToolStripMenuItem_Click);

toolStrip1.Items.AddRange(new ToolStripItem[] {
    goButton, backButton, forwardButton, stopButton,
    refreshButton, homeButton, searchButton, printButton});

goButton.Text = "Go";
backButton.Text = "Back";
forwardButton.Text = "Forward";
stopButton.Text = "Stop";
refreshButton.Text = "Refresh";
homeButton.Text = "Home";
searchButton.Text = "Search";
printButton.Text = "Print";

backButton.Enabled = false;
forwardButton.Enabled = false;

goButton.Click += new System.EventHandler(goButton_Click);
backButton.Click += new System.EventHandler(backButton_Click);
forwardButton.Click += new System.EventHandler(forwardButton_Click);
stopButton.Click += new System.EventHandler(stopButton_Click);
refreshButton.Click += new System.EventHandler(refreshButton_Click);
homeButton.Click += new System.EventHandler(homeButton_Click);
searchButton.Click += new System.EventHandler(searchButton_Click);
printButton.Click += new System.EventHandler(printButton_Click);

toolStrip2.Items.Add(toolStripTextBox1);
toolStripTextBox1.Size = new System.Drawing.Size(250, 25);
toolStripTextBox1.KeyDown +=
    new KeyEventHandler(toolStripTextBox1_KeyDown);
toolStripTextBox1.Click +=
    new System.EventHandler(toolStripTextBox1_Click);

statusStrip1.Items.Add(toolStripStatusLabel1);

webBrowser1.Dock = DockStyle.Fill;
webBrowser1.Navigated +=
    new WebBrowserNavigatedEventHandler(webBrowser1_Navigated);

Controls.AddRange(new Control[] {
    webBrowser1, toolStrip2, toolStrip1,
    menuStrip1, statusStrip1, menuStrip1 });
}

```

```
[STAThread]
static void Main()
{
    Application.EnableVisualStyles();
    Application.Run(new Form1());
}

}
```

```
Imports System
Imports System.Windows.Forms
Imports System.Security.Permissions

<PermissionSet(SecurityAction.Demand, Name:="FullTrust")> _
Public Class Form1
    Inherits Form

    Public Sub New()

        ' Create the form layout. If you are using Visual Studio,
        ' you can replace this code with code generated by the designer.
        InitializeForm()

        ' Load the user's home page.
        webBrowser1.GoHome()

    End Sub

    ' Displays the Save dialog box.
    Private Sub saveAsToolStripMenuItem_Click( _
        ByVal sender As Object, ByVal e As EventArgs) _
        Handles saveAsToolStripMenuItem.Click

        webBrowser1.ShowSaveAsDialog()

    End Sub

    ' Displays the Page Setup dialog box.
    Private Sub pageSetupToolStripMenuItem_Click( _
        ByVal sender As Object, ByVal e As EventArgs) _
        Handles pageSetupToolStripMenuItem.Click

        webBrowser1.ShowPageSetupDialog()

    End Sub

    ' Displays the Print dialog box.
    Private Sub printToolStripMenuItem_Click( _
        ByVal sender As Object, ByVal e As EventArgs) _
        Handles printToolStripMenuItem.Click

        webBrowser1.ShowPrintDialog()

    End Sub

    ' Displays the Print Preview dialog box.
    Private Sub printPreviewToolStripMenuItem_Click( _
        ByVal sender As Object, ByVal e As EventArgs) _
        Handles printPreviewToolStripMenuItem.Click

        webBrowser1.ShowPrintPreviewDialog()

    End Sub

    ' Displays the Properties dialog box.
    Private Sub propertiesToolStripMenuItem_Click(
```

```

    Private Sub propertiesToolStripMenuItem_Click( _
        ByVal sender As Object, ByVal e As EventArgs) _
        Handles propertiesToolStripMenuItem.Click

        webBrowser1.ShowPropertiesDialog()

    End Sub

    ' Selects all the text in the text box when the user clicks it.
    Private Sub toolStripTextBox1_Click( _
        ByVal sender As Object, ByVal e As EventArgs) _
        Handles toolStripTextBox1.Click

        toolStripTextBox1.SelectAll()

    End Sub

    ' Navigates to the URL in the address box when
    ' the ENTER key is pressed while the ToolStripTextBox has focus.
    Private Sub toolStripTextBox1_KeyDown( _
        ByVal sender As Object, ByVal e As KeyEventArgs) _
        Handles toolStripTextBox1.KeyDown

        If (e.KeyCode = Keys.Enter) Then
            Navigate(toolStripTextBox1.Text)
        End If

    End Sub

    ' Navigates to the URL in the address box when
    ' the Go button is clicked.
    Private Sub goButton_Click( _
        ByVal sender As Object, ByVal e As EventArgs) _
        Handles goButton.Click

        Navigate(toolStripTextBox1.Text)

    End Sub

    ' Navigates to the given URL if it is valid.
    Private Sub Navigate(ByVal address As String)

        If String.IsNullOrEmpty(address) Then Return
        If address.Equals("about:blank") Then Return
        If Not address.StartsWith("http://") And _
            Not address.StartsWith("https://") Then
            address = "http://" & address
        End If

        Try
            webBrowser1.Navigate(New Uri(address))
        Catch ex As System.UriFormatException
            Return
        End Try

    End Sub

    ' Updates the URL in TextBoxAddress upon navigation.
    Private Sub webBrowser1_Navigated(ByVal sender As Object, _
        ByVal e As WebBrowserNavigatedEventArgs) _
        Handles webBrowser1.Navigated

        toolStripTextBox1.Text = webBrowser1.Url.ToString()

    End Sub

    ' Navigates webBrowser1 to the previous page in the history.
    Private Sub backButton_Click( _
        ByVal sender As Object, ByVal e As EventArgs) _
        Handles backButton.Click

```

```

Handles backButton.Click

    webBrowser1.GoBack()

End Sub

' Disables the Back button at the beginning of the navigation history.
Private Sub webBrowser1_CanGoBackChanged( _
    ByVal sender As Object, ByVal e As EventArgs) _
    Handles webBrowser1.CanGoBackChanged

    backButton.Enabled = webBrowser1.CanGoBack

End Sub

' Navigates webBrowser1 to the next page in history.
Private Sub forwardButton_Click( _
    ByVal sender As Object, ByVal e As EventArgs) _
    Handles forwardButton.Click

    webBrowser1.GoForward()

End Sub

' Disables the Forward button at the end of navigation history.
Private Sub webBrowser1_CanGoForwardChanged( _
    ByVal sender As Object, ByVal e As EventArgs) _
    Handles webBrowser1.CanGoForwardChanged

    forwardButton.Enabled = webBrowser1.CanGoForward

End Sub

' Halts the current navigation and any sounds or animations on
' the page.
Private Sub stopButton_Click( _
    ByVal sender As Object, ByVal e As EventArgs) _
    Handles stopButton.Click

    webBrowser1.Stop()

End Sub

' Reloads the current page.
Private Sub refreshButton_Click( _
    ByVal sender As Object, ByVal e As EventArgs) _
    Handles refreshButton.Click

    ' Skip refresh if about:blank is loaded to avoid removing
    ' content specified by the DocumentText property.
    If Not webBrowser1.Url.Equals("about:blank") Then
        webBrowser1.Refresh()
    End If

End Sub

' Navigates webBrowser1 to the home page of the current user.
Private Sub homeButton_Click( _
    ByVal sender As Object, ByVal e As EventArgs) _
    Handles homeButton.Click

    webBrowser1.GoHome()

End Sub

' Navigates webBrowser1 to the search page of the current user.
Private Sub searchButton_Click( _
    ByVal sender As Object, ByVal e As EventArgs) _
    Handles searchButton.Click

```

```

    webBrowser1.GoSearch()

End Sub

' Prints the current document Imports the current print settings.
Private Sub printButton_Click( _
    ByVal sender As Object, ByVal e As EventArgs) _
Handles printButton.Click

    webBrowser1.Print()

End Sub

' Updates the status bar with the current browser status text.
Private Sub webBrowser1_StatusTextChanged( _
    ByVal sender As Object, ByVal e As EventArgs) _
Handles webBrowser1.StatusTextChanged

    toolStripStatusLabel1.Text = webBrowser1.StatusText

End Sub

' Updates the title bar with the current document title.
Private Sub webBrowser1_DocumentTitleChanged( _
    ByVal sender As Object, ByVal e As EventArgs) _
Handles webBrowser1.DocumentTitleChanged

    Me.Text = webBrowser1.DocumentTitle

End Sub

' Exits the application.
Private Sub exitToolStripMenuItem_Click( _
    ByVal sender As Object, ByVal e As EventArgs) _
Handles exitToolStripMenuItem.Click

    Application.Exit()

End Sub

' The remaining code in this file provides basic form initialization and
' includes a Main method. If you use the Visual Studio designer to create
' your form, you can use the designer generated code instead of this code,
' but be sure to use the names shown in the variable declarations here.

Private WithEvents webBrowser1 As WebBrowser

Private menuStrip1 As MenuStrip
Private WithEvents fileToolStripMenuItem, saveAsToolStripMenuItem, _
    printToolStripMenuItem, printPreviewToolStripMenuItem, _
    exitToolStripMenuItem, pageSetupToolStripMenuItem, _
    propertiesToolStripMenuItem As ToolStripMenuItem
Private toolStripSeparator1, toolStripSeparator2 As ToolStripSeparator

Private toolStrip1, toolStrip2 As ToolStrip
Private WithEvents toolStripTextBox1 As ToolStripTextBox
Private WithEvents goButton, backButton, forwardButton, _
    stopButton, refreshButton, homeButton, _
    searchButton, printButton As ToolStripButton

Private statusStrip1 As StatusStrip
Private toolStripStatusLabel1 As ToolStripStatusLabel

Private Sub InitializeForm()

    webBrowser1 = New WebBrowser()

    menuStrip1 = New ToolStrip()

```

```

fileToolStripMenuItem = New ToolStripMenuItem()
saveAsToolStripMenuItem = New ToolStripMenuItem()
toolStripSeparator1 = New ToolStripSeparator()
printToolStripMenuItem = New ToolStripMenuItem()
printPreviewToolStripMenuItem = New ToolStripMenuItem()
toolStripSeparator2 = New ToolStripSeparator()
exitToolStripMenuItem = New ToolStripMenuItem()
pageSetupToolStripMenuItem = New ToolStripMenuItem()
propertiesToolStripMenuItem = New ToolStripMenuItem()

toolStrip1 = New ToolStrip()
goButton = New ToolStripButton()
backButton = New ToolStripButton()
forwardButton = New ToolStripButton()
stopButton = New ToolStripButton()
refreshButton = New ToolStripButton()
homeButton = New ToolStripButton()
searchButton = New ToolStripButton()
printButton = New ToolStripButton()

toolStrip2 = New ToolStrip()
toolStripTextBox1 = New ToolStripTextBox()

statusStrip1 = New StatusStrip()
toolStripStatusLabel1 = New ToolStripStatusLabel()

webView1.Dock = DockStyle.Fill

menuStrip1.Items.Add(fileToolStripMenuItem)

fileToolStripMenuItem.DropDownItems.AddRange( _
    New ToolStripItem() { _
        saveAsToolStripMenuItem, toolStripSeparator1, _
        pageSetupToolStripMenuItem, printToolStripMenuItem, _
        printPreviewToolStripMenuItem, toolStripSeparator2, _
        propertiesToolStripMenuItem, exitToolStripMenuItem _
    })

fileToolStripMenuItem.Text = "&File"
saveAsToolStripMenuItem.Text = "Save &As..."
pageSetupToolStripMenuItem.Text = "Page Set&up..."
printToolStripMenuItem.Text = "&Print..."
printPreviewToolStripMenuItem.Text = "Print Pre&view..."
propertiesToolStripMenuItem.Text = "Properties"
exitToolStripMenuItem.Text = "E&xit"

printToolStripMenuItem.ShortcutKeys = Keys.Control Or Keys.P

toolStrip1.Items.AddRange(New ToolStripItem() { _
    goButton, backButton, forwardButton, stopButton, _
    refreshButton, homeButton, searchButton, printButton})

goButton.Text = "Go"
backButton.Text = "Back"
forwardButton.Text = "Forward"
stopButton.Text = "Stop"
refreshButton.Text = "Refresh"
homeButton.Text = "Home"
searchButton.Text = "Search"
printButton.Text = "Print"

backButton.Enabled = False
forwardButton.Enabled = False

toolStrip2.Items.Add(toolStripTextBox1)
toolStripTextBox1.Size = New System.Drawing.Size(250, 25)

statusStrip1.Items.Add(toolStripStatusLabel1)

```

```
Controls.AddRange(New Control() { _
    webBrowser1, toolStrip2, toolStrip1, _
    menuStrip1, statusStrip1, menuStrip1})
End Sub

<STAThread()> _
Public Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())
End Sub

End Class
```

Compiling the Code

This example requires:

- References to the `System`, `System.Drawing`, and `System.Windows.Forms` assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[WebBrowser](#)

[WebBrowser Control](#)

How to: Create an HTML Document Viewer in a Windows Forms Application

5/4/2018 • 1 min to read • [Edit Online](#)

You can use the [WebBrowser](#) control to display and print HTML documents without providing the full functionality of an Internet Web browser. This is useful when you want to take advantage of the formatting capabilities of HTML but do not want your users to load arbitrary Web pages that may contain untrusted Web controls or potentially malicious script code. You might want to restrict the capability of the [WebBrowser](#) control in this manner, for example, to use it as an HTML email viewer or to provide HTML-formatted help in your application.

To create an HTML document viewer

1. Set the [AllowWebBrowserDrop](#) property to `false` to prevent the [WebBrowser](#) control from opening files dropped onto it.

```
webBrowser1.AllowWebBrowserDrop = false;
```

```
webBrowser1.AllowWebBrowserDrop = False
```

2. Set the [Url](#) property to the location of the initial file to display.

```
webBrowser1.Url = new Uri("http://www.contoso.com/");
```

```
webBrowser1.Url = New Uri("http://www.contoso.com/")
```

Compiling the Code

This example requires:

- A [WebBrowser](#) control named `webBrowser1`.
- References to the `System` and `System.Windows.Forms` assemblies.

See Also

[WebBrowser](#)

[AllowWebBrowserDrop](#)

[Url](#)

[WebBrowser Control Overview](#)

[WebBrowser Security](#)

[How to: Navigate to a URL with the WebBrowser Control](#)

[How to: Print with a WebBrowser Control](#)

How to: Implement Two-Way Communication Between DHTML Code and Client Application Code

5/4/2018 • 4 min to read • [Edit Online](#)

You can use the [WebBrowser](#) control to add existing dynamic HTML (DHTML) Web application code to your Windows Forms client applications. This is useful when you have invested significant development time in creating DHTML-based controls and you want to take advantage of the rich user interface capabilities of Windows Forms without having to rewrite existing code.

The [WebBrowser](#) control lets you implement two-way communication between your client application code and your Web page scripting code through the [ObjectForScripting](#) and [Document](#) properties. Additionally, you can configure the [WebBrowser](#) control so that your Web controls blend seamlessly with other controls on your application form, hiding their DHTML implementation. To seamlessly blend the controls, format the page displayed so that its background color and visual style match the rest of the form, and use the [AllowWebBrowserDrop](#), [IsWebBrowserContextMenuEnabled](#), and [WebBrowserShortcutsEnabled](#) properties to disable standard browser features.

To embed DHTML in your Windows Forms application

1. Set the [WebBrowser](#) control's [AllowWebBrowserDrop](#) property to `false` to prevent the [WebBrowser](#) control from opening files dropped onto it.

```
webBrowser1.AllowWebBrowserDrop = false;
```

```
webBrowser1.AllowWebBrowserDrop = False
```

2. Set the control's [IsWebBrowserContextMenuEnabled](#) property to `false` to prevent the [WebBrowser](#) control from displaying its shortcut menu when the user right-clicks it.

```
webBrowser1.IsWebBrowserContextMenuEnabled = false;
```

```
webBrowser1.IsWebBrowserContextMenuEnabled = False
```

3. Set the control's [WebBrowserShortcutsEnabled](#) property to `false` to prevent the [WebBrowser](#) control from responding to shortcut keys.

```
webBrowser1.WebBrowserShortcutsEnabled = false;
```

```
webBrowser1.WebBrowserShortcutsEnabled = False
```

4. Set the [ObjectForScripting](#) property in the form's constructor or a [Load](#) event handler.

The following code uses the form class itself for the scripting object.

NOTE

Component Object Model (COM) must be able to access the scripting object. To make your form visible to COM, add the [ComVisibleAttribute](#) attribute to your form class.

```
webBrowser1.ObjectForScripting = this;
```

```
webBrowser1.ObjectForScripting = Me
```

5. Implement public properties or methods in your application code that your script code will use.

For example, if you use the form class for the scripting object, add the following code to your form class.

```
public void Test(String message)
{
    MessageBox.Show(message, "client code");
}
```

```
Public Sub Test(ByVal message As String)
    MessageBox.Show(message, "client code")
End Sub
```

6. Use the `window.external` object in your scripting code to access public properties and methods of the specified object.

The following HTML code demonstrates how to call a method on the scripting object from a button click. Copy this code into the BODY element of an HTML document that you load using the control's [Navigate](#) method or that you assign to the control's [DocumentText](#) property.

```
<button onclick="window.external.Test('called from script code')"
    call client code from script code
</button>
```

7. Implement functions in your script code that your application code will use.

The following HTML SCRIPT element provides an example function. Copy this code into the HEAD element of an HTML document that you load using the control's [Navigate](#) method or that you assign to the control's [DocumentText](#) property.

```
<script>
function test(message) {
    alert(message);
}
</script>
```

8. Use the [Document](#) property to access the script code from your client application code.

For example, add the following code to a button [Click](#) event handler.

```
webBrowser1.Document.InvokeScript("test",
    new String[] { "called from client code" });
```

```
webBrowser1.Document.InvokeScript("test", _  
    New String() {"called from client code"})
```

9. When you are finished debugging your DHTML, set the control's **ScriptErrorsSuppressed** property to `true` to prevent the [WebBrowser](#) control from displaying error messages for script code problems.

```
// Uncomment the following line when you are finished debugging.  
//webBrowser1.ScriptErrorsSuppressed = true;
```

```
' Uncomment the following line when you are finished debugging.  
'webBrowser1.ScriptErrorsSuppressed = True
```

Example

The following complete code example provides a demonstration application that you can use to understand this feature. The HTML code is loaded into the [WebBrowser](#) control through the **DocumentText** property instead of being loaded from a separate HTML file.

```
using System;
using System.Windows.Forms;
using System.Security.Permissions;

[PermissionSet(SecurityAction.Demand, Name="FullTrust")]
[System.Runtime.InteropServices.ComVisibleAttribute(true)]
public class Form1 : Form
{
    private WebBrowser webBrowser1 = new WebBrowser();
    private Button button1 = new Button();

    [STAThread]
    public static void Main()
    {
        Application.EnableVisualStyles();
        Application.Run(new Form1());
    }

    public Form1()
    {
        button1.Text = "call script code from client code";
        button1.Dock = DockStyle.Top;
        button1.Click += new EventHandler(button1_Click);
        webBrowser1.Dock = DockStyle.Fill;
        Controls.Add(webBrowser1);
        Controls.Add(button1);
        Load += new EventHandler(Form1_Load);
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        webBrowser1.AllowWebBrowserDrop = false;
        webBrowser1.IsWebBrowserContextMenuEnabled = false;
        webBrowser1.WebBrowserShortcutsEnabled = false;
        webBrowser1.ObjectForScripting = this;
        // Uncomment the following line when you are finished debugging.
        //webBrowser1.ScriptErrorsSuppressed = true;

        webBrowser1.DocumentText =
            "<html><head><script>" +
            "function test(message) { alert(message); }" +
            "</script></head><body><button " +
            "onclick=\"window.external.Test('called from script code')\\">" +
            "call client code from script code</button>" +
            "</body></html>";
    }

    public void Test(String message)
    {
        MessageBox.Show(message, "client code");
    }

    private void button1_Click(object sender, EventArgs e)
    {
        webBrowser1.Document.InvokeScript("test",
            new String[] { "called from client code" });
    }
}
```

```

Imports System
Imports System.Windows.Forms
Imports System.Security.Permissions

<PermissionSet(SecurityAction.Demand, Name:="FullTrust")> _
<System.Runtime.InteropServices.ComVisibleAttribute(True)> _
Public Class Form1
    Inherits Form

    Private webBrowser1 As New WebBrowser()
    Private WithEvents button1 As New Button()

    <STAThread()> _
    Public Shared Sub Main()
        Application.EnableVisualStyles()
        Application.Run(New Form1())
    End Sub

    Public Sub New()
        button1.Text = "call script code from client code"
        button1.Dock = DockStyle.Top
        webBrowser1.Dock = DockStyle.Fill
        Controls.Add(webBrowser1)
        Controls.Add(button1)
    End Sub

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs) _
        Handles Me.Load

        webBrowser1.AllowWebBrowserDrop = False
        webBrowser1.IsWebBrowserContextMenuEnabled = False
        webBrowser1.WebBrowserShortcutsEnabled = False
        webBrowser1.ObjectForScripting = Me
        ' Uncomment the following line when you are finished debugging.
        'webBrowser1.ScriptErrorsSuppressed = True

        webBrowser1.DocumentText = _
            "<html><head><script>" & _
            "function test(message) { alert(message); }" & _
            "</script></head><body><button " & _
            "onclick=""window.external.Test('called from script code')"" > " & _
            "call client code from script code</button>" & _
            "</body></html>"
    End Sub

    Public Sub Test(ByVal message As String)
        MessageBox.Show(message, "client code")
    End Sub

    Private Sub button1_Click(ByVal sender As Object, ByVal e As EventArgs) _
        Handles button1.Click

        webBrowser1.Document.InvokeScript("test", _
            New String() {"called from client code"})
    End Sub
End Class

```

Compiling the Code

This code requires:

- References to the System and System.Windows.Forms assemblies.

For information about building this example from the command line for Visual Basic or Visual C#, see [Building from the Command Line](#) or [Command-line Building With csc.exe](#). You can also build this example in Visual Studio by pasting the code into a new project. Also see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#).

See Also

[WebBrowser](#)

[WebBrowser.Document](#)

[WebBrowser.ObjectForScripting](#)

[WebBrowser Control](#)

Using the Managed HTML Document Object Model

5/4/2018 • 1 min to read • [Edit Online](#)

The managed HTML document object model (DOM) provides a wrapper based on the .NET Framework for the HTML classes exposed by Internet Explorer. Use these classes to manipulate HTML pages hosted in the [WebBrowser](#) control, or to build new pages from the beginning.

In This Section

[How to: Access the Managed HTML Document Object Model](#)

Describes how to obtain a valid instance of [HtmlDocument](#) from either a Windows Forms application or a [UserControl](#) hosted in Internet Explorer.

[How to: Access the HTML Source in the Managed HTML Document Object Model](#)

Describes how to obtain the original, unmodified HTML source, and how to obtain the "live" source that reflects the current state of the DOM.

[How to: Change Styles on an Element in the Managed HTML Document Object Model](#)

Describes how to manipulate styles, which are used to control the visual display of elements.

[Accessing Frames in the Managed HTML Document Object Model](#)

Describes what frames and framesets are, and how to access the DOM of a frame.

[Accessing Unexposed Members on the Managed HTML Document Object Model](#)

Describes how to access members of the underlying DOM that do not have a managed equivalent.

Reference

[HtmlDocument](#)

[HtmlElement](#)

[HtmlWindow](#)

Related Sections

[WebBrowser Control](#)

See Also

[About the DHTML Object Model](#)

How to: Access the Managed HTML Document Object Model

5/4/2018 • 1 min to read • [Edit Online](#)

You can access the managed HTML Document Object Model (DOM) from two types of applications:

- A Windows Forms application (.exe) that hosted the managed [WebBrowser](#) control. These two technologies complement one another, with the [WebBrowser](#) control displaying the page to the user and the HTML DOM representing the document's logical structure.
- A Windows Forms [UserControl](#) hosted within Internet Explorer. You can access the HTML DOM representing the page on which your [UserControl](#) is hosted in order to change the document's structure or open modal dialog boxes, among many other possibilities.

To access DOM from a Windows Forms application

1. Host a [WebBrowser](#) control within your Windows Forms application and monitor for the [DocumentCompleted](#) event. For details on hosting controls and monitoring for events, see [Events](#).
2. Retrieve the [HtmlDocument](#) for the current page by accessing the [Document](#) property of the [WebBrowser](#) control.

To access DOM from a UserControl hosted in Internet Explorer

1. Create your own custom derived class of the [UserControl](#) class. For more information, see [How to: Author Composite Controls](#).
2. Place the following code inside of your Load event handler for your [UserControl](#):

```
HtmlDocument doc = null;

private void UserControl1_Load(object sender, EventArgs e)
{
    if (this.Site != null)
    {
        doc = (HtmlDocument)this.Site.GetService(typeof(HtmlDocument));
    }
}
```

```
Private Sub UserControl1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    If (Me.Site IsNot Nothing) Then
        Dim Doc As HtmlDocument = CType(Me.Site.GetService(Type.GetType("System.Windows.Forms.HtmlDocument")),
        HtmlDocument)
        End If
    End Sub
```

Robust Programming

1. When using the DOM through the [WebBrowser](#) control, you should always wait until the [DocumentCompleted](#) event occurs before attempting to access the [Document](#) property of the [WebBrowser](#) control. The [DocumentCompleted](#) event is raised after the entire document has loaded; if you use the DOM before then, you risk causing a run-time exception in your application.

.NET Framework Security

1. Your application or [UserControl](#) will require full trust in order to access the managed HTML DOM. If you are deploying a Windows Forms application using ClickOnce, you can request full trust using either Permission Elevation or Trusted Application Deployment; see [Securing ClickOnce Applications](#) for details.

See Also

[Using the Managed HTML Document Object Model](#)

How to: Access the HTML Source in the Managed HTML Document Object Model

5/4/2018 • 1 min to read • [Edit Online](#)

The `DocumentStream` and `DocumentText` properties on the `WebBrowser` control return the HTML of the current document as it existed when it was first displayed. However, if you modify the page using method and property calls such as `AppendChild` and `InnerText`, these changes will not appear when you call `DocumentStream` and `DocumentText`. To obtain the most up-to-date HTML source for the DOM, you must call the `OuterHtml` property on the HTML element.

The following procedure shows how to retrieve the dynamic source and display it in a separate shortcut menu.

Retrieving the dynamic source with the `OuterHtml` property

1. Create a new Windows Forms application. Start with a single `Form`, and call it `Form1`.
2. Host the `WebBrowser` control in your Windows Forms application, and name it `WebBrowser1`. For more information, see [How to: Add Web Browser Capabilities to a Windows Forms Application](#).
3. Create a second `Form` in your application called `CodeForm`.
4. Add a `RichTextBox` control to `CodeForm` and set its `Dock` property to `Fill`.
5. Create a public property on `CodeForm` called `Code`.

```
public string Code
{
    get
    {
        if (richTextBox1.Text != null)
        {
            return (richTextBox1.Text);
        }
        else
        {
            return ("");
        }
    }
    set
    {
        richTextBox1.Text = value;
    }
}
```

```

Public Property Code() As String
    Get
        If (RichTextBox1.Text IsNot Nothing) Then
            Code = RichTextBox1.Text
        Else
            Code = ""
        End If
    End Get

    Set(ByVal value As String)
        RichTextBox1.Text = value
    End Set
End Property

```

6. Add a **Button** control named `Button1` to your **Form**, and monitor for the **Click** event. For details on monitoring events, see [Events](#).
7. Add the following code to the **Click** event handler.

```

private void button1_Click(object sender, EventArgs e)
{
    HtmlElement elem;

    if (webBrowser1.Document != null)
    {
        CodeForm cf = new CodeForm();
        HtmlElementCollection elems = webBrowser1.Document.GetElementsByTagName("HTML");
        if (elems.Count == 1)
        {
            elem = elems[0];
            cf.Code = elem.OuterHtml;
            cf.Show();
        }
    }
}

```

```

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
    Button1.Click
    Dim elem As HtmlElement

    If (WebBrowser1.Document IsNot Nothing) Then
        Dim cf As New CodeForm()
        Dim elems As HtmlElementCollection = WebBrowser1.Document.GetElementsByTagName("HTML")
        If (elems.Count = 1) Then
            elem = elems(0)
            cf.Code = elem.OuterHtml
            cf.Show()
        End If
    End If
End Sub

```

Robust Programming

Always test the value of **Document** before attempting to retrieve it. If the current page is not finished loading, **Document** or one or more of its child objects may not be initialized.

See Also

[Using the Managed HTML Document Object Model](#)

[WebBrowser Control Overview](#)

How to: Change Styles on an Element in the Managed HTML Document Object Model

5/4/2018 • 5 min to read • [Edit Online](#)

You can use styles in HTML to control the appearance of a document and its elements. [HtmlDocument](#) and [HtmlElement](#) support [Style](#) properties that take style strings of the following format:

```
name1:value1;...;nameN:valueN;
```

Here is a `DIV` with a style string that sets the font to Arial and all text to bold:

```
<DIV style="font-face:arial;font-weight:bold;">  
Hello, world!  
</DIV>
```

The problem with manipulating styles using the [Style](#) property is that it can prove cumbersome to add to and remove individual style settings from the string. For example, it would become a complex procedure for you to render the previous text in italics whenever the user positions the cursor over the `DIV`, and take italics off when the cursor leaves the `DIV`. Time would become an issue if you need to manipulate a large number of styles in this manner.

The following procedure contains code that you can use to easily manipulate styles on HTML documents and elements. The procedure requires that you know how to perform basic tasks in Windows Forms, such as creating a new project and adding a control to a form.

To process style changes in a Windows Forms application

1. Create a new Windows Forms project.
2. Create a new class file ending in the extension appropriate for your programming language.
3. Copy the `StyleGenerator` class code in the Example section of this topic into the class file, and save the code.
4. Save the following HTML to a file named Test.htm.

```
<HTML>  
  <BODY>  
  
    <DIV style="font-face:arial;font-weight:bold;">  
      Hello, world!  
    </DIV><P>  
  
    <DIV>  
      Hello again, world!  
    </DIV><P>  
  
  </BODY>  
</HTML>
```

5. Add a [WebBrowser](#) control named `webBrowser1` to the main form of your project.
6. Add the following code to your project's code file.

IMPORTANT

Ensure that the `webBrowser1_DocumentCompleted` event hander is configured as a listener for the `DocumentCompleted` event. In Visual Studio, double-click on the `WebBrowser` control; in a text editor, configure the listener programmatically.

```
StyleGenerator sg = null;
HtmlElement elem = null;

private void webBrowser1_DocumentCompleted(object sender, WebBrowserDocumentCompletedEventArgs e)
{
    sg = new StyleGenerator();

    webBrowser1.Document.MouseOver += new HtmlElementEventHandler(Document_MouseOver);
    webBrowser1.Document.MouseLeave += new HtmlElementEventHandler(Document_MouseLeave);
}

void Document_MouseOver(object sender, HtmlElementEventArgs e)
{
    elem = webBrowser1.Document.GetElementFromPoint(e.mousePosition);
    if (elem.tagName.Equals("DIV"))
    {
        sg.ParseStyleString(elem.style);
        sg.setStyle("font-style", "italic");
        elem.style = sg.getStyleString();
    }
}

void Document_MouseLeave(object sender, HtmlElementEventArgs e)
{
    if (elem != null)
    {
        sg.RemoveStyle("font-style");
        elem.style = sg.getStyleString();
        // Reset, since we may mouse over a new DIV element next time.
        sg.Clear();
    }
}
```

```

<System.Security.Permissions.PermissionSet(Security.Permissions.SecurityAction.Demand,
Name:="FullTrust")> _
Public Class Form1

    Dim SG As StyleGenerator = Nothing
    Dim Elem As HtmlElement = Nothing
    Dim WithEvents DocumentEvents As HtmlDocument

    Private Sub WebBrowser1_DocumentCompleted(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.WebBrowserDocumentCompletedEventArgs) Handles WebBrowser1.DocumentCompleted
        SG = New StyleGenerator()
        DocumentEvents = WebBrowser1.Document
    End Sub

    Private Sub Document_MouseOver(ByVal sender As Object, ByVal e As HtmlElementEventArgs) Handles
DocumentEvents.MouseOver
        Elem = WebBrowser1.Document.GetElementFromPoint(e.mousePosition)
        If Elem.TagName.Equals("DIV") Then
            SG.ParseStyleString(Elem.Style)
            SG.setStyle("font-style", "italic")
            Elem.Style = SG.getStyleString()
        End If
    End Sub

    Private Sub Document_MouseLeave(ByVal sender As Object, ByVal e As HtmlElementEventArgs) Handles
DocumentEvents.MouseLeave
        If (Elem IsNot Nothing) Then
            SG.RemoveStyle("font-style")
            Elem.Style = SG.getStyleString()
            ' Reset, since we may mouse over a new DIV element next time.
            SG.Clear()
        End If
    End Sub
End Class

```

- Run the project. Run your cursor over the first `DIV` to observe the effects of the code.

Example

The following code example shows the full code for the `StyleGenerator` class, which parses an existing style value, supports adding, changing, and removing styles, and returns a new style value with the requested changes.

```

using System;
using System.Collections.Generic;
using System.Text;

namespace ManagedDOMStyles
{
    public class StyleGenerator
    {
        private Dictionary<string, string> styleDB;

        public StyleGenerator()
        {
            styleDB = new Dictionary<string, string>();
        }

        public bool ContainsStyle(string name)
        {
            return(styleDB.ContainsKey(name));
        }

        public string SetStyle(string name, string value)
        {
            string oldValue = "";

```

```

        string oldValue = null;

        if (!(name.Length > 0))
        {
            throw (new ArgumentException("Parameter name cannot be zero-length."));
        }
        if (!(value.Length > 0))
        {
            throw (new ArgumentException("Parameter value cannot be zero-length."));
        }

        if (styleDB.ContainsKey(name))
        {
            oldValue = styleDB[name];
        }

        styleDB[name] = value;

        return (oldValue);
    }

    public string GetStyle(string name)
    {
        if (!(name.Length > 0))
        {
            throw (new ArgumentException("Parameter name cannot be zero-length."));
        }

        if (styleDB.ContainsKey(name))
        {
            return (styleDB[name]);
        }
        else
        {
            return ("");
        }
    }

    public void RemoveStyle(string name)
    {
        if (styleDB.ContainsKey(name))
        {
            styleDB.Remove(name);
        }
    }

    public string GetStyleString()
    {
        if (styleDB.Count > 0)
        {
            StringBuilder styleString = new StringBuilder("");
            foreach (string key in styleDB.Keys)
            {
                styleString.Append(String.Format("{0}:{1};", (object)key, (object)styleDB[key]));
            }

            return (styleString.ToString());
        }
        else
        {
            return ("");
        }
    }

    public void ParseStyleString(string styles)
    {
        if (styles.Length > 0)
        {
            string[] stylePairs = styles.Split(new char[] { ';' });
            foreach (string pair in stylePairs)
            {
                string[] keyValue = pair.Split(new char[] { '=' });
                if (keyValue.Length == 2)
                {
                    string key = keyValue[0].Trim();
                    string value = keyValue[1].Trim();
                    if (styleDB.ContainsKey(key))
                    {
                        styleDB[key] = value;
                    }
                    else
                    {
                        styleDB.Add(key, value);
                    }
                }
            }
        }
    }
}

```

```

        foreach(string stylePair in stylePairs)
    {
        if (stylePairs.Length > 0)
        {
            string[] styleNameValue = stylePair.Split(new char[] { ':' });
            if (styleNameValue.Length == 2)
            {
                styleDB[styleNameValue[0]] = styleNameValue[1];
            }
        }
    }

    public void Clear()
    {
        styleDB.Clear();
    }
}
}

```

```

Imports System
Imports System.Collections.Generic
Imports System.Text

Public Class StyleGenerator
    Dim styleDB As Dictionary(Of String, String)

    Public Sub New()
        styleDB = New Dictionary(Of String, String)()
    End Sub

    Public Function ContainsStyle(ByVal name As String) As Boolean
        Return styleDB.ContainsKey(name)
    End Function

    Public Function SetStyle(ByVal name As String, ByVal value As String) As String
        Dim oldValue As String = ""

        If (Not name.Length > 0) Then
            Throw New ArgumentException("Parameter name cannot be zero-length.")
        End If
        If (Not value.Length > 0) Then
            Throw New ArgumentException("Parameter value cannot be zero-length.")
        End If

        If (styleDB.ContainsKey(name)) Then
            oldValue = styleDB(name)
        End If

        styleDB(name) = value

        Return oldValue
    End Function

    Public Function GetStyle(ByVal name As String) As String
        If (Not name.Length > 0) Then
            Throw New ArgumentException("Parameter name cannot be zero-length.")
        End If

        If (styleDB.ContainsKey(name)) Then
            Return styleDB(name)
        Else
            Return ""
        End If
    End Function
}

```

```

End Function

Public Sub RemoveStyle(ByVal name As String)
    If (styleDB.ContainsKey(name)) Then
        styleDB.Remove(name)
    End If
End Sub

Public Function GetStyleString() As String
    If (styleDB.Count > 0) Then
        Dim styleString As New StringBuilder("")
        Dim key As String
        For Each key In styleDB.Keys
            styleString.Append(String.Format("{0}:{1};", CType(key, Object), CType(styleDB(key), Object)))
        Next key

        Return styleString.ToString()
    Else
        Return ""
    End If
End Function

Public Sub ParseStyleString(ByVal styles As String)
    If (styles.Length) > 0 Then
        Dim stylePairs As String() = styles.Split(New Char() {";"})
        Dim stylePair As String
        For Each stylePair In stylePairs
            If (stylePairs.Length > 0) Then
                Dim styleNameValue As String() = stylePair.Split(New Char() {":"})
                If (styleNameValue.Length = 2) Then
                    styleDB(styleNameValue(0)) = styleNameValue(1)
                End If
            End If
        Next stylePair
    End If
End Sub

Public Sub Clear()
    styleDB.Clear()
End Sub
End Class

```

See Also

[HtmlElement](#)

Accessing Frames in the Managed HTML Document Object Model

5/4/2018 • 2 min to read • [Edit Online](#)

Some HTML documents are composed out of *frames*, or windows that can hold their own distinct HTML documents. Using frames makes it easy to create HTML pages in which one or more pieces of the page remain static, such as a navigation bar, while other frames constantly change their content.

HTML authors can create frames in one of two ways:

- Using the `FRAMESET` and `FRAME` tags, which create fixed windows.

-or-

- Using the `IFRAME` tag, which creates a floating window that can be repositioned at run time.

1. Because frames contain HTML documents, they are represented in the Document Object Model (DOM) as both window elements and frame elements.
2. When you access a `FRAME` or `IFRAME` tag by using the Frames collection of `HtmlWindow`, you are retrieving the window element corresponding to the frame. This represents all of the frame's dynamic properties, such as its current URL, document, and size.
3. When you access a `FRAME` or `IFRAME` tag by using the `WindowFrameElement` property of `HtmlWindow`, the `Children` collection, or methods such as `GetElementsByName` or `GetElementById`, you are retrieving the frame element. This represents the static properties of the frame, including the URL specified in the original HTML file.

Frames and Security

Access to frames is complicated by the fact that the managed HTML DOM implements a security measure known as *cross-frame scripting security*. If a document contains a `FRAMESET` with two or more `FRAME`s in different domains, these `FRAME`s cannot interact with one another. In other words, a `FRAME` that displays content from your Web site cannot access information in a `FRAME` that hosts a third-party site such as <http://www.adatum.com/>. This security is implemented at the level of the `HtmlWindow` class. You can obtain general information about a `FRAME` hosting another Web site, such as its URL, but you will be unable to access its `Document` or change the size or location of its hosting `FRAME` or `IFRAME`.

This rule also applies to windows that you open using the `Open` and `OpenNew` methods. If the window you open is in a different domain from the page hosted in the `WebBrowser` control, you will not be able to move that window or examine its contents. These restrictions are also enforced if you use the `WebBrowser` control to display a Web site that is different from the Web site used to deploy your Windows Forms-based application. If you use ClickOnce deployment technology to install your application from Web site A, and you use the `WebBrowser` to display Web site B, you will not be able to access Web site B's data.

For more information about cross-site scripting, see [About Cross-Frame Scripting and Security](#).

See Also

[FRAME Element | frame Object](#)

[Using the Managed HTML Document Object Model](#)

Accessing Unexposed Members on the Managed HTML Document Object Model

5/4/2018 • 2 min to read • [Edit Online](#)

The managed HTML Document Object Model (DOM) contains a class called [HtmlElement](#) that exposes the properties, methods, and events that all HTML elements have in common. Sometimes, however, you will need to access members that the managed interface does not directly expose. This topic examines two ways for accessing unexposed members, including JScript and VBScript functions defined inside of a Web page.

Accessing Unexposed Members through Managed Interfaces

[HtmlDocument](#) and [HtmlElement](#) provide four methods that enable access to unexposed members. The following table shows the types and their corresponding methods.

MEMBER TYPE	METHOD(S)
Properties (HtmlElement)	GetAttribute SetAttribute
Methods	InvokeMember
Events (HtmlDocument)	AttachEventHandler DetachEventHandler
Events (HtmlElement)	AttachEventHandler DetachEventHandler
Events (HtmlWindow)	AttachEventHandler DetachEventHandler

When you use these methods, it is assumed that you have an element of the correct underlying type. Suppose that you want to listen to the `Submit` event of a `FORM` element on an HTML page, so that you can perform some pre-processing on the `FORM`'s values before the user submits them to the server. Ideally, if you have control over the HTML, you would define the `FORM` to have a unique `ID` attribute.

```
<HTML>

    <HEAD>
        <TITLE>Form Page</TITLE>
    </HEAD>

    <BODY>
        <FORM ID="form1">
            ... form fields defined here ...
        </FORM>
    </BODY>

</HTML>
```

After you load this page into the [WebBrowser](#) control, you can use the [GetElementById](#) method to retrieve the **FORM** at run time using `form1` as the argument.

```
private void SubmitForm(String formName)
{
    HtmlElementCollection elems = null;
    HtmlElement elem = null;

    if (webBrowser1.Document != null)
    {
        HtmlDocument doc = webBrowser1.Document;
        elems = doc.All.GetElementsByName(formName);
        if (elems != null && elems.Count > 0)
        {
            elem = elems[0];
            if (elem.TagName.Equals("FORM"))
            {
                elem.InvokeMember("Submit");
            }
        }
    }
}
```

```
Private Sub SubmitForm(ByVal FormName As String)
    Dim Elems As HtmlElementCollection
    Dim Elem As HtmlElement

    If (WebBrowser1.Document IsNot Nothing) Then
        With WebBrowser1.Document
            Elems = .All.GetElementsByName(FormName)
            If (Not Elems Is Nothing And Elems.Count > 0) Then
                Elem = Elems(0)
                If (Elem.TagName.Equals("FORM")) Then
                    Elem.InvokeMember("Submit")
                End If
            End If
        End With
    End If
End Sub
```

Accessing Unmanaged Interfaces

You can also access unexposed members on the managed HTML DOM by using the unmanaged Component Object Model (COM) interfaces exposed by each DOM class. This is recommended if you have to make several calls against unexposed members, or if the unexposed members return other unmanaged interfaces not wrapped by the managed HTML DOM.

The following table shows all of the unmanaged interfaces exposed through the managed HTML DOM. Click on each link for an explanation of its usage and for example code.

TYPE	UNMANAGED INTERFACE
HtmlDocument	DomDocument
HtmlElement	DomElement
HtmlWindow	DomWindow

TYPE	UNMANAGED INTERFACE
HtmlHistory	DomHistory

The easiest way to use the COM interfaces is to add a reference to the unmanaged HTML DOM library (MSHTML.dll) from your application, although this is unsupported. For more information, see [Knowledge Base Article 934368](#).

Accessing Script Functions

An HTML page can define one or more functions by using a scripting language such as JScript or VBScript. These functions are placed inside of a `SCRIPT` page in the page, and can be run on demand or in response to an event on the DOM.

You can call any script functions you define in an HTML page using the [InvokeScript](#) method. If the script method returns an HTML element, you can use a cast to convert this return result to an [HtmlElement](#). For details and example code, see [InvokeScript](#).

See Also

[Using the Managed HTML Document Object Model](#)

Windows Forms Controls Used to List Options

5/4/2018 • 1 min to read • [Edit Online](#)

You can add a variety of controls to a Windows Form if you want to provide users with a list of options to choose from. Depending on how much you want to restrict your users' input, you can add a [ListBox](#) control, a [ComboBox](#) control, or a [CheckedListBox](#) control. Use the following links to determine which control best suits your needs.

In This Section

[When to Use a Windows Forms ComboBox Instead of a ListBox](#)

Recommends an appropriate list-based control depending on the needs and restrictions of your Windows Form.

[How to: Access Specific Items in a Windows Forms ComboBox, ListBox, or CheckedListBox Control](#)

Gives instructions for programmatically determining which item in a list appears in a given position.

[How to: Add and Remove Items from a Windows Forms ComboBox, ListBox, or CheckedListBox Control](#)

Gives instructions for adding or removing items from a control's list of items.

[How to: Create a Lookup Table for a Windows Forms ComboBox, ListBox, or CheckedListBox Control](#)

Gives directions for displaying and storing form data in useful formats.

[How to: Bind a Windows Forms ComboBox or ListBox Control to Data](#)

Gives directions for binding a list-based control to a data source.

[How to: Sort the Contents of a Windows Forms ComboBox, ListBox, or CheckedListBox Control](#)

Explains how to sort list data at its data source.

Reference

[CheckedListBox](#)

Describes this class and has links to all its members.

[ComboBox](#)

Describes this class and has links to all its members.

[ListBox](#)

Describes this class and has links to all its members.

Related Sections

[CheckedListBox Control Overview](#)

Explains what this control is and its key features and properties.

[ComboBox Control Overview](#)

Explains what this control is and its key features and properties.

[ListBox Control Overview](#)

Explains what this control is and its key features and properties.

[Controls to Use on Windows Forms](#)

Provides a complete list of Windows Forms controls, with links to information on their use.

When to Use a Windows Forms ComboBox Instead of a ListBox

5/4/2018 • 1 min to read • [Edit Online](#)

The [ComboBox](#) and the [ListBox](#) controls have similar behaviors, and in some cases may be interchangeable. There are times, however, when one or the other is more appropriate to a task.

Generally, a combo box is appropriate when there is a list of suggested choices, and a list box is appropriate when you want to limit input to what is on the list. A combo box contains a text box field, so choices not on the list can be typed in. The exception is when the [DropDownStyle](#) property is set to [DropDownList](#). In that case, the control will select an item if you type its first letter.

In addition, combo boxes save space on a form. Because the full list is not displayed until the user clicks the down arrow, a combo box can easily fit in a small space where a list box would not fit. An exception is when the [DropDownStyle](#) property is set to [Simple](#): the full list is displayed, and the combo box takes up more room than a list box would.

See Also

[ComboBox](#)

[ListBox](#)

[How to: Add and Remove Items from a Windows Forms ComboBox, ListBox, or CheckedListBox Control](#)

[How to: Sort the Contents of a Windows Forms ComboBox, ListBox, or CheckedListBox Control](#)

[Windows Forms Controls Used to List Options](#)

How to: Access Specific Items in a Windows Forms ComboBox, ListBox, or CheckedListBox Control

5/4/2018 • 1 min to read • [Edit Online](#)

Accessing specific items in a Windows Forms combo box, list box, or checked list box is an essential task. It enables you to programmatically determine what is in a list, at any given position.

To access a specific item

1. Query the `Items` collection using the index of the specific item:

```
Private Function GetItemText(i As Integer) As String
    ' Return the text of the item using the index:
    Return ComboBox1.Items(i).ToString
End Function
```

```
private string GetItemText(int i)
{
    // Return the text of the item using the index:
    return (comboBox1.Items[i].ToString());
}
```

```
private:
String^ GetItemText(int i)
{
    // Return the text of the item using the index:
    return (comboBox1->Items->Item[i]->ToString());
}
```

See Also

[ComboBox](#)

[ListBox](#)

[CheckedListBox](#)

[Windows Forms Controls Used to List Options](#)

How to: Add and Remove Items from a Windows Forms ComboBox, ListBox, or CheckedListBox Control

5/4/2018 • 1 min to read • [Edit Online](#)

Items can be added to a Windows Forms combo box, list box, or checked list box in a variety of ways, because these controls can be bound to a variety of data sources. However, this topic demonstrates the simplest method and requires no data binding. The items displayed are usually strings; however, any object can be used. The text that is displayed in the control is the value returned by the object's `ToString` method.

To add items

1. Add the string or object to the list by using the `Add` method of the `ObjectCollection` class. The collection is referenced using the `Items` property:

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

```
comboBox1.Items.Add("Tokyo");
```

```
comboBox1->Items->Add("Tokyo");
```

● or -

2. Insert the string or object at the desired point in the list with the `Insert` method:

```
CheckedListBox1.Items.Insert(0, "Copenhagen")
```

```
checkedListBox1.Items.Insert(0, "Copenhagen");
```

```
checkedListBox1->Items->Insert(0, "Copenhagen");
```

● or -

3. Assign an entire array to the `Items` collection:

```
Dim ItemObject(9) As System.Object  
Dim i As Integer  
For i = 0 To 9  
    ItemObject(i) = "Item" & i  
Next i  
ListBox1.Items.AddRange(ItemObject)
```

```
System.Object[] ItemObject = new System.Object[10];
for (int i = 0; i <= 9; i++)
{
    ItemObject[i] = "Item" + i;
}
listBox1.Items.AddRange(ItemObject);
```

```
Array<System::Object^>^ ItemObject = gcnew Array<System::Object^>(10);
for (int i = 0; i <= 9; i++)
{
    ItemObject[i] = String::Concat("Item", i.ToString());
}
listBox1->Items->AddRange(ItemObject);
```

To remove an item

1. Call the `Remove` or `RemoveAt` method to delete items.

`Remove` has one argument that specifies the item to remove. `RemoveAt` removes the item with the specified index number.

```
' To remove item with index 0:
ComboBox1.Items.RemoveAt(0)
' To remove currently selected item:
ComboBox1.Items.Remove(ComboBox1.SelectedItem)
' To remove "Tokyo" item:
ComboBox1.Items.Remove("Tokyo")
```

```
// To remove item with index 0:
comboBox1.Items.RemoveAt(0);
// To remove currently selected item:
comboBox1.Items.Remove(comboBox1.SelectedItem);
// To remove "Tokyo" item:
comboBox1.Items.Remove("Tokyo");
```

```
// To remove item with index 0:
comboBox1->Items->RemoveAt(0);
// To remove currently selected item:
comboBox1->Items->Remove(comboBox1->SelectedItem);
// To remove "Tokyo" item:
comboBox1->Items->Remove("Tokyo");
```

To remove all items

1. Call the `Clear` method to remove all items from the collection:

```
ListBox1.Items.Clear()
```

```
listBox1.Items.Clear();
```

```
listBox1->Items->Clear();
```

See Also

[ComboBox](#)

[ListBox](#)

[CheckedListBox](#)

[How to: Sort the Contents of a Windows Forms ComboBox, ListBox, or CheckedListBox Control](#)

[When to Use a Windows Forms ComboBox Instead of a ListBox](#)

[Windows Forms Controls Used to List Options](#)

How to: Create a Lookup Table for a Windows Forms ComboBox, ListBox, or CheckedListBox Control

5/4/2018 • 2 min to read • [Edit Online](#)

Sometimes it is useful to display data in a user-friendly format on a Windows Form, but store the data in a format that is more meaningful to your program. For example, an order form for food might display the menu items by name in a list box. However, the data table recording the order would contain the unique ID numbers representing the food. The following tables show an example of how to store and display order-form data for food.

OrderDetailsTable

ORDERID	ITEMID	QUANTITY
4085	12	1
4086	13	3

ItemTable

ID	NAME
12	Potato
13	Chicken

In this scenario, one table, **OrderDetailsTable**, stores the actual information you are concerned with displaying and saving. But to save space, it does so in a fairly cryptic fashion. The other table, **ItemTable**, contains only appearance-related information about which ID number is equivalent to which food name, and nothing about the actual food orders.

The **ItemTable** is connected to the [ComboBox](#), [ListBox](#), or [CheckedListBox](#) control through three properties. The **DataSource** property contains the name of this table. The **DisplayMember** property contains the data column of that table that you want to display in the control (the food name). The **ValueMember** property contains the data column of that table with the stored information (the ID number).

The **OrderDetailsTable** is connected to the control by its bindings collection, accessed through the **DataBindings** property. When you add a binding object to the collection, you connect a control property to a specific data member (the column of ID numbers) in a data source (the **OrderDetailsTable**). When a selection is made in the control, this table is where the form input is saved.

To create a lookup table

1. Add a [ComboBox](#), [ListBox](#), or [CheckedListBox](#) control to the form.
2. Connect to your data source.
3. Establish a data relation between the two tables. See [Introduction to DataRelation Objects](#).
4. Set the following properties. They can be set in code or in the designer.

PROPERTY	SETTING
DataSource	The table that contains information about which ID number is equivalent to which item. In the previous scenario, this is <code>ItemTable</code> .
DisplayMember	The column of the data source table that you want to display in the control. In the previous scenario, this is <code>"Name"</code> (to set in code, use quotation marks).
ValueMember	The column of the data source table that contains the stored information. In the previous scenario, this is <code>"ID"</code> (to set in code, use quotation marks).

5. In a procedure, call the `Add` method of the `ControlBindingsCollection` class to bind the control's `SelectedValue` property to the table recording the form input. You can also do this in the Designer instead of in code, by accessing the control's `DataBindings` property in the **Properties** window. In the previous scenario, this is `OrderDetailsTable`, and the column is `"ItemID"`.

```
ListBox1.DataBindings.Add("SelectedValue", OrderDetailsTable, "ItemID")
```

```
listBox1.DataBindings.Add("SelectedValue", OrderDetailsTable, "ItemID");
```

See Also

[Data Binding and Windows Forms](#)

[ListBox Control Overview](#)

[ComboBox Control Overview](#)

[CheckedListBox Control Overview](#)

[Windows Forms Controls Used to List Options](#)

How to: Bind a Windows Forms ComboBox or ListBox Control to Data

5/4/2018 • 1 min to read • [Edit Online](#)

You can bind the [ComboBox](#) and [ListBox](#) to data to perform tasks such as browsing data in a database, entering new data, or editing existing data.

To bind a ComboBox or ListBox control

1. Set the `DataSource` property to a data source object. Possible data sources include a [BindingSource](#) bound to data, a data table, a data view, a dataset, a data view manager, an array, or any class that implements the [IList](#) interface. For more information, see [Data Sources Supported by Windows Forms](#).
2. If you are binding to a table, set the `DisplayMember` property to the name of a column in the data source.
- or -

If you are binding to an [IList](#), set the display member to a public property of the type in the list.

```
Private Sub BindComboBox()  
    ComboBox1.DataSource = DataSet1.Tables("Suppliers")  
    ComboBox1.DisplayMember = "ProductName"  
End Sub
```

```
private void BindComboBox()  
{  
    comboBox1.DataSource = dataSet1.Tables["Suppliers"];  
    comboBox1.DisplayMember = "ProductName";  
}
```

NOTE

If you are bound to a data source that does not implement the [IBindingList](#) interface, such as an [ArrayList](#), the bound control's data will not be updated when the data source is updated. For example, if you have a combo box bound to an [ArrayList](#) and data is added to the [ArrayList](#), these new items will not appear in the combo box. However, you can force the combo box to be updated by calling the [SuspendBinding](#) and [ResumeBinding](#) methods on the instance of the [BindingContext](#) class to which the control is bound.

See Also

[ComboBox](#)

[ListBox](#)

[Windows Forms Data Binding](#)

[Data Binding and Windows Forms](#)

[Windows Forms Controls Used to List Options](#)

How to: Sort the Contents of a Windows Forms ComboBox, ListBox, or CheckedListBox Control

5/4/2018 • 1 min to read • [Edit Online](#)

Windows Forms controls do not sort when they are data-bound. To display sorted data, use a data source that supports sorting and then have the data source sort it. Data sources that support sorting are data views, data view managers, and sorted arrays.

If the control is not data-bound, you can sort it.

To sort the list

1. Set the `Sorted` property to `true`.

This setting repositions all existing list items in sorted order.

See Also

[ComboBox](#)

[ListBox](#)

[CheckedListBox](#)

[Windows Forms Data Binding](#)

[How to: Add and Remove Items from a Windows Forms ComboBox, ListBox, or CheckedListBox Control](#)

[When to Use a Windows Forms ComboBox Instead of a ListBox](#)

[Windows Forms Controls Used to List Options](#)

Developing Custom Windows Forms Controls with the .NET Framework

5/4/2018 • 1 min to read • [Edit Online](#)

Windows Forms controls are reusable components that encapsulate user interface functionality and are used in client-side Windows-based applications. Not only does Windows Forms provide many ready-to-use controls, it also provides the infrastructure for developing your own controls. You can combine existing controls, extend existing controls, or author your own custom controls. This section provides background information and samples to help you develop Windows Forms controls.

In This Section

[Overview of Using Controls in Windows Forms](#)

Highlights the essential elements of using controls in Windows Forms applications.

[Varieties of Custom Controls](#)

Describes the different kinds of custom controls you can author with the `System.Windows.Forms` namespace.

[Windows Forms Control Development Basics](#)

Discusses the first steps in developing a Windows Forms control.

[Properties in Windows Forms Controls](#)

Shows how to add properties to Windows Forms controls.

[Events in Windows Forms Controls](#)

Shows how to handle and define events in Windows Forms controls.

[Attributes in Windows Forms Controls](#)

Describes the attributes you can apply to properties or other members of your custom controls and components.

[Custom Control Painting and Rendering](#)

Shows how to customize the appearance of your controls.

[Layout in Windows Forms Controls](#)

Shows how to create sophisticated layouts for your controls and forms.

[Multithreading in Windows Forms Controls](#)

Shows how to implement multithreaded controls.

Reference

[System.Windows.Forms.Control](#)

Describes this class and has links to all of its members.

[System.Windows.Forms.UserControl](#)

Describes this class and has links to all of its members.

Related Sections

[Design-Time Attributes for Components](#)

Lists metadata attributes to apply to components and controls so that they are displayed correctly at design time in visual designers.

[Extending Design-Time Support](#)

Describes how to implement classes such as editors and designers that provide design-time support.

[How to: License Components and Controls](#)

Describes how to implement licensing in your control or component.

Also see [Developing Windows Forms Controls at Design Time](#).

Overview of Using Controls in Windows Forms

5/4/2018 • 3 min to read • [Edit Online](#)

This topic describes the essential elements of a Windows Forms application and provides a simple example that uses controls and handles events in a Windows Forms application.

Simple Windows Forms Applications

At a minimum, a Windows Forms application consists of the following elements:

- One or more classes that derive from [System.Windows.Forms.Form](#).
- A `Main` method that invokes the `static` (`shared` in Visual Basic) `Run` method and passes a `Form` instance to it. The `Run` method processes messages from the operating system to the application.

The following code example shows the essential elements of a Windows Forms application.

```
Option Explicit
Option Strict

Imports System
Imports System.Windows.Forms

Public Class MyForm
    Inherits Form

    Public Sub New()
        Me.Text = "Hello World"
    End Sub 'New

    <STAThread()> _
    Public Shared Sub Main()
        Dim aform As New MyForm()
        ' The Application.Run method processes messages from the operating system
        ' to your application. If you comment out the next line of code,
        ' your application will compile and execute, but because it is not in the
        ' message loop, it will exit after an instance of the form is created.
        Application.Run(aform)
    End Sub
End Class
```

```

using System;
using System.Windows.Forms;

public class MyForm : Form {

    public MyForm() {
        this.Text = "Hello World";
    }
    [STAThread]
    public static void Main(string[] args) {
        MyForm aform = new MyForm();
        // The Application.Run method processes messages from the operating system
        // to your application. If you comment out the next line of code,
        // your application will compile and execute, but because it is not in the // message loop, it will exit after
        // an instance of the form is created.
        Application.Run(aform);
    }
}

```

Using Controls in a Windows Forms Application

The following code example shows a simple application that illustrates how Windows Forms applications use controls and handle events. The example consists of three buttons on a form; each button changes the background color when clicked.

```

Option Explicit
Option Strict

Imports System
Imports System.ComponentModel
Imports System.Windows.Forms
Imports System.Resources
Imports System.Drawing

Public Class MyForm
    Inherits Form
    Private red As Button
    Private blue As Button
    Private green As Button

    Public Sub New()
        InitializeComponent()
    End Sub

    Protected Overrides Sub Dispose(disposing As Boolean)
        MyBase.Dispose(disposing)
    End Sub

    ' InitializeComponent is a helper method for the constructor.
    ' It is included for consistency with code that is
    ' auto-generated by the Windows Forms designer in Visual Studio.
    Private Sub InitializeComponent()

        ' Creates three buttons, sets their properties, and attaches
        ' an event handler to each button.
        red = New Button()
        red.Text = "Red"
        red.Location = New Point(100, 50)
        red.Size = New Size(50, 50)
        AddHandler red.Click, AddressOf button_Click
        Controls.Add(red)

        blue = New Button()
        blue.Text = "Blue"

```

```

        blue.Location = New Point(100, 100)
        blue.Size = New Size(50, 50)
        AddHandler blue.Click, AddressOf button_Click
        Controls.Add(blue)

        green = New Button()
        green.Text = "Green"
        green.Location = New Point(100, 150)
        green.Size = New Size(50, 50)
        AddHandler green.Click, AddressOf button_Click
        Controls.Add(green)
    End Sub

    ' Event handler.
    Private Sub button_Click(sender As Object, e As EventArgs)
        If sender Is red Then
            Me.BackColor = Color.Red
        Else
            If sender Is blue Then
                Me.BackColor = Color.Blue
            Else
                Me.BackColor = Color.Green
            End If
        End If
    End Sub

    ' The STAThreadAttribute informs the common language runtime that
    ' Windows Forms uses the single-threaded apartment model.
<STAThread()> _
Public Shared Sub Main()
    Application.Run(New MyForm())
End Sub
End Class

```

```

using System;
using System.ComponentModel;
using System.Windows.Forms;
using System.Resources;
using System.Drawing;

public class MyForm : Form {
    private Button red;
    private Button blue;
    private Button green;

    public MyForm() : base() {
        InitializeComponent();
    }

    protected override void Dispose(bool disposing) {
        base.Dispose(disposing);
    }

    // InitializeComponent is a helper method for the constructor.
    // It is included for consistency with code that is
    // auto-generated by the Windows Forms designer in Visual Studio.
    private void InitializeComponent() {

        // A delegate for the click event of a button. The argument to
        // the constructor contains a reference to the method that performs the
        // event handling logic.
        EventHandler handler = new EventHandler(button_Click);

        // Creates three buttons, sets their properties, and attaches
        // an event handler to each button.

        red = new Button();
        red.Text = "Red";

```

```

red.Text = "Red";
red.Location = new Point(100, 50);
red.Size = new Size(50, 50);
red.Click += handler;
Controls.Add(red);

blue = new Button();
blue.Text = "Blue";
blue.Location = new Point(100, 100);
blue.Size = new Size(50, 50);
blue.Click += handler;
Controls.Add(blue);

green = new Button();
green.Text = "Green";
green.Location = new Point(100, 150);
green.Size = new Size(50, 50);
green.Click += handler;
Controls.Add(green);
}

// Event handler.
private void button_Click(object sender, EventArgs e) {
    if (sender == red) this.BackColor = Color.Red ;
        else if (sender == blue) this.BackColor = Color.Blue;
        else this.BackColor = Color.Green;
}
// The STAThreadAttribute informs the common language runtime that
// Windows Forms uses the single-threaded apartment model.
[STAThread]
public static void Main(string[] args) {
    Application.Run(new MyForm());
}
}

```

See Also

[Developing Custom Windows Forms Controls with the .NET Framework](#)

[Windows Forms Control Development Basics](#)

Varieties of Custom Controls

5/4/2018 • 5 min to read • [Edit Online](#)

With the .NET Framework, you can develop and implement new controls. You can extend the functionality of the familiar user control as well as existing controls through inheritance. You can also write custom controls that perform their own painting.

Deciding which kind of control to create can be confusing. This topic highlights the differences among the various kinds of controls from which you can inherit, and provides you with information about how to choose a particular kind of control for your project.

NOTE

For information about authoring a control to use on Web Forms, see [Developing Custom ASP.NET Server Controls](#).

Base Control Class

The [Control](#) class is the base class for Windows Forms controls. It provides the infrastructure required for visual display in Windows Forms applications.

The [Control](#) class performs the following tasks to provide visual display in Windows Forms applications:

- Exposes a window handle.
- Manages message routing.
- Provides mouse and keyboard events, and many other user interface events.
- Provides advanced layout features.
- Contains many properties specific to visual display, such as [ForeColor](#), [BackColor](#), [Height](#), and [Width](#).
- Provides the security and threading support necessary for a Windows Forms control to act as a Microsoft® ActiveX® control.

Because so much of the infrastructure is provided by the base class, it is relatively easy to develop your own Windows Forms controls.

Kinds of Controls

Windows Forms supports three kinds of user-defined controls: *composite*, *extended*, and *custom*. The following sections describe each kind of control and give recommendations for choosing the kind to use in your projects.

Composite Controls

A composite control is a collection of Windows Forms controls encapsulated in a common container. This kind of control is sometimes called a *user control*. The contained controls are called *constituent controls*.

A composite control holds all of the inherent functionality associated with each of the contained Windows Forms controls and enables you to selectively expose and bind their properties. A composite control also provides a great deal of default keyboard handling functionality with no extra development effort on your part.

For example, a composite control could be built to display customer address data from a database. This control could include a [DataGridView](#) control to display the database fields, a [BindingSource](#) to handle binding to a data

source, and a [BindingNavigator](#) control to move through the records. You could selectively expose data binding properties, and you could package and reuse the entire control from application to application. For an example of this kind of composite control, see [How to: Apply Attributes in Windows Forms Controls](#).

To author a composite control, derive from the [UserControl](#) class. The [UserControl](#) base class provides keyboard routing for child controls and enables child controls to work as a group. For more information, see [Developing a Composite Windows Forms Control](#).

Recommendation

Inherit from the [UserControl](#) class if:

- You want to combine the functionality of several Windows Forms controls into a single reusable unit.

Extended Controls

You can derive an inherited control from any existing Windows Forms control. With this approach, you can retain all of the inherent functionality of a Windows Forms control, and then extend that functionality by adding custom properties, methods, or other features. With this option, you can override the base control's paint logic, and then extend its user interface by changing its appearance.

For example, you can create a control derived from the [Button](#) control that tracks how many times a user has clicked it.

In some controls, you can also add a custom appearance to the graphical user interface of your control by overriding the [OnPaint](#) method of the base class. For an extended button that tracks clicks, you can override the [OnPaint](#) method to call the base implementation of [OnPaint](#), and then draw the click count in one corner of the [Button](#) control's client area.

Recommendation

Inherit from a Windows Forms control if:

- Most of the functionality you need is already identical to an existing Windows Forms control.
- You do not need a custom graphical user interface, or you want to design a new graphical user interface for an existing control.

Custom Controls

Another way to create a control is to create one substantially from the beginning by inheriting from [Control](#). The [Control](#) class provides all of the basic functionality required by controls, including mouse and keyboard handling events, but no control-specific functionality or graphical interface.

Creating a control by inheriting from the [Control](#) class requires much more thought and effort than inheriting from [UserControl](#) or an existing Windows Forms control. Because a great deal of implementation is left for you, your control can have greater flexibility than a composite or extended control, and you can tailor your control to suit your exact needs.

To implement a custom control, you must write code for the [OnPaint](#) event of the control, as well as any feature-specific code you need. You can also override the [WndProc](#) method and handle windows messages directly. This is the most powerful way to create a control, but to use this technique effectively, you need to be familiar with the Microsoft Win32® API.

An example of a custom control is a clock control that duplicates the appearance and behavior of an analog clock. Custom painting is invoked to cause the hands of the clock to move in response to [Tick](#) events from an internal [Timer](#) component. For more information, see [How to: Develop a Simple Windows Forms Control](#).

Recommendation

Inherit from the [Control](#) class if:

- You want to provide a custom graphical representation of your control.
- You need to implement custom functionality that is not available through standard controls.

ActiveX Controls

Although the Windows Forms infrastructure has been optimized to host Windows Forms controls, you can still use ActiveX controls. There is support for this task in Visual Studio. For more information, see [How to: Add ActiveX Controls to Windows Forms](#).

Windowless Controls

The Microsoft Visual Basic® 6.0 and ActiveX technologies support *windowless* controls. Windowless controls are not supported in Windows Forms.

Custom Design Experience

If you need to implement a custom design-time experience, you can author your own designer. For composite controls, derive your custom designer class from the [ParentControlDesigner](#) or the [DocumentDesigner](#) classes. For extended and custom controls, derive your custom designer class from the [ControlDesigner](#) class.

Use the [DesignerAttribute](#) to associate your control with your designer. For more information, see [Extending Design-Time Support](#) and [How to: Create a Windows Forms Control That Takes Advantage of Design-Time Features](#).

See Also

[Developing Custom Windows Forms Controls with the .NET Framework](#)

[How to: Develop a Simple Windows Forms Control](#)

[Developing a Composite Windows Forms Control](#)

[Extending Design-Time Support](#)

[How to: Create a Windows Forms Control That Takes Advantage of Design-Time Features](#)

Control Type Recommendations

5/4/2018 • 3 min to read • [Edit Online](#)

The .NET Framework gives you power to develop and implement new controls. In addition to the familiar user control, you will now find that you are able to write custom controls that perform their own painting, and are even able to extend the functionality of existing controls through inheritance. Deciding which type of control to create can be confusing. This section highlights the differences between the various types of controls from which you can inherit, and gives considerations regarding the type to choose for your project.

NOTE

If you want to author a control to use on Web Forms, see [Developing Custom ASP.NET Server Controls](#).

Inheriting from a Windows Forms Control

You can derive an inherited control from any existing Windows Forms control. This approach allows you to retain all of the inherent functionality of a Windows Forms control, and to then extend that functionality by adding custom properties, methods, or other functionality. For example, you might create a control derived from [TextBox](#) that can accept only numbers and automatically converts input into a value. Such a control might contain validation code that was called whenever the text in the text box changed, and could have an additional property, [Value](#). In some controls, you can also add a custom appearance to the graphical interface of your control by overriding the [OnPaint](#) method of the base class.

Inherit from a Windows Forms control if:

- Most of the functionality you need is already identical to an existing Windows Forms control.
- You do not need a custom graphical interface, or you want to design a new graphical front end for an existing control.

Inheriting from the UserControl Class

A user control is a collection of Windows Forms controls encapsulated into a common container. The container holds all of the inherent functionality associated with each of the Windows Forms controls and allows you to selectively expose and bind their properties. An example of a user control might be a control built to display customer address data from a database. This control would include several textboxes to display each field, and button controls to navigate through the records. Data-binding properties could be selectively exposed, and the entire control could be packaged and reused from application to application.

Inherit from the [UserControl](#) class if:

- You want to combine the functionality of several Windows Forms controls into a single reusable unit.

Inheriting from the Control Class

Another way to create a control is to create one substantially from scratch by inheriting from [Control](#). The [Control](#) class provides all of the basic functionality required by controls (for example, events), but no control-specific functionality or graphical interface. Creating a control by inheriting from the [Control](#) class requires a lot more thought and effort than inheriting from user control or an existing Windows Forms control. The author must write code for the [OnPaint](#) event of the control, as well as any functionality specific code that is needed. Greater flexibility is allowed, however, and you can custom tailor a control to suit your exact needs. An example of a custom control is

a clock control that duplicates the look and action of an analog clock. Custom painting would be invoked to cause the hands of the clock to move in response to [Tick](#) events from an internal timer component.

Inherit from the [Control](#) class if:

- You want to provide a custom graphical representation of your control.
- You need to implement custom functionality that is not available through standard controls.
- [How to: Display a Control in the Choose Toolbox Items Dialog Box](#)
- [Walkthrough: Serializing Collections of Standard Types with the DesignerSerializationVisibilityAttribute](#)
- [Walkthrough: Inheriting from a Windows Forms Control with Visual C#](#)
- [How to: Provide a Toolbox Bitmap for a Control](#)
- [How to: Inherit from Existing Windows Forms Controls](#)
- [Walkthrough: Debugging Custom Windows Forms Controls at Design Time](#)
- [How to: Inherit from the Control Class](#)
- [How to: Test the Run-Time Behavior of a UserControl](#)
- [How to: Align a Control to the Edges of Forms at Design Time](#)
- [How to: Inherit from the UserControl Class](#)
- [How to: Author Controls for Windows Forms](#)
- [How to: Author Composite Controls](#)
- [Walkthrough: Authoring a Composite Control with Visual Basic](#)
- [Walkthrough: Authoring a Composite Control with Visual C#](#)
- [Walkthrough: Inheriting from a Windows Forms Control with Visual Basic](#)
- [How to: Create a Windows Forms Control That Takes Advantage of Design-Time Features](#)
- [How to: Create a Windows Forms Control That Takes Advantage of Design-Time Features](#)

See Also

[How to: Develop a Simple Windows Forms Control](#)

[Varieties of Custom Controls](#)

Windows Forms Control Development Basics

5/4/2018 • 1 min to read • [Edit Online](#)

A Windows Forms control is a class that derives directly or indirectly from [System.Windows.Forms.Control](#). The following list describes common scenarios for developing Windows Forms controls:

- Combining existing controls to author a composite control.

Composite controls encapsulate a user interface that can be reused as a control. An example of a composite control is a control that consists of a text box and a reset button. Visual designers offer rich support for creating composite controls. To author a composite control, derive from [System.Windows.Forms.UserControl](#). The base class [UserControl](#) provides keyboard routing for child controls and enables child controls to work as a group. For more information, see [Developing a Composite Windows Forms Control](#).

- Extending an existing control to customize it or to add to its functionality.

A button whose color cannot be changed and a button that has an additional property that tracks how many times it has been clicked are examples of extended controls. You can customize any Windows Forms control by deriving from it and overriding or adding properties, methods, and events.

- Authoring a control that does not combine or extend existing controls.

In this scenario, derive your control from the base class [Control](#). You can add as well as override properties, methods, and events of the base class. To get started, see [How to: Develop a Simple Windows Forms Control](#).

The base class for Windows Forms controls, [Control](#), provides the plumbing required for visual display in client-side Windows-based applications. [Control](#) provides a window handle, handles message routing, and provides mouse and keyboard events as well as many other user interface events. It provides advanced layout and has properties specific to visual display, such as [ForeColor](#), [BackColor](#), [Height](#), [Width](#), and many others. Additionally, it provides security, threading support, and interoperability with ActiveX controls. Because so much of the infrastructure is provided by the base class, it is relatively easy to develop your own Windows Forms controls.

See Also

- [How to: Develop a Simple Windows Forms Control](#)
- [Developing a Composite Windows Forms Control](#)
- [How to: Create a Windows Forms Control That Shows Progress](#)
- [Varieties of Custom Controls](#)

How to: Develop a Simple Windows Forms Control

5/4/2018 • 7 min to read • [Edit Online](#)

This section walks you through the key steps for authoring a custom Windows Forms control. The simple control developed in this walkthrough allows the alignment of its [Text](#) property to be changed. It does not raise or handle events.

To create a simple custom control

1. Define a class that derives from [System.Windows.Forms.Control](#).

```
Public Class FirstControl  
    Inherits Control  
  
End Class
```

```
public class FirstControl:Control{}
```

2. Define properties. (You are not required to define properties, because a control inherits many properties from the [Control](#) class, but most custom controls generally do define additional properties.) The following code fragment defines a property named [TextAlignment](#) that [FirstControl](#) uses to format the display of the [Text](#) property inherited from [Control](#). For more information about defining properties, see [Properties Overview](#).

```
// ContentAlignment is an enumeration defined in the System.Drawing  
// namespace that specifies the alignment of content on a drawing  
// surface.  
private ContentAlignment alignmentValue = ContentAlignment.MiddleLeft;
```

```
' ContentAlignment is an enumeration defined in the System.Drawing  
' namespace that specifies the alignment of content on a drawing  
' surface.  
Private alignmentValue As ContentAlignment = ContentAlignment.MiddleLeft  
  
<Category("Alignment"), Description("Specifies the alignment of text.")> _  
Public Property TextAlign() As ContentAlignment  
  
    Get  
        Return alignmentValue  
    End Get  
    Set  
        alignmentValue = value  
  
        ' The Invalidate method invokes the OnPaint method described  
        ' in step 3.  
        Invalidate()  
    End Set  
End Property
```

When you set a property that changes the visual display of the control, you must invoke the [Invalidate](#) method to redraw the control. [Invalidate](#) is defined in the base class [Control](#).

3. Override the protected [OnPaint](#) method inherited from [Control](#) to provide rendering logic to your control.

If you do not override [OnPaint](#), your control will not be able to draw itself. In the following code fragment, the [OnPaint](#) method displays the [Text](#) property inherited from [Control](#) with the alignment specified by the [alignmentValue](#) field.

```
protected override void OnPaint(PaintEventArgs e)
{
    base.OnPaint(e);
    StringFormat style = new StringFormat();
    style.Alignment = StringAlignment.Near;
    switch (alignmentValue)
    {
        case ContentAlignment.MiddleLeft:
            style.Alignment = StringAlignment.Near;
            break;
        case ContentAlignment.MiddleRight:
            style.Alignment = StringAlignment.Far;
            break;
        case ContentAlignment.MiddleCenter:
            style.Alignment = StringAlignment.Center;
            break;
    }

    // Call the DrawString method of the System.Drawing class to write
    // text. Text and ClientRectangle are properties inherited from
    // Control.
    e.Graphics.DrawString(
        Text,
        Font,
        new SolidBrush(ForeColor),
        ClientRectangle, style);

}
```

```
Protected Overrides Sub OnPaint(e As PaintEventArgs)

    MyBase.OnPaint(e)
    Dim style As New StringFormat()
    style.Alignment = StringAlignment.Near
    Select Case alignmentValue
        Case ContentAlignment.MiddleLeft
            style.Alignment = StringAlignment.Near
        Case ContentAlignment.MiddleRight
            style.Alignment = StringAlignment.Far
        Case ContentAlignment.MiddleCenter
            style.Alignment = StringAlignment.Center
    End Select

    ' Call the DrawString method of the System.Drawing class to write
    ' text. Text and ClientRectangle are properties inherited from
    ' Control.
    e.Graphics.DrawString( _
        me.Text, _
        me.Font, _
        New SolidBrush(ForeColor), _
        RectangleF.op_Implicit(ClientRectangle), _
        style)

End Sub
```

4. Provide attributes for your control. Attributes enable a visual designer to display your control and its properties and events appropriately at design time. The following code fragment applies attributes to the [TextAlignment](#) property. In a designer such as Visual Studio, the [Category](#) attribute (shown in the code fragment) causes the property to be displayed under a logical category. The [Description](#) attribute causes a

descriptive string to be displayed at the bottom of the **Properties** window when the `TextAlignment` property is selected. For more information about attributes, see [Design-Time Attributes for Components](#).

```
[  
Category("Alignment"),  
Description("Specifies the alignment of text.")  
]
```

```
<Category("Alignment"), Description("Specifies the alignment of text.")> _  
Public Property TextAlignment() As ContentAlignment
```

5. (optional) Provide resources for your control. You can provide a resource, such as a bitmap, for your control by using a compiler option (`/res` for C#) to package resources with your control. At run time, the resource can be retrieved using the methods of the [ResourceManager](#) class. For more information about creating and using resources, see the [Resources in Desktop Apps](#).

6. Compile and deploy your control. To compile and deploy `FirstControl`, execute the following steps:

- Save the code in the following sample to a source file (such as `FirstControl.cs` or `FirstControl.vb`).
- Compile the source code into an assembly and save it in your application's directory. To accomplish this, execute the following command from the directory that contains the source file.

```
vbc -t:library -out:[path to your application's directory]/CustomWinControls.dll -r:System.dll -  
r:System.Windows.Forms.dll -r:System.Drawing.dll FirstControl.vb
```

```
csc -t:library -out:[path to your application's directory]/CustomWinControls.dll -r:System.dll -  
r:System.Windows.Forms.dll -r:System.Drawing.dll FirstControl.cs
```

The `/t:library` compiler option tells the compiler that the assembly you are creating is a library (and not an executable). The `/out` option specifies the path and name of the assembly. The `/r` option provides the name of the assemblies that are referenced by your code. In this example, you create a private assembly that only your applications can use. Hence, you have to save it in your application's directory. For more information about packaging and deploying a control for distribution, see [Deployment](#).

The following sample shows the code for `FirstControl`. The control is enclosed in the namespace `CustomWinControls`. A namespace provides a logical grouping of related types. You can create your control in a new or existing namespace. In C#, the `using` declaration (in Visual Basic, `Imports`) allows types to be accessed from a namespace without using the fully qualified name of the type. In the following example, the `using` declaration allows code to access the class `Control` from `System.Windows.Forms` as simply `Control` instead of having to use the fully qualified name `System.Windows.Forms.Control`.

```
using System;  
using System.Drawing;  
using System.Collections;  
using System.ComponentModel;  
using System.Windows.Forms;  
  
namespace CustomWinControls  
{  
    public class FirstControl : Control  
    {  
  
        public FirstControl()  
    }  
}
```

```

{
}

// ContentAlignment is an enumeration defined in the System.Drawing
// namespace that specifies the alignment of content on a drawing
// surface.
private ContentAlignment alignmentValue = ContentAlignment.MiddleLeft;

[
Category("Alignment"),
Description("Specifies the alignment of text.")
]
public ContentAlignment TextAlignement
{

    get
    {
        return alignmentValue;
    }
    set
    {
        alignmentValue = value;

        // The Invalidate method invokes the OnPaint method described
        // in step 3.
        Invalidate();
    }
}

protected override void OnPaint(PaintEventArgs e)
{
    base.OnPaint(e);
    StringFormat style = new StringFormat();
    style.Alignment = StringAlignment.Near;
    switch (alignmentValue)
    {
        case ContentAlignment.MiddleLeft:
            style.Alignment = StringAlignment.Near;
            break;
        case ContentAlignment.MiddleRight:
            style.Alignment = StringAlignment.Far;
            break;
        case ContentAlignment.MiddleCenter:
            style.Alignment = StringAlignment.Center;
            break;
    }

    // Call the DrawString method of the System.Drawing class to write
    // text. Text and ClientRectangle are properties inherited from
    // Control.
    e.Graphics.DrawString(
        Text,
        Font,
        new SolidBrush(ForeColor),
        ClientRectangle, style);

}
}
}

```

```

Imports System
Imports System.Drawing
Imports System.Collections
Imports System.ComponentModel
Imports System.Windows.Forms

Public Class FirstControl
    Inherits Control

    Public Sub New()
        End Sub

    ' ContentAlignment is an enumeration defined in the System.Drawing
    ' namespace that specifies the alignment of content on a drawing
    ' surface.
    Private alignmentValue As ContentAlignment = ContentAlignment.MiddleLeft

    <Category("Alignment"), Description("Specifies the alignment of text.")> _
    Public Property TextAlign() As ContentAlignment

        Get
            Return alignmentValue
        End Get
        Set
            alignmentValue = value

            ' The Invalidate method invokes the OnPaint method described
            ' in step 3.
            Invalidate()
        End Set
    End Property

    Protected Overrides Sub OnPaint(e As PaintEventArgs)

        MyBase.OnPaint(e)
        Dim style As New StringFormat()
        style.Alignment = StringAlignment.Near
        Select Case alignmentValue
            Case ContentAlignment.MiddleLeft
                style.Alignment = StringAlignment.Near
            Case ContentAlignment.MiddleRight
                style.Alignment = StringAlignment.Far
            Case ContentAlignment.MiddleCenter
                style.Alignment = StringAlignment.Center
        End Select

        ' Call the DrawString method of the System.Drawing class to write
        ' text. Text and ClientRectangle are properties inherited from
        ' Control.
        e.Graphics.DrawString( _
            me.Text, _
            me.Font, _
            New SolidBrush(ForeColor), _
            RectangleF.op_Implicit(ClientRectangle), _
            style)
    End Sub
End Class

```

Using the Custom Control on a Form

The following example shows a simple form that uses `FirstControl`. It creates three instances of `FirstControl`, each with a different value for the `ContentAlignment` property.

To compile and run this sample

1. Save the code in the following example to a source file (SimpleForm.cs or SimpleForms.vb).
2. Compile the source code into an executable assembly by executing the following command from the directory that contains the source file.

```
vbc -r:CustomWinControls.dll -r:System.dll -r:System.Windows.Forms.dll -r:System.Drawing.dll  
SimpleForm.vb
```

```
csc -r:CustomWinControls.dll -r:System.dll -r:System.Windows.Forms.dll -r:System.Drawing.dll  
SimpleForm.cs
```

`CustomWinControls.dll` is the assembly that contains the class `FirstControl`. This assembly must be in the same directory as the source file for the form that accesses it (SimpleForm.cs or SimpleForms.vb).

3. Execute `SimpleForm.exe` using the following command.

```
SimpleForm
```

```
using System;  
using System.Drawing;  
using System.Collections;  
using System.ComponentModel;  
using System.Windows.Forms;  
  
namespace CustomWinControls  
{  
  
    public class SimpleForm : System.Windows.Forms.Form  
    {  
        private FirstControl firstControl1;  
  
        private System.ComponentModel.Container components = null;  
  
        public SimpleForm()  
        {  
            InitializeComponent();  
        }  
  
        protected override void Dispose( bool disposing )  
        {  
            if( disposing )  
            {  
                if (components != null)  
                {  
                    components.Dispose();  
                }  
            }  
            base.Dispose( disposing );  
        }  
  
        private void InitializeComponent()  
        {  
            this.firstControl1 = new FirstControl();  
            this.SuspendLayout();  
  
            //  
            // firstControl1  
        }
```

```
//  
this.firstControl1.BackColor = System.Drawing.SystemColors.ControlDark;  
this.firstControl1.Location = new System.Drawing.Point(96, 104);  
this.firstControl1.Name = "firstControl1";  
this.firstControl1.Size = new System.Drawing.Size(75, 16);  
this.firstControl1.TabIndex = 0;  
this.firstControl1.Text = "Hello World";  
this.firstControl1.TextAlignment = System.Drawing.ContentAlignment.MiddleCenter;  
  
//  
// SimpleForm  
//  
this.ClientSize = new System.Drawing.Size(292, 266);  
this.Controls.Add(this.firstControl1);  
this.Name = "SimpleForm";  
this.Text = "SimpleForm";  
this.ResumeLayout(false);  
  
}  
  
[STAThread]  
static void Main()  
{  
    Application.Run(new SimpleForm());  
}  
  
}  
}
```

```

Imports System
Imports System.Drawing
Imports System.Collections
Imports System.ComponentModel
Imports System.Windows.Forms

Public Class SimpleForm
    Inherits System.Windows.Forms.Form

    Private firstControl1 As FirstControl

    Private components As System.ComponentModel.Container = Nothing

    Public Sub New()
        InitializeComponent()
    End Sub

    Private Sub InitializeComponent()
        Me.firstControl1 = New FirstControl()
        Me.SuspendLayout()

        ' firstControl1
        ' 

        Me.firstControl1.BackColor = System.Drawing.SystemColors.ControlDark
        Me.firstControl1.Location = New System.Drawing.Point(96, 104)
        Me.firstControl1.Name = "firstControl1"
        Me.firstControl1.Size = New System.Drawing.Size(75, 16)
        Me.firstControl1.TabIndex = 0
        Me.firstControl1.Text = "Hello World"
        Me.firstControl1.TextAlignment = System.Drawing.ContentAlignment.MiddleCenter

        ' SimpleForm
        ' 

        Me.ClientSize = New System.Drawing.Size(292, 266)
        Me.Controls.Add(firstControl1)
        Me.Name = "SimpleForm"
        Me.Text = "SimpleForm"
        Me.ResumeLayout(False)
    End Sub

    <STAThread()> _
Shared Sub Main()
    Application.Run(New SimpleForm())
End Sub
End Class

```

See Also

[Properties in Windows Forms Controls](#)

[Events in Windows Forms Controls](#)

How to: Create a Windows Forms Control That Shows Progress

5/4/2018 • 19 min to read • [Edit Online](#)

The following code example shows a custom control called `FlashTrackBar` that can be used to show the user the level or the progress of an application. It uses a gradient to visually represent progress.

The `FlashTrackBar` control illustrates the following concepts:

- Defining custom properties.
- Defining custom events. (`FlashTrackBar` defines the `ValueChanged` event.)
- Overriding the `OnPaint` method to provide logic to draw the control.
- Computing the area available for drawing the control by using its `ClientRect` property. `FlashTrackBar` does this in its `OptimizedInvalidate` method.
- Implementing serialization or persistence for a property when it is changed in the Windows Forms Designer. `FlashTrackBar` defines the `ShouldSerializeStartColor` and `ShouldSerializeEndColor` methods for serializing its `StartColor` and `EndColor` properties.

The following table shows the custom properties defined by `FlashTrackBar`.

PROPERTY	DESCRIPTION
<code>AllowUserEdit</code>	Indicates whether the user can change the value of the flash track bar by clicking and dragging it.
<code>EndColor</code>	Specifies the ending color of the track bar.
<code>DarkenBy</code>	Specifies how much to darken the background with respect to the foreground gradient.
<code>Max</code>	Specifies the maximum value of the track bar.
<code>Min</code>	Specifies the minimum value of the track bar.
<code>StartColor</code>	Specifies the starting color of the gradient.
<code>ShowPercentage</code>	Indicates whether to display a percentage over the gradient.
<code>ShowValue</code>	Indicates whether to display the current value over the gradient.
<code>ShowGradient</code>	Indicates whether the track bar should display a color gradient showing the current value.
- <code>Value</code>	Specifies the current value of the track bar.

The following table shows additional members defined by `FlashTrackBar`: the property-changed event and the

method that raises the event.

MEMBER	DESCRIPTION
<code>ValueChanged</code>	The event that is raised when the <code>Value</code> property of the track bar changes.
<code>OnValueChanged</code>	The method that raises the <code>ValueChanged</code> event.

NOTE

`FlashTrackBar` uses the `EventArgs` class for event data and `EventHandler` for the event delegate.

To handle the corresponding *EventName* events, `FlashTrackBar` overrides the following methods that it inherits from `System.Windows.Forms.Control`:

- `OnPaint`
- `OnMouseDown`
- `OnMouseMove`
- `OnMouseUp`
- `OnResize`

To handle the corresponding property-changed events, `FlashTrackBar` overrides the following methods that it inherits from `System.Windows.Forms.Control`:

- `OnBackColorChanged`
- `OnBackgroundImageChanged`
- `OnTextChanged`

Example

The `FlashTrackBar` control defines two UI type editors, `FlashTrackBarValueEditor` and `FlashTrackBarDarkenByEditor`, which are shown in the following code listings. The `HostApp` class uses the `FlashTrackBar` control on a Windows Form.

```
namespace Microsoft.Samples.WinForms.Cs.FlashTrackBar {
    using System;
    using System.ComponentModel;
    using System.ComponentModel.Design;
    using System.Drawing;
    using System.Drawing.Drawing2D;
    using System.Drawing.Design;
    using System.Windows.Forms;
    using System.Diagnostics;

    [System.Security.Permissions.PermissionSet(System.Security.Permissions.SecurityAction.Demand, Name =
"FullTrust")]
    public class FlashTrackBar : System.Windows.Forms.Control {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Container components;

        private const int LeftRightBorder = 10;
```

```

private int value = 0;
private int min = 0;
private int max = 100;
private bool showPercentage = false;
private bool showValue = false;
private bool allowUserEdit = true;
private bool showGradient = true;
private int dragValue = 0;
private bool dragging = false;
private Color startColor = Color.Red;
private Color endColor = Color.LimeGreen;
private EventHandler onValueChanged;
private Brush baseBackground = null;
private Brush backgroundDim = null;
private byte darkenBy = 200;

public FlashTrackBar() {
    //
    // Required for Windows Form Designer support
    //
    InitializeComponent();

    SetStyle(ControlStyles.Opaque, true);
    SetStyle(ControlStyles.ResizeRedraw, true);
    Debug.Assert(GetStyle(ControlStyles.ResizeRedraw), "Should be redraw!");
}

/// <summary>
///     Clean up any resources being used.
/// </summary>
protected override void Dispose(bool disposing)
{
    if (disposing) {
        if (components != null) {
            components.Dispose();
        }
    }
    base.Dispose(disposing);
}

/// <summary>
///     Required method for Designer support - do not modify
///     the contents of this method with the code editor.
/// </summary>
void InitializeComponent () {
    this.components = new System.ComponentModel.Container ();
    this.ForeColor = System.Drawing.Color.White;
    this.BackColor = System.Drawing.Color.Black;
    this.Size = new System.Drawing.Size(100, 23);
    this.Text = "FlashTrackBar";
}

[
    Category("Flash"),
    DefaultValue(true)
]
public bool AllowUserEdit {
    get {
        return allowUserEdit;
    }
    set {
        if (value != allowUserEdit) {
            allowUserEdit = value;
            if (!allowUserEdit) {
                Capture = false;
                dragging = false;
            }
        }
    }
}

```

```

        }

    }

    [
        Category("Flash")
    ]
    public Color EndColor {
        get {
            return endColor;
        }
        set {
            endColor = value;
            if (baseBackground != null && showGradient) {
                baseBackground.Dispose();
                baseBackground = null;
            }
            Invalidate();
        }
    }

    public bool ShouldSerializeEndColor() {
        return !(endColor == Color.LimeGreen);
    }

    [
        Category("Flash"),
        Editor(typeof(FlashTrackBarDarkenByEditor), typeof(UITypeEditor)),
        DefaultValue((byte)200)
    ]
    public byte DarkenBy {
        get {
            return darkenBy;
        }
        set {
            if (value != darkenBy) {
                darkenBy = value;
                if (backgroundDim != null) {
                    backgroundDim.Dispose();
                    backgroundDim = null;
                }
                OptimizedInvalidate(Value, max);
            }
        }
    }

    [
        Category("Flash"),
        DefaultValue(100)
    ]
    public int Max {
        get {
            return max;
        }
        set {
            if (max != value) {
                max = value;
                Invalidate();
            }
        }
    }

    [
        Category("Flash"),
        DefaultValue(0)
    ]
    public int Min {
        get {
            return min;
        }
    }
}

```

```

        }
        set {
            if (min != value) {
                min = value;
                Invalidate();
            }
        }
    }

    [
        Category("Flash")
    ]
    public Color StartColor {
        get {
            return startColor;
        }
        set {
            startColor = value;
            if (baseBackground != null && showGradient) {
                baseBackground.Dispose();
                baseBackground = null;
            }
            Invalidate();
        }
    }

    public bool ShouldSerializeStartColor() {
        return !(startColor == Color.Red);
    }

    [
        Category("Flash"),
        RefreshProperties(RefreshProperties.Repaint),
        DefaultValue(false)
    ]
    public bool ShowPercentage {
        get {
            return showPercentage;
        }
        set {
            if (value != showPercentage) {
                showPercentage = value;
                if (showPercentage) {
                    showValue = false;
                }
                Invalidate();
            }
        }
    }

    [
        Category("Flash"),
        RefreshProperties(RefreshProperties.Repaint),
        DefaultValue(false)
    ]
    public bool ShowValue {
        get {
            return showValue;
        }
        set {
            if (value != showValue) {
                showValue = value;
                if (showValue) {
                    showPercentage = false;
                }
                Invalidate();
            }
        }
    }
}

```

```

        }

    }

    [
        Category("Flash"),
        DefaultValue(true)
    ]
    public bool ShowGradient {
        get {
            return showGradient;
        }
        set {
            if (value != showGradient) {
                showGradient = value;
                if (baseBackground != null) {
                    baseBackground.Dispose();
                    baseBackground = null;
                }
                Invalidate();
            }
        }
    }

    [
        Category("Flash"),
        Editor(typeof(FlashTrackBarValueEditor), typeof(UITypeEditor)),
        DefaultValue(0)
    ]
    public int Value {
        get {
            if (dragging) {
                return dragValue;
            }
            return value;
        }
        set {
            if (value != this.value) {
                int old = this.value;
                this.value = value;
                OnValueChangedEventArgs.Empty);
                OptimizedInvalidate(old, this.value);
            }
        }
    }

    // ValueChanged Event
    [Description("Raised when the Value displayed changes")]
    public event EventHandler ValueChanged {
        add {
            onValueChanged += value;
        }
        remove {
            onValueChanged -= value;
        }
    }

    private void OptimizedInvalidate(int oldValue, int newValue) {
        Rectangle client = ClientRectangle;

        float oldPercentValue = ((float)oldValue / ((float)Max - (float)Min));
        int oldNonDimLength = (int)(oldPercentValue * (float)client.Width);

        float newPercentValue = ((float)newValue / ((float)Max - (float)Min));
        int newNonDimLength = (int)(newPercentValue * (float)client.Width);

        int min = Math.Min(oldNonDimLength, newNonDimLength);
        int max = Math.Max(oldNonDimLength, newNonDimLength);

        Rectangle invalid = new Rectangle(

```

```

        Rectangle invalid = new Rectangle(
            client.X + min,
            client.Y,
            max - min,
            client.Height);

        Invalidate(invalid);

        string oldToDisplay;
        string newToDisplay;

        if (ShowPercentage) {
            oldToDisplay = Convert.ToString((int)(oldPercentValue * 100f)) + "%";
            newToDisplay = Convert.ToString((int)(newPercentValue * 100f)) + "%";
        }
        else if (ShowValue) {
            oldToDisplay = Convert.ToString(oldValue);
            newToDisplay = Convert.ToString(newValue);
        }
        else {
            oldToDisplay = null;
            newToDisplay = null;
        }

        if (oldToDisplay != null && newToDisplay != null) {
            Graphics g = CreateGraphics();
            SizeF oldFontSize = g.MeasureString(oldToDisplay, Font);
            SizeF newFontSize = g.MeasureString(newToDisplay, Font);
            RectangleF oldFontRect = new RectangleF(new PointF(0, 0), oldFontSize);
            RectangleF newFontRect = new RectangleF(new PointF(0, 0), newFontSize);
            oldFontRect.X = (client.Width - oldFontRect.Width) / 2;
            oldFontRect.Y = (client.Height - oldFontRect.Height) / 2;
            newFontRect.X = (client.Width - newFontRect.Width) / 2;
            newFontRect.Y = (client.Height - newFontRect.Height) / 2;

            Invalidate(new Rectangle((int)oldFontRect.X, (int)oldFontRect.Y, (int)oldFontRect.Width,
            (int)oldFontRect.Height));
            Invalidate(new Rectangle((int)newFontRect.X, (int)newFontRect.Y, (int)newFontRect.Width,
            (int)newFontRect.Height));
        }
    }

    protected override void OnMouseDown(MouseEventArgs e) {
        base.OnMouseDown(e);
        if (!allowUserEdit) {
            return;
        }
        Capture = true;
        dragging = true;
        SetDragValue(new Point(e.X, e.Y));
    }

    protected override void OnMouseMove(MouseEventArgs e) {
        base.OnMouseMove(e);
        if (!allowUserEdit || !dragging) {
            return;
        }
        SetDragValue(new Point(e.X, e.Y));
    }

    protected override void OnMouseUp(MouseEventArgs e) {
        base.OnMouseUp(e);
        if (!allowUserEdit || !dragging) {
            return;
        }
        Capture = false;
        dragging = false;
        value = dragValue;
        OnValueChanged(EventArgs.Empty);
    }
}

```

```

}

protected override void OnPaint(PaintEventArgs e) {

    base.OnPaint(e);
    if (baseBackground == null) {
        if (showGradient) {
            baseBackground = new LinearGradientBrush(new Point(0, 0),
                new Point(ClientSize.Width, 0),
                StartColor,
                EndColor);
        }
        else if (BackgroundImage != null) {
            baseBackground = new TextureBrush(BackgroundImage);
        }
        else {
            baseBackground = new SolidBrush(BackColor);
        }
    }

    if (backgroundDim == null) {
        backgroundDim = new SolidBrush(Color.FromArgb(DarkenBy, Color.Black));
    }

    Rectangle toDim = ClientRectangle;
    float percentValue = ((float)Value / ((float)Max - (float)Min));
    int nonDimLength = (int)(percentValue * (float)toDim.Width);
    toDim.X += nonDimLength;
    toDim.Width -= nonDimLength;

    string text = Text;
    string toDisplay = null;
    RectangleF textRect = new RectangleF();

    if (ShowPercentage || ShowValue || text.Length > 0) {

        if (ShowPercentage) {
            toDisplay = Convert.ToString((int)(percentValue * 100f)) + "%";
        }
        else if (ShowValue) {
            toDisplay = Convert.ToString(Value);
        }
        else {
            toDisplay = text;
        }

        SizeF textSize = e.Graphics.MeasureString(toDisplay, Font);
        textRect.Width = textSize.Width;
        textRect.Height = textSize.Height;
        textRect.X = (ClientRectangle.Width - textRect.Width) / 2;
        textRect.Y = (ClientRectangle.Height - textRect.Height) / 2;
    }

    e.Graphics.FillRectangle(baseBackground, ClientRectangle);
    e.Graphics.FillRectangle(backgroundDim, toDim);
    e.Graphics.Flush();
    if (toDisplay != null && toDisplay.Length > 0) {
        e.Graphics.DrawString(toDisplay, Font, new SolidBrush(ForeColor), textRect);
    }
}

protected override void OnTextChanged(EventArgs e) {
    base.OnTextChanged(e);
    Invalidate();
}

protected override void OnBackColorChanged(EventArgs e) {
    base.OnBackColorChanged(e);
}

```

```

        if ((baseBackground != null) && (!showGradient)) {
            baseBackground.Dispose();
            baseBackground = null;
        }
    }

protected override void OnBackgroundImageChanged(EventArgs e) {
    base.OnTextChanged(e);
    if ((baseBackground != null) && (!showGradient)) {
        baseBackground.Dispose();
        baseBackground = null;
    }
}

protected override void OnResize(EventArgs e) {
    base.OnResize(e);
    if (baseBackground != null) {
        baseBackground.Dispose();
        baseBackground = null;
    }
}

protected virtual void OnValueChanged(EventArgs e) {
    if (onValueChanged != null) {
        onValueChanged.Invoke(this, e);
    }
}

private void SetDragValue(Point mouseLocation) {

    Rectangle client = ClientRectangle;

    if (client.Contains(mouseLocation)) {
        float percentage = (float)mouseLocation.X / (float)ClientRectangle.Width;
        int newDragValue = (int)(percentage * (float)(max - min));
        if (newDragValue != dragValue) {
            int old = dragValue;
            dragValue = newDragValue;
            OptimizedInvalidate(old, dragValue);
        }
    }
    else {
        if (client.Y <= mouseLocation.Y && mouseLocation.Y <= client.Y + client.Height) {
            if (mouseLocation.X <= client.X && mouseLocation.X > client.X - LeftRightBorder) {
                int newDragValue = min;
                if (newDragValue != dragValue) {
                    int old = dragValue;
                    dragValue = newDragValue;
                    OptimizedInvalidate(old, dragValue);
                }
            }
            else if (mouseLocation.X >= client.X + client.Width && mouseLocation.X < client.X + client.Width + LeftRightBorder) {
                int newDragValue = max;
                if (newDragValue != dragValue) {
                    int old = dragValue;
                    dragValue = newDragValue;
                    OptimizedInvalidate(old, dragValue);
                }
            }
        }
        else {
            if (dragValue != value) {
                int old = dragValue;
                dragValue = value;
                OptimizedInvalidate(old, dragValue);
            }
        }
    }
}

```

```
    }
}
```

```
Imports System
Imports System.ComponentModel
Imports System.ComponentModel.Design
Imports System.Drawing
Imports System.Drawing.Drawing2D
Imports System.Drawing.Design
Imports System.Windows.Forms
Imports System.Diagnostics

Namespace Microsoft.Samples.WinForms.VB.FlashTrackBar

<System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.Demand,
Name:="FullTrust")> _
Public Class FlashTrackBar
Inherits System.Windows.Forms.Control

Private LeftRightBorder As Integer = 10
Private myValue As Integer = 0
Private myMin As Integer = 0
Private myMax As Integer = 100
Private myShowPercentage As Boolean = False
Private myShowValue As Boolean = False
Private myAllowUserEdit As Boolean = True
Private myShowGradient As Boolean = True
Private dragValue As Integer = 0
Private dragging As Boolean = False
Private myStartColor As Color = Color.Red
Private myEndColor As Color = Color.LimeGreen
Private baseBackground As Brush
Private backgroundDim As Brush
Private myDarkenBy As Byte = 200

Public Sub New()

 MyBase.New()

 'This call is required by the Windows Forms Designer.
 InitializeComponent()

 SetStyle(ControlStyles.Opaque, True)
 SetStyle(ControlStyles.ResizeRedraw, True)
 Debug.Assert(GetStyle(ControlStyles.ResizeRedraw), "Should be redraw!")
End Sub

'Form overrides dispose to clean up the component list.
Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing Then
        If (components IsNot Nothing) Then
            components.Dispose()
        End If
    End If
    MyBase.Dispose(disposing)
End Sub

#Region " Windows Form Designer generated code "

'Required by the Windows Form Designer
Private components As System.ComponentModel.Container

'NOTE: The following procedure is required by the Windows Form Designer
'It can be modified using the Windows Form Designer.
'Do not modify it using the code editor.

```

```

Private Sub InitializeComponent()
    Me.components = New System.ComponentModel.Container()
    Me.ForeColor = System.Drawing.Color.White
    Me.BackColor = System.Drawing.Color.Black
    Me.Size = New System.Drawing.Size(100, 23)
    Me.Text = "FlashTrackBar"
End Sub

#End Region

<Category("Flash"), DefaultValue(True)> Public Property AllowUserEdit() As Boolean
    Get
        Return myAllowUserEdit
    End Get

    Set(ByVal Value As Boolean)
        If (Value <> myAllowUserEdit) Then
            myAllowUserEdit = Value
            If Not (myAllowUserEdit) Then
                Capture = False
                dragging = False
            End If
        End If
    End Set
End Property

<Category("Flash")> Public Property EndColor() As Color
    Get
        Return myEndColor
    End Get

    Set(ByVal Value As Color)
        myEndColor = Value
        If ((baseBackground IsNot Nothing) And myShowGradient) Then
            baseBackground.Dispose()
            baseBackground = Nothing
        End If
        Invalidate()
    End Set
End Property

Public Function ShouldPersistEndColor() As Boolean
    Return Not (myEndColor.Equals(Color.LimeGreen))
End Function

<Category("Flash"), _
Editor(GetType(FlashTrackBarDarkenByEditor), GetType(UITypeEditor)), _
DefaultValue(200)> _
Public Property DarkenBy() As Byte
    Get
        Return myDarkenBy
    End Get
    Set(ByVal Value As Byte)
        If (Value <> myDarkenBy) Then
            myDarkenBy = Value
            If (backgroundDim IsNot Nothing) Then
                backgroundDim.Dispose()
                backgroundDim = Nothing
            End If
            OptimizedInvalidate(Value, Max)
        End If
    End Set
End Property

```

```

<Category("Flash"), DefaultValue(100)> _
Public Property Max() As Integer
    Get
        Return myMax
    End Get
    Set(ByVal Value As Integer)
        If (myMax <> Value) Then
            myMax = Value
            Invalidate()
        End If
    End Set
End Property

<Category("Flash"), DefaultValue(0)> _
Public Property Min() As Integer
    Get
        Return myMin
    End Get
    Set(ByVal Value As Integer)
        If (Min <> Value) Then
            Min = Value
            Invalidate()
        End If
    End Set
End Property

<Category("Flash")> Public Property StartColor() As Color
    Get
        Return myStartColor
    End Get
    Set(ByVal Value As Color)
        myStartColor = Value
        If ((baseBackground IsNot Nothing) And myShowGradient) Then
            baseBackground.Dispose()
            baseBackground = Nothing
        End If
        Invalidate()
    End Set
End Property

Public Function ShouldPersistStartColor() As Boolean
    Return Not (StartColor.Equals(Color.Red))
End Function

<Category("Flash"), _
RefreshProperties(RefreshProperties.Repaint), _
DefaultValue(False)> _
Public Property ShowPercentage() As Boolean
    Get
        Return myShowPercentage
    End Get
    Set(ByVal Value As Boolean)
        If (Value <> myShowPercentage) Then
            myShowPercentage = Value
            If (myShowPercentage) Then
                myShowValue = False
            End If
            Invalidate()
        End If
    End Set
End Property

<Category("Flash"), _
RefreshProperties(RefreshProperties.Repaint), _
DefaultValue(False)> _

```

```

Public Property ShowValue() As Boolean
    Get
        Return myShowValue
    End Get
    Set(ByVal Value As Boolean)
        If (Value <> myShowValue) Then
            myShowValue = Value
            If (myShowValue) Then
                myShowPercentage = False
            End If
            Invalidate()
        End If
    End Set
End Property

<Category("Flash"), DefaultValue(True)> _
Public Property ShowGradient() As Boolean
    Get
        Return myShowGradient
    End Get
    Set(ByVal Value As Boolean)
        If (Value <> myShowGradient) Then
            myShowGradient = Value
            If (baseBackground IsNot Nothing) Then
                baseBackground.Dispose()
                baseBackground = Nothing
            End If
            Invalidate()
        End If
    End Set
End Property

<Category("flash"), _
Editor(GetType(FlashTrackBarValueEditor), GetType(UITypeEditor)), _
DefaultValue(0)> _
Public Property Value() As Integer
    Get
        If (dragging) Then
            Return dragValue
        End If
        Return myValue
    End Get
    Set(ByVal Value As Integer)
        If (Value <> myValue) Then
            Dim old As Integer = myValue
            myValue = Value
            OnValueChangedEventArgs.Empty)
            OptimizedInvalidate(old, myValue)
        End If
    End Set
End Property

' ValueChanged Event

<Description("Raised when the Value displayed changes")> _
Public Event ValueChanged(ByVal sender As Object, ByVal ev As EventArgs) 'As EventHandler

Private Sub OptimizedInvalidate(ByVal oldValue As Integer, ByVal newValue As Integer)
    Dim client As Rectangle = ClientRectangle

    Dim oldPercentValue As Single = CSng(oldValue) / (CSng(Max) - CSng(Min))
    Dim oldNonDimLength As Integer = CInt(oldPercentValue * CSng(client.Width))

    Dim newPercentValue As Single = (CSng(newValue) / (CSng(Max) - CSng(Min)))
    Dim newNonDimLength As Integer = CInt(newPercentValue * CSng(client.Width))

```

```

        Dim lmin As Integer = Math.Min(oldNonDimLength, newNonDimLength)
        Dim lmax As Integer = Math.Max(oldNonDimLength, newNonDimLength)

        Dim invalid As Rectangle = New Rectangle( _
            client.X + lmin, _
            client.Y, _
            lmax - lmin, _
            client.Height)

        Invalidate(invalid)

        Dim oldToDisplay As String
        Dim newToDisplay As String

        If (ShowPercentage) Then
            oldToDisplay = Convert.ToString(CInt(oldPercentValue * 100F)) + "%"
            newToDisplay = Convert.ToString(CInt(newPercentValue * 100F)) + "%"
        ElseIf (ShowValue) Then
            oldToDisplay = Convert.ToString(oldValue)
            newToDisplay = Convert.ToString(newValue)
        Else
            oldToDisplay = Nothing
            newToDisplay = Nothing
        End If

        If (oldToDisplay <> Nothing And newToDisplay <> Nothing) Then
            Dim g As Graphics = CreateGraphics()
            Dim oldFontSize As SizeF = g.MeasureString(oldToDisplay, Font)
            Dim newFontSize As SizeF = g.MeasureString(newToDisplay, Font)
            Dim oldFontRectF As RectangleF = New RectangleF(New PointF(0, 0), oldFontSize)
            Dim newFontRectF As RectangleF = New RectangleF(New PointF(0, 0), newFontSize)
            oldFontRectF.X = (client.Width - oldFontRectF.Width) / 2
            oldFontRectF.Y = (client.Height - oldFontRectF.Height) / 2
            newFontRectF.X = (client.Width - newFontRectF.Width) / 2
            newFontRectF.Y = (client.Height - newFontRectF.Height) / 2

            Dim oldFontRect As Rectangle = New Rectangle(CInt(oldFontRectF.X), CInt(oldFontRectF.Y),
CInt(oldFontRectF.Width), CInt(oldFontRectF.Height))
            Dim newFontRect As Rectangle = New Rectangle(CInt(newFontRectF.X), CInt(newFontRectF.Y),
CInt(newFontRectF.Width), CInt(newFontRectF.Height))

            Invalidate(oldFontRect)
            Invalidate(newFontRect)
        End If
    End Sub

    Protected Overrides Sub OnMouseDown(ByVal e As MouseEventArgs)
        MyBase.OnMouseDown(e)
        If Not (myAllowUserEdit) Then
            Return
        End If
        Capture = True
        dragging = True
        SetDragValue(New Point(e.X, e.Y))
    End Sub

    Protected Overrides Sub OnMouseMove(ByVal e As MouseEventArgs)
        MyBase.OnMouseMove(e)
        If (Not myAllowUserEdit Or Not dragging) Then
            Return
        End If
        SetDragValue(New Point(e.X, e.Y))
    End Sub

    Protected Overrides Sub OnMouseUp(ByVal e As MouseEventArgs)
        MyBase.OnMouseUp(e)
        If (Not myAllowUserEdit Or Not dragging) Then
            Return
        End If
    End Sub

```

```

    Return
End If
Capture = False
dragging = False
Value = dragValue
OnValueChanged(EventArgs.Empty)
End Sub

Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)

 MyBase.OnPaint(e)

 If (baseBackground Is Nothing) Then

 If (myShowGradient) Then
 baseBackground = New LinearGradientBrush(New Point(0, 0), _
                                         New Point(ClientSize.Width, 0), _
                                         StartColor, _
                                         EndColor)
 ElseIf (BackgroundImage IsNot Nothing) Then
 baseBackground = New TextureBrush(BackgroundImage)
 Else
 baseBackground = New SolidBrush(BackColor)
 End If

End If

If (backgroundDim Is Nothing) Then
 backgroundDim = New SolidBrush(Color.FromArgb(DarkenBy, Color.Black))
End If

Dim toDim As Rectangle = ClientRectangle
Dim percentValue As Single = (CSng(Value) / (CSng(Max) - CSng(Min)))
Dim nonDimLength As Integer = CInt(percentValue * CSng(toDim.Width))
toDim.X = toDim.X + nonDimLength
toDim.Width = toDim.Width - nonDimLength

Dim ltext As String = Me.Text
Dim toDisplay As String
Dim textRect As RectangleF = New RectangleF(0, 0, 0, 0)

If (ShowPercentage Or ShowValue Or ltext.Length > 0) Then

 If (ShowPercentage) Then
 toDisplay = Convert.ToString(CInt(percentValue * 100F)) + "%"
 ElseIf (ShowValue) Then
 toDisplay = Convert.ToString(Value)
 Else
 toDisplay = ltext
 End If

Dim textSize As SizeF = e.Graphics.MeasureString(toDisplay, Font)
textRect.Width = textSize.Width
textRect.Height = textSize.Height
textRect.X = (ClientRectangle.Width - textRect.Width) / 2
textRect.Y = (ClientRectangle.Height - textRect.Height) / 2
End If

e.Graphics.FillRectangle(baseBackground, ClientRectangle)
e.Graphics.FillRectangle(backgroundDim, toDim)
e.Graphics.Flush()
If (toDisplay <> Nothing And toDisplay.Length > 0) Then
 e.Graphics.DrawString(toDisplay, Font, New SolidBrush(ForeColor), textRect)
End If
End Sub

Protected Overrides Sub OnTextChanged(ByVal E As EventArgs)
 MyBase.OnTextChanged(E)
 Invalidate()

```

```

End Sub

Protected Overrides Sub OnBackColorChanged(ByVal E As EventArgs)
    MyBase.OnBackColorChanged(E)
    If (baseBackground IsNot Nothing) And Not ShowGradient Then
        baseBackground.Dispose()
        baseBackground = Nothing
    End If
End Sub

Protected Overrides Sub OnBackgroundImageChanged(ByVal E As EventArgs)
    MyBase.OnTextChanged(E)
    If (baseBackground IsNot Nothing) And Not ShowGradient Then
        baseBackground.Dispose()
        baseBackground = Nothing
    End If
End Sub

Protected Overrides Sub OnResize(ByVal e As EventArgs)
    MyBase.OnResize(e)
    If (baseBackground IsNot Nothing) Then
        baseBackground.Dispose()
        baseBackground = Nothing
    End If
End Sub

Protected Overridable Sub OnValueChanged(ByVal e As EventArgs)
    RaiseEvent ValueChanged(Me, e)
End Sub

Private Sub SetDragValue(ByVal mouseLocation As Point)

    Dim client As Rectangle = ClientRectangle

    If (client.Contains(mouseLocation)) Then

        Dim percentage As Single = CSng(mouseLocation.X) / CSng(ClientRectangle.Width)
        Dim newDragValue As Integer = CInt(percentage * CSng(Max - Min))

        If (newDragValue <> dragValue) Then
            Dim old As Integer = dragValue
            dragValue = newDragValue
            OptimizedInvalidate(old, dragValue)
        End If

    Else

        If (client.Y <= mouseLocation.Y And mouseLocation.Y <= client.Y + client.Height) Then
            If (mouseLocation.X <= client.X And mouseLocation.X > client.X - LeftRightBorder) Then
                Dim newDragValue As Integer = Min
                If (newDragValue <> dragValue) Then
                    Dim old As Integer = dragValue
                    dragValue = newDragValue
                    OptimizedInvalidate(old, dragValue)
                End If

                ElseIf (mouseLocation.X >= client.X + client.Width And mouseLocation.X < client.X + client.Width + LeftRightBorder) Then
                    Dim newDragValue As Integer = Max
                    If (newDragValue <> dragValue) Then
                        Dim old As Integer = dragValue
                        dragValue = newDragValue
                        OptimizedInvalidate(old, dragValue)
                    End If
                End If

            Else
                If (dragValue <> Value) Then
                    Dim old As Integer = dragValue

```

```

        dragValue = Value
        OptimizedInvalidate(old, dragValue)
    End If

    End If

End Sub

End Class

End Namespace

```

```

namespace Microsoft.Samples.WinForms.Cs.FlashTrackBar {
    using System;
    using System.ComponentModel;
    using System.ComponentModel.Design;
    using System.Diagnostics;
    using System.Drawing;
    using System.Drawing.Drawing2D;
    using System.Drawing.Design;
    using System.Windows.Forms;
    using System.Windows.Forms.ComponentModel;
    using System.Windows.Forms.Design;

    public class FlashTrackBarDarkenByEditor : FlashTrackBarValueEditor {
        protected override void SetEditorProps(FlashTrackBar editingInstance, FlashTrackBar editor) {
            base.SetEditorProps(editingInstance, editor);
            editor.Min = 0;
            editor.Max = byte.MaxValue;
        }
    }
}

```

```

Imports System
Imports System.ComponentModel
Imports System.ComponentModel.Design
Imports System.Drawing
Imports System.Drawing.Drawing2D
Imports System.Drawing.Design
Imports System.Windows.Forms
Imports System.Diagnostics

Namespace Microsoft.Samples.WinForms.VB.FlashTrackBar

    Public Class FlashTrackBarDarkenByEditor
        Inherits FlashTrackBarValueEditor

        Overrides Protected Sub SetEditorProps(editingInstance As FlashTrackBar, editor As FlashTrackBar)
            MyBase.SetEditorProps(editingInstance, editor)
            editor.Min = 0
            editor.Max = System.Byte.MaxValue
        End Sub

    End Class

End Namespace

```

```

namespace Microsoft.Samples.WinForms.Cs.FlashTrackBar {
    using System;
    using System.ComponentModel;
    using System.ComponentModel.Design;

```

```

using System.Diagnostics;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Drawing.Design;
using System.Windows.Forms;
using System.Windows.Forms.ComponentModel;
using System.Windows.Forms.Design;

[System.Security.Permissions.PermissionSet(System.Security.Permissions.SecurityAction.Demand, Name =
"FullTrust")]
public class FlashTrackBarValueEditor : System.Drawing.Design.UITypeEditor {

    private IWindowsFormsEditorService edSvc = null;

    protected virtual void SetEditorProps(FlashTrackBar editingInstance, FlashTrackBar editor) {
        editor.ShowValue = true;
        editor.StartColor = Color.Navy;
        editor.EndColor = Color.White;
        editor.ForeColor = Color.White;
        editor.Min = editingInstance.Min;
        editor.Max = editingInstance.Max;
    }

    public override object EditValue(ITypeDescriptorContext context, IServiceProvider provider, object
value) {

        if (context != null
            && context.Instance != null
            && provider != null) {

            edSvc = (IWindowsFormsEditorService)provider.GetService(typeof(IWindowsFormsEditorService));

            if (edSvc != null) {
                FlashTrackBar trackBar = new FlashTrackBar();
                trackBar.ValueChanged += new EventHandler(this.ValueChanged);
                SetEditorProps((FlashTrackBar)context.Instance, trackBar);
                bool asInt = true;
                if (value is int) {
                    trackBar.Value = (int)value;
                }
                else if (value is byte) {
                    asInt = false;
                    trackBar.Value = (byte)value;
                }
                edSvc.DropDownControl(trackBar);
                if (asInt) {
                    value = trackBar.Value;
                }
                else {
                    value = (byte)trackBar.Value;
                }
            }
        }

        return value;
    }

    public override UITypeEditorEditStyle GetEditStyle(ITypeDescriptorContext context) {
        if (context != null && context.Instance != null) {
            return UITypeEditorEditStyle.DropDown;
        }
        return base.GetEditStyle(context);
    }

    private void ValueChanged(object sender, EventArgs e) {
        if (edSvc != null) {
            edSvc.CloseDropDown();
        }
    }
}

```

```
    }  
}
```

```
Imports System  
Imports System.ComponentModel  
Imports System.ComponentModel.Design  
Imports System.Drawing  
Imports System.Drawing.Drawing2D  
Imports System.Drawing.Design  
Imports System.Windows.Forms  
Imports System.Diagnostics  
Imports System.Windows.Forms.ComponentModel  
Imports System.Windows.Forms.Design

Namespace Microsoft.Samples.WinForms.VB.FlashTrackBar

    <System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.Demand,  
Name:="FullTrust")> _  
    Public Class FlashTrackBarValueEditor  
        Inherits UITypeEditor

        Private edSvc As IWindowsFormsEditorService

        Overridable Protected Sub SetEditorProps(editingInstance As FlashTrackBar, editor As FlashTrackBar)  
            editor.ShowValue = true  
            editor.StartColor = Color.Navy  
            editor.EndColor = Color.White  
            editor.ForeColor = Color.White  
            editor.Min = editingInstance.Min  
            editor.Max = editingInstance.Max  
        End Sub

        Overrides Overloads Public Function EditValue(context As ITypeDescriptorContext, provider As  
IServiceProvider, value As Object) As Object

            If ((context IsNot Nothing) And (context.Instance IsNot Nothing) And (provider IsNot Nothing))  
Then

                edSvc = CType(provider.GetService(GetType(IWindowsFormsEditorService)),  
IWindowsFormsEditorService)

                If (edSvc IsNot Nothing) Then

                    Dim trackBar As FlashTrackBar = New FlashTrackBar()  
                    AddHandler trackBar.ValueChanged, AddressOf Me.ValueChanged  
                    SetEditorProps(CType(context.Instance, FlashTrackBar), trackBar)

                    Dim asInt As Boolean = True

                    If (TypeOf value Is Integer) Then  
                        trackBar.Value = CInt(value)
                    ElseIf (TypeOf value Is System.Byte) Then  
                        asInt = False
                        trackBar.Value = CType(value, Byte)
                    End If

                    edSvc.DropDownControl(trackBar)

                    If (asInt) Then  
                        value = trackBar.Value
                    Else
                        value = CType(trackBar.Value, Byte)
                    End If

                End If
            End If
        End Function
    End Class
End Namespace
```

```
End If

Return value

End Function

Overrides Overloads Public Function GetEditStyle(context As ITypeDescriptorContext) As
UITypeEditorEditStyle
    If ((context IsNot Nothing) And (context.Instance IsNot Nothing)) Then
        Return UITypeEditorEditStyle.DropDown
    End If
    Return MyBase.GetEditStyle(context)
End Function

private Sub ValueChanged(sender As object, e As EventArgs)
    If (edSvc IsNot Nothing) Then
        edSvc.CloseDropDown()
    End If
End Sub

End Class

End Namespace
```

```
namespace Microsoft.Samples.WinForms.Cs.HostApp {
    using System;
    using System.ComponentModel;
    using System.Drawing;
    using System.Windows.Forms;
    using Microsoft.Samples.WinForms.Cs.FlashTrackBar;

    public class HostApp : System.Windows.Forms.Form {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Container components;
        protected internal Microsoft.Samples.WinForms.Cs.FlashTrackBar.FlashTrackBar flashTrackBar1;

        public HostApp() {
            //
            // Required for Windows Form Designer support
            //
            InitializeComponent();
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose(bool disposing)
        {
            if (disposing) {
                if (components != null) {
                    components.Dispose();
                }
            }
            base.Dispose(disposing);
        }

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent() {
            this.components = new System.ComponentModel.Container ();
            this.flashTrackBar1 = new Microsoft.Samples.WinForms.Cs.FlashTrackBar.FlashTrackBar ();
            this.Text = "Control Example";
            this.ClientSize = new System.Drawing.Size (600, 450);
            flashTrackBar1.BackColor = System.Drawing.Color.Black;
            flashTrackBar1.Dock = System.Windows.Forms.DockStyle.Fill;
            flashTrackBar1.TabIndex = 0;
            flashTrackBar1.ForeColor = System.Drawing.Color.White;
            flashTrackBar1.Text = "Drag the Mouse and say Wow!";
            flashTrackBar1.Value = 73;
            flashTrackBar1.Size = new System.Drawing.Size (600, 450);
            this.Controls.Add (this.flashTrackBar1);
        }

        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        public static void Main(string[] args) {
            Application.Run(new HostApp());
        }
    }
}
```

```

Imports System
Imports System.ComponentModel
Imports System.Drawing
Imports System.Windows.Forms
Imports Microsoft.Samples.WinForms.VB.FlashTrackBar

Namespace Microsoft.Samples.WinForms.VB.HostApp

    Public Class HostApp
        Inherits System.Windows.Forms.Form

        Public Sub New()
            MyBase.New()

            HostApp = Me

            'This call is required by the Windows Forms Designer.
            InitializeComponent()
        End Sub

        'Form overrides dispose to clean up the component list.
        Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
            If disposing Then
                If (components IsNot Nothing) Then
                    components.Dispose()
                End If
            End If
            MyBase.Dispose(disposing)
        End Sub

#Region " Windows Form Designer generated code "

        'Required by the Windows Form Designer
        Private components As System.ComponentModel.Container
        Private WithEvents flashTrackBar1 As Microsoft.Samples.WinForms.VB.FlashTrackBar.FlashTrackBar

        Private WithEvents HostApp As System.Windows.Forms.Form

        'NOTE: The following procedure is required by the Windows Form Designer
        'It can be modified using the Windows Form Designer.
        'Do not modify it using the code editor.
        Private Sub InitializeComponent()
            Me.components = New System.ComponentModel.Container()
            Me.flashTrackBar1 = New Microsoft.Samples.WinForms.VB.FlashTrackBar.FlashTrackBar()

            flashTrackBar1.Dock = System.Windows.Forms.DockStyle.Fill
            flashTrackBar1.ForeColor = System.Drawing.Color.White
            flashTrackBar1.BackColor = System.Drawing.Color.Black
            flashTrackBar1.Size = New System.Drawing.Size(600, 450)
            flashTrackBar1.TabIndex = 0
            flashTrackBar1.Value = 73
            flashTrackBar1.Text = "Drag the Mouse and say Wow!"

            Me.Text = "Control Example"
            Me.ClientSize = New System.Drawing.Size(528, 325)

            Me.Controls.Add(flashTrackBar1)
        End Sub

#End Region

        'The main entry point for the application
        <STAThread()> Shared Sub Main()
    End Class
End Namespace

```

```
System.Windows.Forms.Application.Run(New HostApp())
End Sub

End Class

End Namespace
```

See Also

[Extending Design-Time Support](#)

[Windows Forms Control Development Basics](#)

Developing a Composite Windows Forms Control

5/4/2018 • 1 min to read • [Edit Online](#)

You can develop a composite Windows Forms control by combining other Windows Forms controls. Composite controls that derive from [UserControl](#) are called user controls. The base class, [UserControl](#), provides keyboard routing for the child controls, thus ensuring that child controls can receive focus. For an example of a user control, see the [UserControl](#) sample in [How to: Apply Attributes in Windows Forms Controls](#).

The Windows Forms designer in Microsoft Visual Studio 2005 provides rich design-time support for authoring user controls.

- [How to: Display a Control in the Choose Toolbox Items Dialog Box](#)
- [Walkthrough: Serializing Collections of Standard Types with the DesignerSerializationVisibilityAttribute](#)
- [Walkthrough: Inheriting from a Windows Forms Control with Visual C#](#)
- [How to: Provide a Toolbox Bitmap for a Control](#)
- [How to: Inherit from Existing Windows Forms Controls](#)
- [Walkthrough: Debugging Custom Windows Forms Controls at Design Time](#)
- [How to: Inherit from the Control Class](#)
- [How to: Test the Run-Time Behavior of a UserControl](#)
- [How to: Align a Control to the Edges of Forms at Design Time](#)
- [How to: Inherit from the UserControl Class](#)
- [How to: Author Controls for Windows Forms](#)
- [How to: Author Composite Controls](#)
- [Walkthrough: Authoring a Composite Control with Visual Basic](#)
- [Walkthrough: Authoring a Composite Control with Visual C#](#)
- [Walkthrough: Inheriting from a Windows Forms Control with Visual Basic](#)
- [How to: Create a Windows Forms Control That Takes Advantage of Design-Time Features](#)
- [How to: Create a Windows Forms Control That Takes Advantage of Design-Time Features](#)

See Also

[How to: Apply Attributes in Windows Forms Controls](#)

[Developing Custom Windows Forms Controls with the .NET Framework](#)

[Varieties of Custom Controls](#)

Properties in Windows Forms Controls

5/4/2018 • 1 min to read • [Edit Online](#)

A Windows Forms control inherits many properties from the base class [System.Windows.Forms.Control](#). These include properties such as [Font](#), [ForeColor](#), [BackColor](#), [Bounds](#), [ClientRectangle](#), [DisplayRectangle](#), [Enabled](#), [Focused](#), [Height](#), [Width](#), [Visible](#), [AutoSize](#), and many others. For details about inherited properties, see [System.Windows.Forms.Control](#).

You can override inherited properties in your control as well as define new properties.

In This Section

[Defining a Property](#)

Shows how to implement a property for a custom control or component and shows how to integrate the property into the design environment.

[Defining Default Values with the ShouldSerialize and Reset Methods](#)

Shows how to define default property values for a custom control or component.

[Property-Changed Events](#)

Describes how to enable property-change notifications when a property value changes.

[How to: Expose Properties of Constituent Controls](#)

Shows how to expose properties of constituent controls in a custom composite control.

[Method Implementation in Custom Controls](#)

Describes how to implement methods in custom controls and components.

Reference

[UserControl](#)

Documents the base class for implementing composite controls.

[TypeConverterAttribute](#)

Documents the attribute that specifies the [TypeConverter](#) to use for a custom property type.

[EditorAttribute](#)

Documents the attribute that specifies the [UITypeEditor](#) to use for a custom property.

Related Sections

[Attributes in Windows Forms Controls](#)

Describes the attributes you can apply to properties or other members of your custom controls and components.

[Design-Time Attributes for Components](#)

Lists metadata attributes to apply to components and controls so that they are displayed correctly at design time in visual designers.

[Extending Design-Time Support](#)

Describes how to implement classes such as editors and designers that provide design-time support.

Defining a Property in Windows Forms Controls

5/4/2018 • 3 min to read • [Edit Online](#)

For an overview of properties, see [Properties Overview](#). There are a few important considerations when defining a property:

- You must apply attributes to the properties you define. Attributes specify how the designer should display a property. For details, see [Design-Time Attributes for Components](#).
- If changing the property affects the visual display of the control, call the `Invalidate` method (that your control inherits from `Control`) from the `set` accessor. `Invalidate` in turn calls the `OnPaint` method, which redraws the control. Multiple calls to `Invalidate` result in a single call to `OnPaint` for efficiency.
- The .NET Framework class library provides type converters for common data types such as integers, decimal numbers, Boolean values, and others. The purpose of a type converter is generally to provide string-to-value conversion (from string data to other data types). Common data types are associated with default type converters that convert values into strings and strings into the appropriate data types. If you define a property that is a custom (that is, nonstandard) data type, you will have to apply an attribute that specifies the type converter to associate with that property. You can also use an attribute to associate a custom UI type editor with a property. A UI type editor provides a user interface for editing a property or data type. A color picker is an example of a UI type editor. Examples of attributes are given at the end of this topic.

NOTE

If a type converter or a UI type editor is not available for your custom property, you can implement one as described in [Extending Design-Time Support](#).

The following code fragment defines a custom property named `EndColor` for the custom control `FlashTrackBar`.

```

Public Class FlashTrackBar
    Inherits Control
    ...
    ' Private data member that backs the EndColor property.
    Private _endColor As Color = Color.LimeGreen

    ' The Category attribute tells the designer to display
    ' it in the Flash grouping.
    ' The Description attribute provides a description of
    ' the property.
    <Category("Flash"), _
    Description("The ending color of the bar.")> _
    Public Property EndColor() As Color
        ' The public property EndColor accesses _endColor.
        Get
            Return _endColor
        End Get
        Set
            _endColor = value
            If Not (baseBackground Is Nothing) And showGradient Then
                baseBackground.Dispose()
                baseBackground = Nothing
            End If
            ' The Invalidate method calls the OnPaint method, which redraws
            ' the control.
            Invalidate()
        End Set
    End Property
    ...
End Class

```

```

public class FlashTrackBar : Control {
    ...
    // Private data member that backs the EndColor property.
    private Color endColor = Color.LimeGreen;
    // The Category attribute tells the designer to display
    // it in the Flash grouping.
    // The Description attribute provides a description of
    // the property.
    [
    Category("Flash"),
    Description("The ending color of the bar.")
    ]
    // The public property EndColor accesses endColor.
    public Color EndColor {
        get {
            return endColor;
        }
        set {
            endColor = value;
            if (baseBackground != null && showGradient) {
                baseBackground.Dispose();
                baseBackground = null;
            }
            // The Invalidate method calls the OnPaint method, which redraws
            // the control.
            Invalidate();
        }
    }
    ...
}

```

The following code fragment associates a type converter and a UI type editor with the property `Value`. In this case `Value` is an integer and has a default type converter, but the [TypeConverterAttribute](#) attribute applies a custom

type converter (`FlashTrackBarValueConverter`) that enables the designer to display it as a percentage. The UI type editor, `FlashTrackBarValueEditor`, allows the percentage to be displayed visually. This example also shows that the type converter or editor specified by the `TypeConverterAttribute` or `EditorAttribute` attribute overrides the default converter.

```
<Category("Flash"), _  
TypeConverter(GetType(FastMember.FlashTrackBarValueConverter)), _  
Editor(GetType(FastMember.FlashTrackBarValueEditor)), _  
GetType(UITypeEditor)), _  
Description("The current value of the track bar. You can enter an actual value or a percentage.")> _  
Public ReadOnly Property Value() As Integer  
...  
End Property
```

```
[  
Category("Flash"),  
TypeConverter(typeof(FastMember.FlashTrackBarValueConverter)),  
Editor(typeof(FastMember.FlashTrackBarValueEditor), typeof(UITypeEditor)),  
Description("The current value of the track bar. You can enter an actual value or a percentage.")  
]  
public int Value {  
...  
}
```

See Also

[Properties in Windows Forms Controls](#)

[Defining Default Values with the ShouldSerialize and Reset Methods](#)

[Property-Changed Events](#)

[Attributes in Windows Forms Controls](#)

Defining Default Values with the ShouldSerialize and Reset Methods

5/4/2018 • 3 min to read • [Edit Online](#)

`ShouldSerialize` and `Reset` are optional methods that you can provide for a property, if the property does not have a simple default value. If the property has a simple default value, you should apply the `DefaultValueAttribute` and supply the default value to the attribute class constructor instead. Either of these mechanisms enables the following features in the designer:

- The property provides visual indication in the property browser if it has been modified from its default value.
- The user can right-click on the property and choose **Reset** to restore the property to its default value.
- The designer generates more efficient code.

NOTE

Either apply the `DefaultValueAttribute` or provide `Reset PropertyName` and `ShouldSerialize PropertyName` methods. Do not use both.

The `Reset PropertyName` method sets a property to its default value, as shown in the following code fragment.

```
Public Sub ResetMyFont()
    MyFont = Nothing
End Sub
```

```
public void ResetMyFont() {
    MyFont = null;
}
```

NOTE

If a property does not have a `Reset` method, is not marked with a `DefaultValueAttribute`, and does not have a default value supplied in its declaration, the `Reset` option for that property is disabled in the shortcut menu of the **Properties** window of the Windows Forms Designer in Visual Studio.

Designers such as Visual Studio use the `ShouldSerialize PropertyName` method to check whether a property has changed from its default value and write code into the form only if a property is changed, thus allowing for more efficient code generation. For example:

```
'Returns true if the font has changed; otherwise, returns false.
' The designer writes code to the form only if true is returned.
Public Function ShouldSerializeMyFont() As Boolean
    Return Not (theFont Is Nothing)
End Function
```

```
// Returns true if the font has changed; otherwise, returns false.  
// The designer writes code to the form only if true is returned.  
public bool ShouldSerializeMyFont() {  
    return thefont != null;  
}
```

A complete code example follows.

```
Option Explicit  
Option Strict  
  
Imports System  
Imports System.Windows.Forms  
Imports System.Drawing  
  
Public Class MyControl  
    Inherits Control  
  
    ' Declare an instance of the Font class  
    ' and set its default value to Nothing.  
    Private thefont As Font = Nothing  
  
    ' The MyFont property.  
    Public Property MyFont() As Font  
        ' Note that the Font property never  
        ' returns null.  
        Get  
            If Not (thefont Is Nothing) Then  
                Return thefont  
            End If  
            If Not (Parent Is Nothing) Then  
                Return Parent.Font  
            End If  
            Return Control.DefaultFont  
        End Get  
        Set  
            thefont = value  
        End Set  
    End Property  
  
    Public Function ShouldSerializeMyFont() As Boolean  
        Return Not (thefont Is Nothing)  
    End Function  
  
    Public Sub ResetMyFont()  
        MyFont = Nothing  
    End Sub  
End Class
```

```

using System;
using System.Windows.Forms;
using System.Drawing;

public class MyControl : Control {
    // Declare an instance of the Font class
    // and set its default value to null.
    private Font thefont = null;

    // The MyFont property.
    public Font MyFont {
        // Note that the MyFont property never
        // returns null.
        get {
            if (thefont != null) return thefont;
            if (Parent != null) return Parent.Font;
            return Control.DefaultFont;
        }
        set {
            thefont = value;
        }
    }

    public bool ShouldSerializeMyFont() {
        return thefont != null;
    }

    public void ResetMyFont() {
        MyFont = null;
    }
}

```

In this case, even when the value of the private variable accessed by the `MyFont` property is `null`, the property browser does not display `null`; instead, it displays the `Font` property of the parent, if it is not `null`, or the default `Font` value defined in `Control`. Thus the default value for `MyFont` cannot be simply set, and a `DefaultValueAttribute` cannot be applied to this property. Instead, the `ShouldSerialize` and `Reset` methods must be implemented for the `MyFont` property.

See Also

[Properties in Windows Forms Controls](#)

[Defining a Property](#)

[Property-Changed Events](#)

Property-Changed Events

5/4/2018 • 1 min to read • [Edit Online](#)

If you want your control to send notifications when a property named *PropertyName* changes, define an event named *PropertyName* `Changed` and a method named `OnPropertyName Changed` that raises the event. The naming convention in Windows Forms is to append the word *Changed* to the name of the property. The associated event delegate type for property-changed events is [EventHandler](#), and the event data type is [EventArgs](#). The base class [Control](#) defines many property-changed events, such as [BackColorChanged](#), [BackgroundImageChanged](#), [FontChanged](#), [LocationChanged](#), and others. For background information about events, see [Events](#) and [Events in Windows Forms Controls](#).

Property-changed events are useful because they allow consumers of a control to attach event handlers that respond to the change. If your control needs to respond to a property-changed event that it raises, override the corresponding `OnPropertyName Changed` method instead of attaching a delegate to the event. A control typically responds to a property-changed event by updating other properties or by redrawing some or all of its drawing surface.

The following example shows how the [FlashTrackBar](#) custom control responds to some of the property-changed events that it inherits from [Control](#). For the complete sample, see [How to: Create a Windows Forms Control That Shows Progress](#).

```
protected override void OnTextChanged(EventArgs e) {
    base.OnTextChanged(e);
    Invalidate();
}

protected override void OnBackColorChanged(EventArgs e) {
    base.OnBackColorChanged(e);
    if ((baseBackground != null) && (!showGradient)) {
        baseBackground.Dispose();
        baseBackground = null;
    }
}
```

```
Protected Overrides Sub OnTextChanged(ByVal E As EventArgs)
    MyBase.OnTextChanged(E)
    Invalidate()
End Sub

Protected Overrides Sub OnBackColorChanged(ByVal E As EventArgs)
    MyBase.OnBackColorChanged(E)
    If (baseBackground IsNot Nothing) And Not ShowGradient Then
        baseBackground.Dispose()
        baseBackground = Nothing
    End If
End Sub
```

See Also

[Events](#)

[Events in Windows Forms Controls](#)

[Properties in Windows Forms Controls](#)

How to: Expose Properties of Constituent Controls

5/4/2018 • 1 min to read • [Edit Online](#)

The controls that make up a composite control are called *constituent controls*. These controls are normally declared private, and thus cannot be accessed by the developer. If you want to make properties of these controls available to future users, you must expose them to the user. A property of a constituent control is exposed by creating a property in the user control, and using the `get` and `set` accessors of that property to effect the change in the private property of the constituent control.

Consider a hypothetical user control with a constituent button named `MyButton`. In this example, when the user requests the `ConstituentButtonBackColor` property, the value stored in the `BackColor` property of `MyButton` is delivered. When the user assigns a value to this property, that value is automatically passed to the `BackColor` property of `MyButton` and the `set` code will execute, changing the color of `MyButton`.

The following example shows how to expose the `BackColor` property of the constituent button:

```
Public Property ButtonColor() as System.Drawing.Color
    Get
        Return MyButton.BackColor
    End Get
    Set(Value as System.Drawing.Color)
        MyButton.BackColor = Value
    End Set
End Property
```

```
public Color ButtonColor
{
    get
    {
        return(myButton.BackColor);
    }
    set
    {
        myButton.BackColor = value;
    }
}
```

To expose a property of a constituent control

1. Create a public property for your user control.
2. In the `get` section of the property, write code that retrieves the value of the property you want to expose.
3. In the `set` section of the property, write code that passes the value of the property to the exposed property of the constituent control.

See Also

[UserControl](#)

[Properties in Windows Forms Controls](#)

[Varieties of Custom Controls](#)

Method Implementation in Custom Controls

5/4/2018 • 2 min to read • [Edit Online](#)

A method is implemented in a control in the same manner a method would be implemented in any other component.

In Visual Basic, if a method is required to return a value, it is implemented as a `Public Function`. If no value is returned, it is implemented as a `Public Sub`. Methods are declared using the following syntax:

```
Public Function ConvertMatterToEnergy(Matter as Integer) As Integer
    ' Conversion code goes here.
End Function
```

Because functions return a value, they must specify a return type, such as integer, string, object, and so on. The arguments `Function` or `Sub` procedures take, if any, must also be specified.

C# makes no distinction between functions and procedures, as Visual Basic does. A method either returns a value or returns `void`. The syntax for declaring a C# public method is:

```
public int ConvertMatterToEnergy(int matter)
{
    // Conversion code goes here.
}
```

When you declare a method, declare all of its arguments as explicit data types whenever possible. Arguments that take object references should be declared as specific class types — for example, `As Widget` instead of `As Object`. In Visual Basic, the default setting `Option Strict` automatically enforces this rule.

Typed arguments allow many developer errors to be caught by the compiler, rather than at run time. The compiler always catches errors, whereas run-time testing is only as good as the test suite.

Overloaded Methods

If you want to allow users of your control to supply different combinations of parameters to a method, provide multiple overloads of the method, using explicit data types. Avoid creating parameters declared `As Object` that can contain any data type, as this can lead to errors that might not be caught in testing.

NOTE

The universal data type in the common language runtime is `Object` rather than `Variant`. `Variant` has been removed from the language.

For example, the `Spin` method of a hypothetical `Widget` control might allow either direct specification of spin direction and speed, or specification of another `Widget` object from which angular momentum is to be absorbed:

```
Overloads Public Sub Spin( _
    ByVal SpinDirection As SpinDirectionsEnum, _
    ByVal RevolutionsPerSecond As Double)
    ' Implementation code here.
End Sub
Overloads Public Sub Spin(ByVal Driver As Widget) _
    ' Implementation code here.
End Sub
```

```
public void Spin(SpinDirectionsEnum spinDirection, double revolutionsPerSecond)
{
    // Implementation code here.
}

public void Spin(Widget driver)
{
    // Implementation code here.
}
```

See Also

[Events](#)

[Properties in Windows Forms Controls](#)

Events in Windows Forms Controls

5/4/2018 • 1 min to read • [Edit Online](#)

A Windows Forms control inherits more than sixty events from [System.Windows.Forms.Control](#). These include the [Paint](#) event, which causes a control to be drawn, events related to displaying a window, such as the [Resize](#) and [Layout](#) events, and low-level mouse and keyboard events. Some low-level events are synthesized by [Control](#) into semantic events such as [Click](#) and [DoubleClick](#). For details about inherited events, see [Control](#).

If your custom control needs to override inherited event functionality, override the inherited `on EventName` method instead of attaching a delegate. If you are not familiar with the event model in the .NET Framework, see [Raising Events from a Component](#).

See Also

[Overriding the OnPaint Method](#)

[Handling User Input](#)

[Defining an Event](#)

[Events](#)

Overriding the OnPaint Method

5/4/2018 • 1 min to read • [Edit Online](#)

The basic steps for overriding any event defined in the .NET Framework are identical and are summarized in the following list.

To override an inherited event

1. Override the protected `on EventName` method.
2. Call the `on EventName` method of the base class from the overridden `on EventName` method, so that registered delegates receive the event.

The `Paint` event is discussed in detail here because every Windows Forms control must override the `Paint` event that it inherits from `Control`. The base `Control` class does not know how a derived control needs to be drawn and does not provide any painting logic in the `OnPaint` method. The `OnPaint` method of `Control` simply dispatches the `Paint` event to registered event receivers.

If you worked through the sample in [How to: Develop a Simple Windows Forms Control](#), you have seen an example of overriding the `OnPaint` method. The following code fragment is taken from that sample.

```
Public Class FirstControl
    Inherits Control

    Public Sub New()
        MyBase.OnPaint += New PaintEventHandler(AddressOf OnPaint)
    End Sub

    Protected Overrides Sub OnPaint(e As PaintEventArgs)
        ' Call the OnPaint method of the base class.
        MyBase.OnPaint(e)
        ' Call methods of the System.Drawing.Graphics object.
        e.Graphics.DrawString(Text, Font, New SolidBrush(ForeColor), RectangleF.op_Implicit(ClientRectangle))
    End Sub
End Class
```

```
public class FirstControl : Control{
    public FirstControl() {}
    protected override void OnPaint(PaintEventArgs e) {
        // Call the OnPaint method of the base class.
        base.OnPaint(e);
        // Call methods of the System.Drawing.Graphics object.
        e.Graphics.DrawString(Text, Font, new SolidBrush(ForeColor), ClientRectangle);
    }
}
```

The `PaintEventArgs` class contains data for the `Paint` event. It has two properties, as shown in the following code.

```
Public Class PaintEventArgs
    Inherits EventArgs
    ...
    Public ReadOnly Property ClipRectangle() As System.Drawing.Rectangle
        ...
    End Property

    Public ReadOnly Property Graphics() As System.Drawing.Graphics
        ...
    End Property
    ...
End Class
```

```
public class PaintEventArgs : EventArgs {
    ...
    public System.Drawing.Rectangle ClipRectangle {}
    public System.Drawing.Graphics Graphics {}
    ...
}
```

[ClipRectangle](#) is the rectangle to be painted, and the [Graphics](#) property refers to a [Graphics](#) object. The classes in the [System.Drawing](#) namespace are managed classes that provide access to the functionality of GDI+, the new Windows graphics library. The [Graphics](#) object has methods to draw points, strings, lines, arcs, ellipses, and many other shapes.

A control invokes its [OnPaint](#) method whenever it needs to change its visual display. This method in turn raises the [Paint](#) event.

See Also

[Events](#)

[Rendering a Windows Forms Control](#)

[Defining an Event](#)

Handling User Input

5/4/2018 • 2 min to read • [Edit Online](#)

This topic describes the main keyboard and mouse events provided by `System.Windows.Forms.Control`. When handling an event, control authors should override the protected `on EventName` method rather than attaching a delegate to the event. For a review of events, see [Raising Events from a Component](#).

NOTE

If there is no data associated with an event, an instance of the base class `EventArgs` is passed as an argument to the `on EventName` method.

Keyboard Events

The common keyboard events that your control can handle are `KeyDown`, `KeyPress`, and `KeyUp`.

EVENT NAME	METHOD TO OVERRIDE	DESCRIPTION OF EVENT
<code>KeyDown</code>	<code>void OnKeyDown(KeyEventEventArgs)</code>	Raised only when a key is initially pressed.
<code>KeyPress</code>	<code>void OnKeyPress</code> <code>(KeyPressEventArgs)</code>	Raised every time a key is pressed. If a key is held down, a <code>KeyPress</code> event is raised at the repeat rate defined by the operating system.
<code>KeyUp</code>	<code>void OnKeyUp(KeyEventEventArgs)</code>	Raised when a key is released.

NOTE

Handling keyboard input is considerably more complex than overriding the events in the preceding table and is beyond the scope of this topic. For more information, see [User Input in Windows Forms](#).

Mouse Events

The mouse events that your control can handle are `MouseDown`, `MouseEnter`, `MouseHover`, `MouseLeave`, `MouseMove`, and `MouseUp`.

EVENT NAME	METHOD TO OVERRIDE	DESCRIPTION OF EVENT
<code>MouseDown</code>	<code>void OnMouseDown(MouseEventEventArgs)</code>	Raised when the mouse button is pressed while the pointer is over the control.
<code>MouseEnter</code>	<code>void OnMouseEnter(EventArgs)</code>	Raised when the pointer first enters the region of the control.
<code>MouseHover</code>	<code>void OnMouseHover(EventArgs)</code>	Raised when the pointer hovers over the control.

EVENT NAME	METHOD TO OVERRIDE	DESCRIPTION OF EVENT
MouseLeave	void OnMouseLeave(EventArgs)	Raised when the pointer leaves the region of the control.
MouseMove	void OnMouseMove(MouseEventArgs)	Raised when the pointer moves in the region of the control.
MouseUp	void OnMouseUp(MouseEventArgs)	Raised when the mouse button is released while the pointer is over the control or the pointer leaves the region of the control.

The following code fragment shows an example of overriding the [MouseDown](#) event.

```
protected override void OnMouseDown(MouseEventArgs e) {
    base.OnMouseDown(e);
    if (!allowUserEdit) {
        return;
    }
    Capture = true;
    dragging = true;
    SetDragValue(new Point(e.X, e.Y));
}
```

```
Protected Overrides Sub OnMouseDown(ByVal e As MouseEventArgs)
    MyBase.OnMouseDown(e)
    If Not (myAllowUserEdit) Then
        Return
    End If
    Capture = True
    dragging = True
    SetDragValue(New Point(e.X, e.Y))
End Sub
```

The following code fragment shows an example of overriding the [MouseMove](#) event.

```
protected override void OnMouseMove(MouseEventArgs e) {
    base.OnMouseMove(e);
    if (!allowUserEdit || !dragging) {
        return;
    }
    SetDragValue(new Point(e.X, e.Y));
}
```

```
Protected Overrides Sub OnMouseMove(ByVal e As MouseEventArgs)
    MyBase.OnMouseMove(e)
    If (Not myAllowUserEdit Or Not dragging) Then
        Return
    End If
    SetDragValue(New Point(e.X, e.Y))
End Sub
```

The following code fragment shows an example of overriding the [MouseUp](#) event.

```
protected override void OnMouseUp(MouseEventArgs e) {
    base.OnMouseUp(e);
    if (!allowUserEdit || !dragging) {
        return;
    }
    Capture = false;
    dragging = false;
    value = dragValue;
    OnValueChanged(EventArgs.Empty);
}
```

```
Protected Overrides Sub OnMouseUp(ByVal e As MouseEventArgs)
    MyBase.OnMouseUp(e)
    If (Not myAllowUserEdit Or Not dragging) Then
        Return
    End If
    Capture = False
    dragging = False
    Value = dragValue
    OnValueChanged(EventArgs.Empty)
End Sub
```

For the complete source code for the `FlashTrackBar` sample, see [How to: Create a Windows Forms Control That Shows Progress](#).

See Also

[Events in Windows Forms Controls](#)

[Defining an Event](#)

[Events](#)

[User Input in Windows Forms](#)

Defining an Event in Windows Forms Controls

5/4/2018 • 1 min to read • [Edit Online](#)

For details about defining custom events, see [Events](#). If you define an event that does not have any associated data, use the base type for event data, [EventArgs](#), and use [EventHandler](#) as the event delegate. All that remains to do is to define an event member and a protected `OnEventName` method that raises the event.

The following code fragment shows how the [FlashTrackBar](#) custom control defines a custom event, `ValueChanged`.

For the complete code for the [FlashTrackBar](#) sample, see the [How to: Create a Windows Forms Control That Shows Progress](#).

```
Option Explicit
Option Strict

Imports System
Imports System.Windows.Forms
Imports System.Drawing

Public Class FlashTrackBar
    Inherits Control

    ' The event does not have any data, so EventHandler is adequate
    ' as the event delegate.
    ' Define the event member using the event keyword.
    ' In this case, for efficiency, the event is defined
    ' using the event property construct.
    Public Event ValueChanged As EventHandler
    ' The protected method that raises the ValueChanged
    ' event when the value has actually
    ' changed. Derived controls can override this method.
    Protected Overridable Sub OnValuechanged(e As EventArgs)
        RaiseEvent ValueChanged(Me, e)
    End Sub
End Class
```

```
using System;
using System.Windows.Forms;
using System.Drawing;

public class FlashTrackBar : Control {
    // The event does not have any data, so EventHandler is adequate
    // as the event delegate.
    private EventHandler onValueChanged;
    // Define the event member using the event keyword.
    // In this case, for efficiency, the event is defined
    // using the event property construct.
    public event EventHandler ValueChanged {
        add {
            onValueChanged += value;
        }
        remove {
            onValueChanged -= value;
        }
    }
    // The protected method that raises the ValueChanged
    // event when the value has actually
    // changed. Derived controls can override this method.
    protected virtual void OnValueChanged(EventArgs e) {
        if (ValueChanged != null) {
            ValueChanged(this, e);
        }
    }
}
```

See Also

[Events in Windows Forms Controls](#)

[Events](#)

[Events](#)

Attributes in Windows Forms Controls

5/4/2018 • 2 min to read • [Edit Online](#)

The .NET Framework provides a variety of attributes you can apply to the members of your custom controls and components. Some of these attributes affect the run-time behavior of a class, and others affect the design-time behavior.

Attributes for Control and Component Properties

The following table shows the attributes you can apply to properties or other members of your custom controls and components. For an example that uses many of these attributes, see [How to: Apply Attributes in Windows Forms Controls](#).

ATTRIBUTE	DESCRIPTION
AmbientValueAttribute	Specifies the value to pass to a property to cause the property to get its value from another source. This is known as <i>ambience</i> .
BrowsableAttribute	Specifies whether a property or event should be displayed in a Properties window.
CategoryAttribute	Specifies the name of the category in which to group the property or event when displayed in a PropertyGrid control set to Categorized mode.
DefaultValueAttribute	Specifies the default value for a property.
DescriptionAttribute	Specifies a description for a property or event.
DisplayNameAttribute	Specifies the display name for a property, event, or <code>public void</code> method that takes no arguments.
EditorAttribute	Specifies the editor to use to change a property.
EditorBrowsableAttribute	Specifies that a property or method is viewable in an editor.
HelpKeywordAttribute	Specifies the context keyword for a class or member.
LocalizableAttribute	Specifies whether a property should be localized.
PasswordPropertyTextAttribute	Indicates that an object's text representation is obscured by characters such as asterisks.
ReadOnlyAttribute	Specifies whether the property this attribute is bound to is read-only or read/write at design time.
RefreshPropertiesAttribute	Indicates that the property grid should refresh when the associated property value changes.

ATTRIBUTE	DESCRIPTION
TypeConverterAttribute	Specifies what type to use as a converter for the object this attribute is bound to.

Attributes for Data Binding Properties

The following table shows the attributes you can apply to specify how your custom controls and components interact with data binding.

ATTRIBUTE	DESCRIPTION
BindableAttribute	Specifies whether a property is typically used for binding.
ComplexBindingPropertiesAttribute	Specifies the data source and data member properties for a component.
DefaultBindingPropertyAttribute	Specifies the default binding property for a component.
LookupBindingPropertiesAttribute	Specifies the data source and data member properties for a component.
AttributeProviderAttribute	Enables attribute redirection.

Attributes for Classes

The following table shows the attributes you can apply to specify the behavior of your custom controls and components at design time.

ATTRIBUTE	DESCRIPTION
DefaultEventAttribute	Specifies the default event for a component.
DefaultPropertyAttribute	Specifies the default property for a component.
DesignerAttribute	Specifies the class used to implement design-time services for a component.
DesignerCategoryAttribute	Specifies that the designer for a class belongs to a certain category.
ToolboxItemAttribute	Represents an attribute of a toolbox item.
ToolboxItemFilterAttribute	Specifies the filter string and filter type to use for a Toolbox item.

See Also

[Attribute](#)

[How to: Apply Attributes in Windows Forms Controls](#)

[Extending Design-Time Support](#)

[Developing Custom Windows Forms Controls with the .NET Framework](#)

How to: Apply Attributes in Windows Forms Controls

5/4/2018 • 32 min to read • [Edit Online](#)

To develop components and controls that interact correctly with the design environment and execute correctly at run time, you need to apply attributes correctly to classes and members.

Example

The following code example demonstrates how to use several attributes on a custom control. The control demonstrates a simple logging capability. When the control is bound to a data source, it displays the values sent by the data source in a `DataGridView` control. If a value exceeds the value specified by the `Threshold` property, a `ThresholdExceeded` event is raised.

The `AttributesDemoControl` logs values with a `LogEntry` class. The `LogEntry` class is a template class, which means it is parameterized on the type that it logs. For example, if the `AttributesDemoControl` is logging values of type `float`, each `LogEntry` instance is declared and used as follows.

```
// This method handles the timer's Elapsed event. It queries
// the performance counter for the next value, packs the
// value in a LogEntry object, and adds the new LogEntry to
// the list managed by the BindingSource.
private void timer1_Elapsed(
    object sender,
    System.Timers.ElapsedEventArgs e)
{
    // Get the latest value from the performance counter.
    float val = this.performanceCounter1.NextValue();

    // The performance counter returns values of type float,
    // but any type that implements the IComparable interface
    // will work.
    LogEntry<float> entry = new LogEntry<float>(val, DateTime.Now);

    // Add the new LogEntry to the BindingSource list.
    this.bindingSource1.Add(entry);
}
```

```

' This method handles the timer's Elapsed event. It queries
' the performance counter for the next value, packs the
' value in a LogEntry object, and adds the new LogEntry to
' the list managed by the BindingSource.
Private Sub timer1_Elapsed( _
    ByVal sender As Object, _
    ByVal e As System.Timers.ElapsedEventArgs) _
Handles timer1.Elapsed

    ' Get the latest value from the performance counter.
    Dim val As Single = Me.performanceCounter1.NextValue()

    ' The performance counter returns values of type float,
    ' but any type that implements the IComparable interface
    ' will work.
    Dim entry As LogEntry(Of Single) = _
        New LogEntry(Of Single)(val, DateTime.Now)

    ' Add the new LogEntry to the BindingSource list.
    Me.bindingSource1.Add(entry)
End Sub

```

NOTE

Because `LogEntry` is parameterized by an arbitrary type, it must use reflection to operate on the parameter type. For the threshold feature to work, the parameter type `T` must implement the `IComparable` interface.

The form that hosts the `AttributesDemoControl` queries a performance counter periodically. Each value is packaged in a `LogEntry` of the appropriate type and added to the form's `BindingSource`. The `AttributesDemoControl` receives the value through its data binding and displays the value in a `DataGridView` control.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.ComponentModel.Design;
using System.Diagnostics;
using System.Drawing;
using System.Data;
using System.Reflection;
using System.Text;
using System.Windows.Forms;

// This sample demonstrates the use of various attributes for
// authoring a control.
namespace AttributesDemoControlLibrary
{
    // This is the event handler delegate for the ThresholdExceeded event.
    public delegate void ThresholdExceededEventHandler(ThresholdExceededEventArgs e);

    // This control demonstrates a simple logging capability.
    [ComplexBindingProperties("DataSource", "DataMember")]
    [DefaultBindingProperty("TitleText")]
    [DefaultEvent("ThresholdExceeded")]
    [DefaultProperty("Threshold")]
    [HelpKeywordAttribute(typeof(UserControl))]
    [ToolboxItem("System.Windows.Forms.Design.AutoSizeToolboxItem, System.Design")]
    public class AttributesDemoControl : UserControl
    {

        // This backs the Threshold property.
        private object thresholdValue;
    }
}

```

```

// The default fore color value for DataGridView cells that
// contain values that exceed the threshold.
private static Color defaultAlertForeColorValue = Color.White;

// The default back color value for DataGridView cells that
// contain values that exceed the threshold.
private static Color defaultAlertBackColorValue = Color.Red;

// The ambient color value.
private static Color ambientColorValue = Color.Empty;

// The fore color value for DataGridView cells that
// contain values that exceed the threshold.
private Color alertForeColorValue = defaultAlertForeColorValue;

// The back color value for DataGridView cells that
// contain values that exceed the threshold.
private Color alertBackColorValue = defaultAlertBackColorValue;

// Child controls that comprise this UserControl.
private TableLayoutPanel tableLayoutPanel1;
private DataGridView dataGridView1;
private Label label1;

// Required for designer support.
private System.ComponentModel.IContainer components = null;

// Default constructor.
public AttributesDemoControl()
{
    InitializeComponent();
}

[Category("Appearance")]
[Description("The title of the log data.")]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)]
[Localizable(true)]
[HelpKeywordAttribute("AttributesDemoControlLibrary.AttributesDemoControl.TitleText")]
public string TitleText
{
    get
    {
        return this.label1.Text;
    }

    set
    {
        this.label1.Text = value;
    }
}

// The inherited Text property is hidden at design time and
// raises an exception at run time. This enforces a requirement
// that client code uses the TitleText property instead.
[Browsable(false)]
[EditorBrowsable(EditorBrowsableState.Never)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
public override string Text
{
    get
    {
        throw new NotSupportedException();
    }

    set
    {
        throw new NotSupportedException();
    }
}

```

```

[AmbientValue(typeof(Color), "Empty")]
[Category("Appearance")]
[DefaultValue(typeof(Color), "White")]
[Description("The color used for painting alert text.")]
public Color AlertForeColor
{
    get
    {
        if (this.alertForeColorValue == Color.Empty &&
            this.Parent != null)
        {
            return Parent.ForeColor;
        }

        return this.alertForeColorValue;
    }

    set
    {
        this.alertForeColorValue = value;
    }
}

// This method is used by designers to enable resetting the
// property to its default value.
public void ResetAlertForeColor()
{
    this.AlertForeColor = AttributesDemoControl.defaultAlertForeColorValue;
}

// This method indicates to designers whether the property
// value is different from the ambient value, in which case
// the designer should persist the value.
private bool ShouldSerializeAlertForeColor()
{
    return (this.alertForeColorValue != AttributesDemoControl.ambientColorValue);
}

[AmbientValue(typeof(Color), "Empty")]
[Category("Appearance")]
[DefaultValue(typeof(Color), "Red")]
[Description("The background color for painting alert text.")]
public Color AlertBackColor
{
    get
    {
        if (this.alertBackColorValue == Color.Empty &&
            this.Parent != null)
        {
            return Parent.BackColor;
        }

        return this.alertBackColorValue;
    }

    set
    {
        this.alertBackColorValue = value;
    }
}

// This method is used by designers to enable resetting the
// property to its default value.
public void ResetAlertBackColor()
{
    this.AlertBackColor = AttributesDemoControl.defaultAlertBackColorValue;
}

```

```

// This method indicates to designers whether the property
// value is different from the ambient value, in which case
// the designer should persist the value.
private bool ShouldSerializeAlertBackColor()
{
    return (this.alertBackColorValue != AttributesDemoControl.ambientColorValue);
}

[Category("Data")]
[Description("Indicates the source of data for the control.")]
[RefreshProperties(RefreshProperties.Repaint)]
[AttributeProvider(typeof(IListSource))]
public object DataSource
{
    get
    {
        return this.dataGridView1.DataSource;
    }

    set
    {
        this.dataGridView1.DataSource = value;
    }
}

[Category("Data")]
[Description("Indicates a sub-list of the data source to show in the control.")]
public string DataMember
{
    get
    {
        return this.dataGridView1.DataMember;
    }

    set
    {
        this.dataGridView1.DataMember = value;
    }
}

// This property would normally have its BrowsableAttribute
// set to false, but this code demonstrates using
// ReadOnlyAttribute, so BrowsableAttribute is true to show
// it in any attached PropertyGrid control.
[Browsable(true)]
[Category("Behavior")]
[Description("The timestamp of the latest entry.")]
[ReadOnly(true)]
public DateTime CurrentLogTime
{
    get
    {
        int lastRowIndex =
            this.dataGridView1.Rows.GetLastRow(
                DataGridViewElementStates.Visible);

        if (lastRowIndex > -1)
        {
            DataGridViewRow lastRow = this.dataGridView1.Rows[lastRowIndex];
            DataGridViewCell lastCell = lastRow.Cells["EntryTime"];
            return ((DateTime)lastCell.Value);
        }
        else
        {
            return DateTime.MinValue;
        }
    }

    set

```

```

    {
    }
}

[Category("Behavior")]
[Description("The value above which the ThresholdExceeded event will be raised.")]
public object Threshold
{
    get
    {
        return this.thresholdValue;
    }

    set
    {
        this.thresholdValue = value;
    }
}

// This property exists only to demonstrate the
// PasswordPropertyText attribute. When this control
// is attached to a PropertyGrid control, the returned
// string will be displayed with obscuring characters
// such as asterisks. This property has no other effect.
[Category("Security")]
[Description("Demonstrates PasswordPropertyTextAttribute.")]
[PasswordPropertyText(true)]
public string Password
{
    get
    {
        return "This is a demo password.";
    }
}

// This property exists only to demonstrate the
// DisplayName attribute. When this control
// is attached to a PropertyGrid control, the
// property will appear as "RenamedProperty"
// instead of "MisnamedProperty".
[Description("Demonstrates DisplayNameAttribute.")]
[DisplayName("RenamedProperty")]
public bool MisnamedProperty
{
    get
    {
        return true;
    }
}

// This is the declaration for the ThresholdExceeded event.
public event ThresholdEventHandler ThresholdExceeded;

#region Implementation

// This is the event handler for the DataGridView control's
// CellFormatting event. Handling this event allows the
// AttributesDemoControl to examine the incoming log entries
// from the data source as they arrive.
//
// If the cell for which this event is raised holds the
// log entry's timestamp, the cell value is formatted with
// the full date/time pattern.
//
// Otherwise, the cell's value is assumed to hold the log
// entry value. If the value exceeds the threshold value,
// the cell is painted with the colors specified by the
// AlertForeColor and AlertBackColor properties, after which
// the ThresholdExceeded is raised. For this comparison to

```

```

// succeed, the log entry's type must implement the IComparable
// interface.
private void dataGridView1_CellFormatting(
    object sender,
    DataGridViewCellFormattingEventArgs e)
{
    try
    {
        if (e.Value != null)
        {
            if (e.Value is DateTime)
            {
                // Display the log entry time with the
                // full date/time pattern (long time).
                e.CellStyle.Format = "F";
            }
            else
            {
                // Scroll to the most recent entry.
                DataGridViewRow row = this.dataGridView1.Rows[e.RowIndex];
                DataGridViewCell cell = row.Cells[e.ColumnIndex];
                this.dataGridView1.FirstDisplayedCell = cell;

                if (this.thresholdValue != null)
                {
                    // Get the type of the log entry.
                    object val = e.Value;
                    Type paramType = val.GetType();

                    // Compare the log entry value to the threshold value.
                    // Use reflection to call the CompareTo method on the
                    // template parameter's type.
                    int compareVal = (int)paramType.InvokeMember(
                        "CompareTo",
                        BindingFlags.Default | BindingFlags.InvokeMethod,
                        null,
                        e.Value,
                        new object[] { this.thresholdValue },
                        System.Globalization.CultureInfo.InvariantCulture);

                    // If the log entry value exceeds the threshold value,
                    // set the cell's fore color and back color properties
                    // and raise the ThresholdExceeded event.
                    if (compareVal > 0)
                    {
                        e.CellStyle.BackColor = this.alertBackColorValue;
                        e.CellStyle.ForeColor = this.alertForeColorValue;

                        ThresholdExceededEventArgs teeaa =
                            new ThresholdExceededEventArgs(
                                this.thresholdValue,
                                e.Value);
                        this.ThresholdExceeded(teeaa);
                    }
                }
            }
        }
    }
    catch (Exception ex)
    {
        Trace.WriteLine(ex.Message);
    }
}

protected override void Dispose(bool disposing)
{
    if (disposing && (components != null))
    {
        components.Dispose();
    }
}

```

```

        }

        base.Dispose(disposing);
    }

    private void InitializeComponent()
    {
        System.Windows.Forms.DataGridViewCellStyle dataGridViewCellStyle1 = new
System.Windows.Forms.DataGridViewCellStyle();
        this.tableLayoutPanel1 = new System.Windows.Forms.TableLayoutPanel();
        this.dataGridView1 = new System.Windows.Forms.DataGridView();
        this.label1 = new System.Windows.Forms.Label();
        this.tableLayoutPanel1.SuspendLayout();
        ((System.ComponentModel.ISupportInitialize)(this.dataGridView1)).BeginInit();
        this.SuspendLayout();
        // 
        // tableLayoutPanel1
        // 
        this.tableLayoutPanel1.AutoResize = true;
        this.tableLayoutPanel1.ColumnCount = 1;
        this.tableLayoutPanel1.ColumnStyles.Add(new
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Absolute, 100F));
        this.tableLayoutPanel1.ColumnStyles.Add(new
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Absolute, 100F));
        this.tableLayoutPanel1.Controls.Add(this.dataGridView1, 0, 1);
        this.tableLayoutPanel1.Controls.Add(this.label1, 0, 0);
        this.tableLayoutPanel1.Dock = System.Windows.Forms.DockStyle.Fill;
        this.tableLayoutPanel1.Location = new System.Drawing.Point(10, 10);
        this.tableLayoutPanel1.Name = "tableLayoutPanel1";
        this.tableLayoutPanel1.Padding = new System.Windows.Forms.Padding(10);
        this.tableLayoutPanel1.RowCount = 2;
        this.tableLayoutPanel1.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 10F));
        this.tableLayoutPanel1.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 80F));
        this.tableLayoutPanel1.Size = new System.Drawing.Size(425, 424);
        this.tableLayoutPanel1.TabIndex = 0;
        // 
        // dataGridView1
        // 
        this.dataGridView1.AllowUserToAddRows = false;
        this.dataGridView1.AllowUserToDeleteRows = false;
        this.dataGridView1.AllowUserToOrderColumns = true;
        this.dataGridView1.ColumnHeadersHeightSizeMode = DataGridViewColumnHeadersHeightSizeMode.AutoSize;
        this.dataGridView1.AutoSizeColumnsMode = System.Windows.Forms.DataGridViewAutoSizeColumnsMode.AllCells;
        dataGridViewCellStyle1.Alignment = System.Windows.Forms.DataGridViewContentAlignment.MiddleLeft;
        dataGridViewCellStyle1.BackColor = System.Drawing.SystemColors.Control;
        dataGridViewCellStyle1.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
        dataGridViewCellStyle1.ForeColor = System.Drawing.SystemColors.WindowText;
        dataGridViewCellStyle1.SelectionBackColor = System.Drawing.SystemColors.Highlight;
        dataGridViewCellStyle1.SelectionForeColor = System.Drawing.SystemColors.HighlightText;
        dataGridViewCellStyle1.WrapMode = System.Windows.Forms.DataGridViewTriState.False;
        this.dataGridView1.ColumnHeadersDefaultCellStyle = dataGridViewCellStyle1;
        this.dataGridView1.ColumnHeadersHeight = 4;
        this.dataGridView1.Dock = System.Windows.Forms.DockStyle.Fill;
        this.dataGridView1.Location = new System.Drawing.Point(13, 57);
        this.dataGridView1.Name = "dataGridView1";
        this.dataGridView1.ReadOnly = true;
        this.dataGridView1.RowHeadersVisible = false;
        this.dataGridView1.Size = new System.Drawing.Size(399, 354);
        this.dataGridView1.TabIndex = 1;
        this.dataGridView1.CellFormatting += new
System.Windows.Forms.DataGridViewCellFormattingEventHandler(this.dataGridView1_CellFormatting);
        // 
        // label1
        // 
        this.label1.AutoSize = true;
        this.label1.Location = new System.Drawing.Point(13, 13);
        this.label1.Name = "label1";
        this.label1.Size = new System.Drawing.Size(59, 16);
        this.label1.TabIndex = 2;
        this.label1.Text = "label1";
        this.SuspendLayout();
        this.Controls.Add(this.tableLayoutPanel1);
        this.Controls.Add(this.label1);
        this.Name = "Form1";
        this.Text = "Form1";
        this.ResumeLayout(false);
    }
}

```

```

        this.label1.AutoSize = true;
        this.label1.BackColor = System.Drawing.SystemColors.Control;
        this.label1.Dock = System.Windows.Forms.DockStyle.Fill;
        this.label1.Location = new System.Drawing.Point(13, 13);
        this.label1.Name = "label1";
        this.label1.Size = new System.Drawing.Size(399, 38);
        this.label1.TabIndex = 2;
        this.label1.Text = "label1";
        this.label1.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;
        //
        // AttributesDemoControl
        //
        this.Controls.Add(this.tableLayoutPanel1);
        this.Name = "AttributesDemoControl";
        this.Padding = new System.Windows.Forms.Padding(10);
        this.Size = new System.Drawing.Size(445, 444);
        this.tableLayoutPanel1.ResumeLayout(false);
        this.tableLayoutPanel1.PerformLayout();
        ((System.ComponentModel.ISupportInitialize)(this.dataGridView1)).EndInit();
        this.ResumeLayout(false);
        this.PerformLayout();
        this.PerformLayout();

    }

}

// This is the EventArgs class for the ThresholdExceeded event.
public class ThresholdExceededEventArgs : EventArgs
{
    private object thresholdValue = null;
    private object exceedingValue = null;

    public ThresholdExceededEventArgs(
        object thresholdValue,
        object exceedingValue)
    {
        this.thresholdValue = thresholdValue;
        this.exceedingValue = exceedingValue;
    }

    public object ThresholdValue
    {
        get
        {
            return this.thresholdValue;
        }
    }

    public object ExceedingValue
    {
        get
        {
            return this.exceedingValue;
        }
    }
}

// This class encapsulates a log entry. It is a parameterized
// type (also known as a template class). The parameter type T
// defines the type of data being logged. For threshold detection
// to work, this type must implement the IComparable interface.
[TypeConverter("LogEntryTypeConverter")]
public class LogEntry<T> where T : IComparable
{
    private T entryValue;
    private DateTime entryTimeValue;

    public LogEntry(
        T entryValue,
        DateTime entryTimeValue)
    {
        this.entryValue = entryValue;
        this.entryTimeValue = entryTimeValue;
    }

    public T Value
    {
        get { return entryValue; }
        set { entryValue = value; }
    }

    public DateTime Time
    {
        get { return entryTimeValue; }
        set { entryTimeValue = value; }
    }

    public int CompareTo(T other)
    {
        if (other == null)
            return 1;
        else
            return entryValue.CompareTo(other);
    }
}

```

```

        I value,
        DateTime time)
{
    this.entryValue = value;
    this.entryTimeValue = time;
}

public T Entry
{
    get
    {
        return this.entryValue;
    }
}

public DateTime EntryTime
{
    get
    {
        return this.entryTimeValue;
    }
}

// This is the TypeConverter for the LogEntry class.
public class LogEntryTypeConverter : TypeConverter
{
    public override bool CanConvertFrom(
        ITypeDescriptorContext context,
        Type sourceType)
    {
        if (sourceType == typeof(string))
        {
            return true;
        }

        return base.CanConvertFrom(context, sourceType);
    }

    public override object ConvertFrom(
        ITypeDescriptorContext context,
        System.Globalization.CultureInfo culture,
        object value)
    {
        if (value is string)
        {
            string[] v = ((string)value).Split(new char[] { '|' });

            Type paramType = typeof(T);
            T entryValue = (T)paramType.InvokeMember(
                "Parse",
                BindingFlags.Static | BindingFlags.Public | BindingFlags.InvokeMethod,
                null,
                null,
                new string[] { v[0] },
                culture);

            return new LogEntry<T>(
                entryValue,
                DateTime.Parse(v[2]));
        }

        return base.ConvertFrom(context, culture, value);
    }

    public override object ConvertTo(
        ITypeDescriptorContext context,
        System.Globalization.CultureInfo culture,
        object value,
        Type destinationType)

```

```
        {
            if (destinationType == typeof(string))
            {
                LogEntry<T> le = value as LogEntry<T>;
                string stringRepresentation =
                    String.Format("{0} | {1}",
                                  le.Entry,
                                  le.EntryTime);
                return stringRepresentation;
            }
            return base.ConvertTo(context, culture, value, destinationType);
        }
    }
}
```

```
Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.ComponentModel.Design
Imports System.Diagnostics
Imports System.Drawing
Imports System.Data
Imports System.Reflection
Imports System.Text
Imports System.Windows.Forms

' This sample demonstrates the use of various attributes for
' authoring a control.

Namespace AttributesDemoControlLibrary

    ' This is the event handler delegate for the ThresholdExceeded event.
    Delegate Sub ThresholdExceededEventHandler(ByVal e As ThresholdExceededEventArgs)

    ' This control demonstrates a simple logging capability.
    <ComplexBindingProperties("DataSource", "DataMember"), _
    DefaultBindingProperty("TitleText"), _
    DefaultEvent("ThresholdExceeded"), _
    DefaultProperty("Threshold"), _
    HelpKeywordAttribute(GetType(UserControl)), _
    ToolboxItem("System.Windows.Forms.Design.AutoSizeToolboxItem, System.Design")> _
    Public Class AttributesDemoControl
        Inherits UserControl

        ' This backs the Threshold property.
        Private thresholdValue As Object

        ' The default fore color value for DataGridView cells that
        ' contain values that exceed the threshold.
        Private Shared defaultAlertForeColorValue As Color = Color.White

        ' The default back color value for DataGridView cells that
        ' contain values that exceed the threshold.
        Private Shared defaultAlertBackColorValue As Color = Color.Red

        ' The ambient color value.
        Private Shared ambientColorValue As Color = Color.Empty

        ' The fore color value for DataGridView cells that
        ' contain values that exceed the threshold.
        Private alertForeColorValue As Color = defaultAlertForeColorValue

        ' The back color value for DataGridView cells that
        ' contain values that exceed the threshold.
        Private alertBackColorValue As Color = defaultAlertBackColorValue
```

```

Private alertBackColorValue As Color = defaultAlertBackColorValue

' Child controls that comprise this UserControl.
Private tableLayoutPanel1 As TableLayoutPanel
Private WithEvents dataGridView1 As DataGridView
Private label1 As Label

' Required for designer support.
Private components As System.ComponentModel.IContainer = Nothing

' Default constructor.
Public Sub New()
    InitializeComponent()
End Sub

<Category("Appearance"), _
Description("The title of the log data."), _
DesignerSerializationVisibility(DesignerSerializationVisibility.Visible), Localizable(True), _
HelpKeywordAttribute("AttributesDemoControlLibrary.AttributesDemoControl.TitleText")> _
Public Property TitleText() As String
    Get
        Return Me.label1.Text
    End Get

    Set(ByVal value As String)
        Me.label1.Text = value
    End Set
End Property

' The inherited Text property is hidden at design time and
' raises an exception at run time. This enforces a requirement
' that client code uses the TitleText property instead.
<Browsable(False), _
EditorBrowsable(EditorBrowsableState.Never), _
DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)> _
Public Overrides Property Text() As String
    Get
        Throw New NotSupportedException()
    End Get
    Set(ByVal value As String)
        Throw New NotSupportedException()
    End Set
End Property

<AmbientValue(GetType(Color), "Empty"), _
Category("Appearance"), _
DefaultValue(GetType(Color), "White"), _
Description("The color used for painting alert text.")> _
Public Property AlertForeColor() As Color
    Get
        If Me.alertForeColorValue = Color.Empty AndAlso (Me.Parent IsNot Nothing) Then
            Return Parent.ForeColor
        End If

        Return Me.alertForeColorValue
    End Get

    Set(ByVal value As Color)
        Me.alertForeColorValue = value
    End Set
End Property

' This method is used by designers to enable resetting the
' property to its default value.
Public Sub ResetAlertForeColor()
    Me.AlertForeColor = AttributesDemoControl.defaultAlertForeColorValue
End Sub

' This method indicates to designers whether the property

```

```

' value is different from the ambient value, in which case
' the designer should persist the value.
Private Function ShouldSerializeAlertForeColor() As Boolean
    Return Me.alertForeColorValue <> AttributesDemoControl.ambientColorValue
End Function

<AmbientValue(GetType(Color), "Empty"), _
Category("Appearance"), _
DefaultValue(GetType(Color), "Red"), _
Description("The background color for painting alert text.")> _
Public Property AlertBackColor() As Color
    Get
        If Me.alertBackColorValue = Color.Empty AndAlso (Me.Parent IsNot Nothing) Then
            Return Parent.BackColor
        End If

        Return Me.alertBackColorValue
    End Get

    Set(ByVal value As Color)
        Me.alertBackColorValue = value
    End Set
End Property

' This method is used by designers to enable resetting the
' property to its default value.
Public Sub ResetAlertBackColor()
    Me.AlertBackColor = AttributesDemoControl.defaultAlertBackColorValue
End Sub

' This method indicates to designers whether the property
' value is different from the ambient value, in which case
' the designer should persist the value.
Private Function ShouldSerializeAlertBackColor() As Boolean
    Return Me.alertBackColorValue <> AttributesDemoControl.ambientColorValue
End Function 'ShouldSerializeAlertBackColor

<Category("Data"), _
Description("Indicates the source of data for the control."), _
RefreshProperties(RefreshProperties.Repaint), _
AttributeProvider(GetType(IListSource))> _
Public Property DataSource() As Object
    Get
        Return Me.dataGridView1.DataSource
    End Get

    Set(ByVal value As Object)
        Me.dataGridView1.DataSource = value
    End Set
End Property

<Category("Data"), _
Description("Indicates a sub-list of the data source to show in the control.")> _
Public Property DataMember() As String
    Get
        Return Me.dataGridView1.DataMember
    End Get

    Set(ByVal value As String)
        Me.dataGridView1.DataMember = value
    End Set
End Property

' This property would normally have its BrowsableAttribute
' set to false, but this code demonstrates using
' ReadOnlyAttribute, so BrowsableAttribute is true to show
' it in any attached PropertyGrid control.
<Browsable(True), _
Category("Behavior"), _

```

```

        Description("The timestamp of the latest entry."), _
        ReadOnlyAttribute(True)> _
    Public Property CurrentLogTime() As DateTime
        Get
            Dim lastRowIndex As Integer = _
                Me.dataGridView1.Rows.GetLastRow(DataGridViewElementStates.Visible)

            If lastRowIndex > -1 Then
                Dim lastRow As DataGridViewRow = Me.dataGridView1.Rows(lastRowIndex)
                Dim lastCell As DataGridViewCell = lastRow.Cells("EntryTime")
                Return CType(lastCell.Value, DateTime)
            Else
                Return DateTime.MinValue
            End If
        End Get

        Set(ByVal value As DateTime)
        End Set
    End Property

    <Category("Behavior"), _
    Description("The value above which the ThresholdExceeded event will be raised.")> _
    Public Property Threshold() As Object
        Get
            Return Me.thresholdValue
        End Get

        Set(ByVal value As Object)
            Me.thresholdValue = value
        End Set
    End Property

    ' This property exists only to demonstrate the
    ' PasswordPropertyText attribute. When this control
    ' is attached to a PropertyGrid control, the returned
    ' string will be displayed with obscuring characters
    ' such as asterisks. This property has no other effect.
    <Category("Security"), _
    Description("Demonstrates PasswordPropertyTextAttribute."), _
    PasswordPropertyText(True)> _
    Public ReadOnly Property Password() As String
        Get
            Return "This is a demo password."
        End Get
    End Property

    ' This property exists only to demonstrate the
    ' DisplayName attribute. When this control
    ' is attached to a PropertyGrid control, the
    ' property will appear as "RenamedProperty"
    ' instead of "MisnamedProperty".
    <Description("Demonstrates DisplayNameAttribute."), _
    DisplayName("RenamedProperty")> _
    Public ReadOnly Property MisnamedProperty() As Boolean
        Get
            Return True
        End Get
    End Property

    ' This is the declaration for the ThresholdExceeded event.
    'Public Event ThresholdExceeded As ThresholdExceededEventHandler
    Public Event ThresholdExceeded(ByVal e As ThresholdExceededEventArgs)

#Region "Implementation"

    ' This is the event handler for the DataGridView control's
    ' CellFormatting event. Handling this event allows the
    ' AttributesDemoControl to examine the incoming log entries
    ' from the data source as they arrive.

```

```

' If the cell for which this event is raised holds the
' log entry's timestamp, the cell value is formatted with
' the full date/time pattern.
'

' Otherwise, the cell's value is assumed to hold the log
' entry value. If the value exceeds the threshold value,
' the cell is painted with the colors specified by the
' AlertForeColor and AlertBackColor properties, after which
' the ThresholdExceeded is raised. For this comparison to
' succeed, the log entry's type must implement the IComparable
' interface.

Private Sub dataGridView1_CellFormatting( _
    ByVal sender As Object, _
    ByVal e As DataGridViewCellFormattingEventArgs) _
Handles dataGridView1.CellFormatting

    Try
        If (e.Value IsNot Nothing) Then
            If TypeOf e.Value Is DateTime Then
                ' Display the log entry time with the
                ' full date/time pattern (long time).
                e.CellStyle.Format = "F"
            Else
                ' Scroll to the most recent entry.
                Dim row As DataGridViewRow = Me.dataGridView1.Rows(e.RowIndex)
                Dim cell As DataGridViewCell = row.Cells(e.ColumnIndex)
                Me.dataGridView1.FirstDisplayedCell = cell

                If (Me.thresholdValue IsNot Nothing) Then
                    ' Get the type of the log entry.
                    Dim val As Object = e.Value
                    Dim paramType As Type = val.GetType()

                    ' Compare the log entry value to the threshold value.
                    ' Use reflection to call the CompareTo method on the
                    ' template parameter's type.
                    Dim compareVal As Integer = _
                        Fix(paramType.InvokeMember("CompareTo", _
                        BindingFlags.Default Or BindingFlags.InvokeMethod, _
                        Nothing, _
                        e.Value, _
                        New Object() {Me.thresholdValue}, _
                        System.Globalization.CultureInfo.InvariantCulture))

                    ' If the log entry value exceeds the threshold value,
                    ' set the cell's fore color and back color properties
                    ' and raise the ThresholdExceeded event.
                    If compareVal > 0 Then
                        e.CellStyle.BackColor = Me.alertBackColorValue
                        e.CellStyle.ForeColor = Me.alertForeColorValue

                        Dim tee As New ThresholdExceededEventArgs(Me.thresholdValue, e.Value)
                        RaiseEvent ThresholdExceeded(tee)
                    End If
                End If
            End If
        End If
    Catch ex As Exception
        Trace.WriteLine(ex.Message)
    End Try
End Sub

Protected Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing AndAlso (components IsNot Nothing) Then
        components.Dispose()
    End If
    MyBase.Dispose(disposing)
End Sub

```

```

Private Sub InitializeComponent()
    Dim DataGridViewCellStyle1 As System.Windows.Forms.DataGridViewColumnStyle = New
System.Windows.Forms.DataGridViewColumnStyle
    Me.tableLayoutPanel1 = New System.Windows.Forms.TableLayoutPanel
    Me.dataGridView1 = New System.Windows.Forms.DataGridView
    Me.label1 = New System.Windows.Forms.Label
    Me.tableLayoutPanel1.SuspendLayout()
    CType(Me.dataGridView1, System.ComponentModel.ISupportInitialize).BeginInit()
    Me.SuspendLayout()
    '
    'tableLayoutPanel1
    '
    '
    Me.tableLayoutPanel1.AutoSize = True
    Me.tableLayoutPanel1.ColumnCount = 1
    Me.tableLayoutPanel1.ColumnStyles.Add(New
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Absolute, 100.0!))
    Me.tableLayoutPanel1.ColumnStyles.Add(New
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Absolute, 100.0!))
    Me.tableLayoutPanel1.Controls.Add(Me.dataGridView1, 0, 1)
    Me.tableLayoutPanel1.Controls.Add(Me.label1, 0, 0)
    Me.tableLayoutPanel1.Dock = System.Windows.Forms.DockStyle.Fill
    Me.tableLayoutPanel1.Location = New System.Drawing.Point(10, 10)
    Me.tableLayoutPanel1.Name = "tableLayoutPanel1"
    Me.tableLayoutPanel1.Padding = New System.Windows.Forms.Padding(10)
    Me.tableLayoutPanel1.RowCount = 2
    Me.tableLayoutPanel1.RowStyles.Add(New
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 10.0!))
    Me.tableLayoutPanel1.RowStyles.Add(New
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 80.0!))
    Me.tableLayoutPanel1.RowStyles.Add(New
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 10.0!))
    Me.tableLayoutPanel1.Size = New System.Drawing.Size(425, 424)
    Me.tableLayoutPanel1.TabIndex = 0
    '
    'dataGridView1
    '
    '
    Me.dataGridView1.AllowUserToAddRows = False
    Me.dataGridView1.AllowUserToDeleteRows = False
    Me.dataGridView1.AllowUserToOrderColumns = True
    Me.dataGridView1.ColumnHeadersHeightSizeMode = DataGridViewColumnHeadersHeightSizeMode.AutoSize
    Me.dataGridView1.AutoSizeColumnsMode = System.Windows.Forms.DataGridViewAutoSizeColumnsMode.AllCells
    DataGridViewCellStyle1.Alignment = System.Windows.Forms.DataGridViewContentAlignment.MiddleLeft
    DataGridViewCellStyle1.BackColor = System.Drawing.SystemColors.Control
    DataGridViewCellStyle1.ForeColor = System.Drawing.SystemColors.WindowText
    DataGridViewCellStyle1.SelectionBackColor = System.Drawing.SystemColors.Highlight
    DataGridViewCellStyle1.SelectionForeColor = System.Drawing.SystemColors.HighlightText
    DataGridViewCellStyle1.WrapMode = System.Windows.Forms.DataGridViewTriState.[False]
    Me.dataGridView1.ColumnHeadersDefaultCellStyle = DataGridViewCellStyle1
    Me.dataGridView1.ColumnHeadersHeight = 4
    Me.dataGridView1.Dock = System.Windows.Forms.DockStyle.Fill
    Me.dataGridView1.Location = New System.Drawing.Point(13, 57)
    Me.dataGridView1.Name = "dataGridView1"
    Me.dataGridView1.ReadOnly = True
    Me.dataGridView1.RowHeadersVisible = False
    Me.dataGridView1.Size = New System.Drawing.Size(399, 354)
    Me.dataGridView1.TabIndex = 1
    '
    'label1
    '
    '
    Me.label1.AutoSize = True
    Me.label1.BackColor = System.Drawing.SystemColors.Control
    Me.label1.Dock = System.Windows.Forms.DockStyle.Fill
    Me.label1.Location = New System.Drawing.Point(13, 13)
    Me.label1.Name = "label1"
    Me.label1.Size = New System.Drawing.Size(399, 38)
    Me.label1.TabIndex = 2
    Me.label1.Text = "label1"
    Me.label1.TextAlign = System.Drawing.ContentAlignment.MiddleCenter

```

```

Me.TableLayoutPanel1 = System.Drawing.ContentAlignment.MiddleCenter
'
' AttributesDemoControl
'

Me.Controls.Add(Me.tableLayoutPanel1)
Me.Name = "AttributesDemoControl"
Me.Padding = New System.Windows.Forms.Padding(10)
Me.Size = New System.Drawing.Size(445, 444)
Me.tableLayoutPanel1.ResumeLayout(False)
Me.tableLayoutPanel1.PerformLayout()
 CType(Me.dataGridView1, System.ComponentModel.ISupportInitialize).EndInit()
Me.ResumeLayout(False)
Me.PerformLayout()

End Sub 'InitializeComponent

#End Region

End Class

' This is the EventArgs class for the ThresholdExceeded event.
Public Class ThresholdExceededEventArgs
    Inherits EventArgs
    Private thresholdVal As Object = Nothing
    Private exceedingVal As Object = Nothing

    Public Sub New(ByVal thresholdValue As Object, ByVal exceedingValue As Object)
        Me.thresholdVal = thresholdValue
        Me.exceedingVal = exceedingValue
    End Sub

    Public ReadOnly Property ThresholdValue() As Object
        Get
            Return Me.thresholdVal
        End Get
    End Property

    Public ReadOnly Property ExceedingValue() As Object
        Get
            Return Me.exceedingVal
        End Get
    End Property
End Class

' This class encapsulates a log entry. It is a parameterized
' type (also known as a template class). The parameter type T
' defines the type of data being logged. For threshold detection
' to work, this type must implement the IComparable interface.
<TypeConverter("LogEntryTypeConverter")> _
Public Class LogEntry(Of T As IComparable)

    Private entryValue As T

    Private entryTimeValue As DateTime

    Public Sub New(ByVal value As T, ByVal time As DateTime)
        Me.entryValue = value
        Me.entryTimeValue = time
    End Sub

    Public ReadOnly Property Entry() As T
        Get
            Return Me.entryValue
        End Get
    End Property

```

```

    Public Readonly Property EntryTime() As Dateime
        Get
            Return Me.entryTimeValue
        End Get
    End Property

    ' This is the TypeConverter for the LogEntry class.
    Public Class LogEntryTypeConverter
        Inherits TypeConverter

        Public Overrides Function CanConvertFrom( _
            ByVal context As ITypeDescriptorContext, _
            ByVal sourceType As Type) As Boolean
            If sourceType Is GetType(String) Then
                Return True
            End If

            Return MyBase.CanConvertFrom(context, sourceType)
        End Function

        Public Overrides Function ConvertFrom( _
            ByVal context As ITypeDescriptorContext, _
            ByVal culture As System.Globalization.CultureInfo, _
            ByVal value As Object) As Object
            If TypeOf value Is String Then
                Dim v As String() = CStr(value).Split(New Char() {"|"c})

                Dim paramType As Type = GetType(T)
                Dim entryValue As T = CType(paramType.InvokeMember("Parse", BindingFlags.Static Or
BindingFlags.Public Or BindingFlags.InvokeMethod, Nothing, Nothing, New String() {v(0)}, culture), T)

                Return New LogEntry(Of T)(entryValue, DateTime.Parse(v(2))) '
            End If

            Return MyBase.ConvertFrom(context, culture, value)
        End Function

        Public Overrides Function ConvertTo(ByVal context As ITypeDescriptorContext, ByVal culture As
System.Globalization.CultureInfo, ByVal value As Object, ByVal destinationType As Type) As Object
            If destinationType Is GetType(String) Then

                Dim le As LogEntry(Of T) = CType(value, LogEntry(Of T))

                Dim stringRepresentation As String = String.Format("{0} | {1}", le.Entry, le.EntryTime)

                Return stringRepresentation
            End If

            Return MyBase.ConvertTo(context, culture, value, destinationType)
        End Function
    End Class
End Class

End Namespace

```

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.Windows.Forms;
using AttributesDemoControlLibrary;

// This sample demonstrates using the AttributesDemoControl to log
// data from a data source.
namespace AttributesDemoControlTest

```

```
namespace AttributesDemoControl1Test
{
    public class Form1 : Form
    {
        private BindingSource bindingSource1;
        private System.Diagnostics.PerformanceCounter performanceCounter1;
        private Button startButton;
        private Button stopButton;
        private System.Timers.Timer timer1;
        private ToolStripStatusLabel statusStripPanel1;
        private NumericUpDown numericUpDown1;
        private GroupBox groupBox1;
        private GroupBox groupBox2;
        private TableLayoutPanel tableLayoutPanel1;
        private AttributesDemoControl attributesDemoControl1;
        private System.ComponentModel.IContainer components = null;

        // This form uses an AttributesDemoControl to display a stream
        // of LogEntry objects. The data stream is generated by polling
        // a performance counter and communicating the counter values
        // to the control with data binding.
        public Form1()
        {
            InitializeComponent();

            // Set the initial value of the threshold up/down control
            // to the control's threshold value.
            this.numericUpDown1.Value =
                (decimal)(float)this.attributesDemoControl1.Threshold;

            // Assign the performance counter's name to the control's
            // title text.
            this.attributesDemoControl1.TitleText =
                this.performanceCounter1.CounterName;
        }

        // This method handles the ThresholdExceeded event. It posts
        // the value that exceeded the threshold to the status strip.
        private void attributesDemoControl1_ThresholdExceeded(
            ThresholdExceededEventArgs e)
        {
            string msg = String.Format(
                "{0}: Value {1} exceeded threshold {2}",
                this.attributesDemoControl1.CurrentLogTime,
                e.ExceedingValue,
                e.ThresholdValue);

            this.ReportStatus( msg );
        }

        // This method handles the timer's Elapsed event. It queries
        // the performance counter for the next value, packs the
        // value in a LogEntry object, and adds the new LogEntry to
        // the list managed by the BindingSource.
        private void timer1_Elapsed(
            object sender,
            System.Timers.ElapsedEventArgs e)
        {
            // Get the latest value from the performance counter.
            float val = this.performanceCounter1.NextValue();

            // The performance counter returns values of type float,
            // but any type that implements the IComparable interface
            // will work.
            LogEntry<float> entry = new LogEntry<float>(val, DateTime.Now);

            // Add the new LogEntry to the BindingSource list.
            this.bindingSource1.Add(entry);
        }
    }
}
```

```

private void numericUpDown1_ValueChanged(object sender, EventArgs e)
{
    this.attributesDemoControl1.Threshold =
        (float)this.numericUpDown1.Value;

    string msg = String.Format(
        "Threshold changed to {0}",
        this.attributesDemoControl1.Threshold);

    this.ReportStatus(msg);
}

private void startButton_Click(object sender, EventArgs e)
{
    this.ReportStatus(DateTime.Now + ": Starting");

    this.timer1.Start();
}

private void stopButton_Click(object sender, EventArgs e)
{
    this.ReportStatus(DateTime.Now + ": Stopping");

    this.timer1.Stop();
}

private void ReportStatus(string msg)
{
    if (msg != null)
    {
        this.statusStripPanel1.Text = msg;
    }
}

[STAThread]
static void Main()
{
    Application.EnableVisualStyles();
    Application.Run(new Form1());
}

protected override void Dispose(bool disposing)
{
    if (disposing && (components != null))
    {
        components.Dispose();
    }
    base.Dispose(disposing);
}

private void InitializeComponent()
{
    this.components = new System.ComponentModel.Container();
    this.bindingSource1 = new System.Windows.Forms.BindingSource(this.components);
    this.performanceCounter1 = new System.Diagnostics.PerformanceCounter();
    this.startButton = new System.Windows.Forms.Button();
    this.stopButton = new System.Windows.Forms.Button();
    this.timer1 = new System.Timers.Timer();
    this.statusStripPanel1 = new System.Windows.Forms.ToolStripStatusLabel();
    this.numericUpDown1 = new System.Windows.Forms.NumericUpDown();
    this.groupBox1 = new System.Windows.Forms.GroupBox();
    this.groupBox2 = new System.Windows.Forms.GroupBox();
    this.tableLayoutPanel1 = new System.Windows.Forms.TableLayoutPanel();
    this.attributesDemoControl1 = new AttributesDemoControlLibrary.AttributesDemoControl();
    ((System.ComponentModel.ISupportInitialize)(this.bindingSource1)).BeginInit();
    ((System.ComponentModel.ISupportInitialize)(this.performanceCounter1)).BeginInit();
    ((System.ComponentModel.ISupportInitialize)(this.timer1)).BeginInit();
}

```

```
((System.ComponentModel.ISupportInitialize)(this.numericUpDown1)).BeginInit();
this.groupBox1.SuspendLayout();
this.groupBox2.SuspendLayout();
this.tableLayoutPanel1.SuspendLayout();
this.SuspendLayout();
//
// performanceCounter1
//
this.performanceCounter1.CategoryName = ".NET CLR Memory";
this.performanceCounter1.CounterName = "Gen 0 heap size";
this.performanceCounter1.InstanceName = "_Global_";
//
// startButton
//
this.startButton.Location = new System.Drawing.Point(31, 25);
this.startButton.Name = "startButton";
this.startButton.TabIndex = 1;
this.startButton.Text = "Start";
this.startButton.Click += new System.EventHandler(this.startButton_Click);
//
// stopButton
//
this.stopButton.Location = new System.Drawing.Point(112, 25);
this.stopButton.Name = "stopButton";
this.stopButton.TabIndex = 2;
this.stopButton.Text = "Stop";
this.stopButton.Click += new System.EventHandler(this.stopButton_Click);
//
// timer1
//
this.timer1.Interval = 1000;
this.timer1.SynchronizingObject = this;
this.timer1.Elapsed += new System.Timers.ElapsedEventHandler(this.timer1_Elapsed);
//
// statusStripPanel1
//
this.statusStripPanel1.BorderStyle = System.Windows.Forms.Border3DStyle.SunkenOuter;
this.statusStripPanel1.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Text;
this.statusStripPanel1.Name = "statusStripPanel1";
this.statusStripPanel1.Text = "Ready";
//
// numericUpDown1
//
this.numericUpDown1.Location = new System.Drawing.Point(37, 29);
this.numericUpDown1.Maximum = new decimal(new int[] {
    1410065408,
    2,
    0,
    0});
this.numericUpDown1.Name = "numericUpDown1";
this.numericUpDown1.TabIndex = 7;
this.numericUpDown1.ValueChanged += new System.EventHandler(this.numericUpDown1_ValueChanged);
//
// groupBox1
//
this.groupBox1.Anchor = System.Windows.Forms.AnchorStyles.None;
this.groupBox1.Controls.Add(this.numericUpDown1);
this.groupBox1.Location = new System.Drawing.Point(280, 326);
this.groupBox1.Name = "groupBox1";
this.groupBox1.Size = new System.Drawing.Size(200, 70);
this.groupBox1.TabIndex = 13;
this.groupBox1.TabStop = false;
this.groupBox1.Text = "Threshold Value";
//
// groupBox2
//
this.groupBox2.Anchor = System.Windows.Forms.AnchorStyles.None;
this.groupBox2.Controls.Add(this.startButton);
```

```

this.groupBox2.Controls.Add(this.stopButton);
this.groupBox2.Location = new System.Drawing.Point(26, 327);
this.groupBox2.Name = "groupBox2";
this.groupBox2.Size = new System.Drawing.Size(214, 68);
this.groupBox2.TabIndex = 14;
this.groupBox2.TabStop = false;
this.groupBox2.Text = "Logging";
//
// tableLayoutPanel1
//
this.tableLayoutPanel1.ColumnCount = 2;
this.tableLayoutPanel1.ColumnStyles.Add(new
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 50F));
this.tableLayoutPanel1.ColumnStyles.Add(new
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 50F));
this.tableLayoutPanel1.Controls.Add(this.groupBox2, 0, 1);
this.tableLayoutPanel1.Controls.Add(this.groupBox1, 1, 1);
this.tableLayoutPanel1.Dock = System.Windows.Forms.DockStyle.Fill;
this.tableLayoutPanel1.Location = new System.Drawing.Point(0, 0);
this.tableLayoutPanel1.Name = "tableLayoutPanel1";
this.tableLayoutPanel1.Padding = new System.Windows.Forms.Padding(10);
this.tableLayoutPanel1.RowCount = 2;
this.tableLayoutPanel1.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 80F));
this.tableLayoutPanel1.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 20F));
this.tableLayoutPanel1.Size = new System.Drawing.Size(514, 411);
this.tableLayoutPanel1.TabIndex = 15;
//
// attributesDemoControl1
//
this.tableLayoutPanel1.SetColumnSpan(this.attributesDemoControl1, 2);
this.attributesDemoControl1.DataMember = "";
this.attributesDemoControl1.DataSource = this.bindingSource1;
this.attributesDemoControl1.Dock = System.Windows.Forms.DockStyle.Fill;
this.attributesDemoControl1.Location = new System.Drawing.Point(13, 13);
this.attributesDemoControl1.Name = "attributesDemoControl1";
this.attributesDemoControl1.Padding = new System.Windows.Forms.Padding(10);
this.attributesDemoControl1.Size = new System.Drawing.Size(488, 306);
this.attributesDemoControl1.TabIndex = 0;
this.attributesDemoControl1.Threshold = 20000F;
this.attributesDemoControl1.TitleText = "TITLE";
this.attributesDemoControl1.ThresholdExceeded += new
AttributesDemoControlLibrary.ThresholdExceededEventHandler(this.attributesDemoControl1_ThresholdExceeded);
//
// Form1
//
this.BackColor = System.Drawing.SystemColors.Control;
this.ClientSize = new System.Drawing.Size(514, 430);
this.Controls.Add(this.tableLayoutPanel1);
this.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.Name = "Form1";
this.Text = "Form1";
((System.ComponentModel.ISupportInitialize)(this.bindingSource1)).EndInit();
((System.ComponentModel.ISupportInitialize)(this.performanceCounter1)).EndInit();
((System.ComponentModel.ISupportInitialize)(this.timer1)).EndInit();
((System.ComponentModel.ISupportInitialize)(this.numericUpDown1)).EndInit();
this.groupBox1.ResumeLayout(false);
this.groupBox2.ResumeLayout(false);
this.tableLayoutPanel1.ResumeLayout(false);
this.ResumeLayout(false);

}

}
}

```

```

Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Diagnostics
Imports System.Drawing
Imports System.Windows.Forms
Imports AttributesDemoControlLibrary

' This sample demonstrates using the AttributesDemoControl to log
' data from a data source.

Public Class Form1
    Inherits Form
    Private bindingSource1 As BindingSource
    Private performanceCounter1 As System.Diagnostics.PerformanceCounter
    Private WithEvents startButton As Button
    Private WithEvents stopButton As Button
    Private WithEvents timer1 As System.Timers.Timer
    Private statusStrip1 As ToolStripStatusLabel
    Private statusStripPanel1 As ToolStripStatusLabel
    Private WithEvents numericUpDown1 As NumericUpDown
    Private groupBox1 As GroupBox
    Private groupBox2 As GroupBox
    Private tableLayoutPanel1 As TableLayoutPanel
    Private WithEvents attributesDemoControl1 As AttributesDemoControl
    Private components As System.ComponentModel.IContainer = Nothing

    ' This form uses an AttributesDemoControl to display a stream
    ' of LogEntry objects. The data stream is generated by polling
    ' a performance counter and communicating the counter values
    ' to the control with data binding.

    Public Sub New()
        InitializeComponent()

        ' Set the initial value of the threshold up/down control
        ' to the control's threshold value.
        Me.numericUpDown1.Value = _
            System.Convert.ToDecimal( _
                System.Convert.ToString(Me.attributesDemoControl1.Threshold))

        ' Assign the performance counter's name to the control's
        ' title text.
        Me.attributesDemoControl1.TitleText = _
            Me.performanceCounter1.CounterName
    End Sub

    ' This method handles the ThresholdExceeded event. It posts
    ' the value that exceeded the threshold to the status strip.
    Private Sub attributesDemoControl1_ThresholdExceeded( _
        ByVal e As ThresholdExceededEventArgs) _
        Handles attributesDemoControl1.ThresholdExceeded
        Dim msg As String = String.Format( _
            "{0}: Value {1} exceeded threshold {2}", _
            Me.attributesDemoControl1.CurrentLogTime, _
            e.ExceedingValue, _
            e.ThresholdValue)

        Me.ReportStatus(msg)
    End Sub

    ' This method handles the timer's Elapsed event. It queries
    ' the performance counter for the next value, packs the
    ' value in a LogEntry object, and adds the new LogEntry to
    ' the list managed by the BindingSource.
    Private Sub timer1_Elapsed( _
        ByVal sender As Object, _
        ByVal e As System.Timers.ElapsedEventArgs) _

```

```

Handles timer1.Elapsed

    ' Get the latest value from the performance counter.
    Dim val As Single = Me.performanceCounter1.NextValue()

    ' The performance counter returns values of type float,
    ' but any type that implements the IComparable interface
    ' will work.
    Dim entry As LogEntry(Of Single) = _
        New LogEntry(Of Single)(val, DateTime.Now)

    ' Add the new LogEntry to the BindingSource list.
    Me.bindingSource1.Add(entry)
End Sub

Private Sub numericUpDown1_ValueChanged( _
    ByVal sender As Object, _
    ByVal e As EventArgs) _
Handles numericUpDown1.ValueChanged

    Me.attributesDemoControl1.Threshold = _
        System.Convert.ToSingle(Me.numericUpDown1.Value)

    Dim msg As String = String.Format( _
        "Threshold changed to {0}", _
        Me.attributesDemoControl1.Threshold)

    Me.ReportStatus(msg)
End Sub

Private Sub startButton_Click( _
    ByVal sender As Object, _
    ByVal e As EventArgs) _
Handles startButton.Click

    Me.ReportStatus((DateTime.Now.ToString() + ": Starting"))

    Me.timer1.Start()
End Sub

Private Sub stopButton_Click( _
    ByVal sender As Object, _
    ByVal e As EventArgs) _
Handles stopButton.Click

    Me.ReportStatus((DateTime.Now.ToString() + ": Stopping"))

    Me.timer1.Stop()
End Sub

Private Sub ReportStatus(msg As String)
    If (msg IsNot Nothing) Then
        Me.statusStripPanel1.Text = msg
    End If
End Sub

<STAThread()> _
Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())
End Sub

Protected Overrides Sub Dispose(disposing As Boolean)
    If disposing AndAlso (components IsNot Nothing) Then
        components.Dispose()
    End If
End Sub

```

```

End If
 MyBase.Dispose(disposing)
End Sub

Private Sub InitializeComponent()
 Me.components = New System.ComponentModel.Container()
 Me.bindingSource1 = New System.Windows.Forms.BindingSource(Me.components)
 Me.performanceCounter1 = New System.Diagnostics.PerformanceCounter()
 Me.startButton = New System.Windows.Forms.Button()
 Me.stopButton = New System.Windows.Forms.Button()
 Me.timer1 = New System.Timers.Timer()
 Me.statusStripPanel1 = New System.Windows.Forms.ToolStripStatusLabel()
 Me.numericUpDown1 = New System.Windows.Forms.NumericUpDown()
 Me.groupBox1 = New System.Windows.Forms.GroupBox()
 Me.groupBox2 = New System.Windows.Forms.GroupBox()
 Me.tableLayoutPanel1 = New System.Windows.Forms.TableLayoutPanel()
 Me.attributesDemoControl1 = New AttributesDemoControlLibrary.AttributesDemoControl()
 CType(Me.bindingSource1, System.ComponentModel.ISupportInitialize).BeginInit()
 CType(Me.performanceCounter1, System.ComponentModel.ISupportInitialize).BeginInit()
 CType(Me.timer1, System.ComponentModel.ISupportInitialize).BeginInit()
 CType(Me.numericUpDown1, System.ComponentModel.ISupportInitialize).BeginInit()
 Me.groupBox1.SuspendLayout()
 Me.groupBox2.SuspendLayout()
 Me.tableLayoutPanel1.SuspendLayout()
 Me.SuspendLayout()

' performanceCounter1
'

Me.performanceCounter1.CategoryName = ".NET CLR Memory"
Me.performanceCounter1.CounterName = "Gen 0 heap size"
Me.performanceCounter1.InstanceName = "_Global_"
'

' startButton
'

Me.startButton.Location = New System.Drawing.Point(31, 25)
Me.startButton.Name = "startButton"
Me.startButton.TabIndex = 1
Me.startButton.Text = "Start"
'

' stopButton
'

Me.stopButton.Location = New System.Drawing.Point(112, 25)
Me.stopButton.Name = "stopButton"
Me.stopButton.TabIndex = 2
Me.stopButton.Text = "Stop"
'

' timer1
'

Me.timer1.Interval = 1000
Me.timer1.SynchronizingObject = Me
'

' statusStripPanel1
'

Me.statusStripPanel1.BorderStyle = System.Windows.Forms.BorderStyle.SunkenOuter
Me.statusStripPanel1.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Text
Me.statusStripPanel1.Name = "statusStripPanel1"
Me.statusStripPanel1.Text = "Ready"
'

' numericUpDown1
'

Me.numericUpDown1.Location = New System.Drawing.Point(37, 29)
Me.numericUpDown1.Maximum = New Decimal(New Integer() {1410065408, 2, 0, 0})
Me.numericUpDown1.Name = "numericUpDown1"
Me.numericUpDown1.TabIndex = 7
'

' groupBox1
'

Me.groupBox1.Anchor = System.Windows.Forms.AnchorStyles.None

```

```
Me.groupBox1.Controls.Add(Me.numericUpDown1)
Me.groupBox1.Location = New System.Drawing.Point(280, 326)
Me.groupBox1.Name = "groupBox1"
Me.groupBox1.Size = New System.Drawing.Size(200, 70)
Me.groupBox1.TabIndex = 13
Me.groupBox1.TabStop = False
Me.groupBox1.Text = "Threshold Value"
'
' groupBox2
'

Me.groupBox2.Anchor = System.Windows.Forms.AnchorStyles.None
Me.groupBox2.Controls.Add(Me.startButton)
Me.groupBox2.Controls.Add(Me.stopButton)
Me.groupBox2.Location = New System.Drawing.Point(26, 327)
Me.groupBox2.Name = "groupBox2"
Me.groupBox2.Size = New System.Drawing.Size(214, 68)
Me.groupBox2.TabIndex = 14
Me.groupBox2.TabStop = False
Me.groupBox2.Text = "Logging"
'
' tableLayoutPanel1
'

Me.tableLayoutPanel1.ColumnCount = 2
Me.tableLayoutPanel1.ColumnStyles.Add(New
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 50F))
Me.tableLayoutPanel1.ColumnStyles.Add(New
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 50F))
Me.tableLayoutPanel1.Controls.Add(Me.groupBox2, 0, 1)
Me.tableLayoutPanel1.Controls.Add(Me.groupBox1, 1, 1)
Me.tableLayoutPanel1.Controls.Add(Me.attributesDemoControl1, 0, 0)
Me.tableLayoutPanel1.Dock = System.Windows.Forms.DockStyle.Fill
Me.tableLayoutPanel1.Location = New System.Drawing.Point(0, 0)
Me.tableLayoutPanel1.Name = "tableLayoutPanel1"
Me.tableLayoutPanel1.Padding = New System.Windows.Forms.Padding(10)
Me.tableLayoutPanel1.RowCount = 2
Me.tableLayoutPanel1.RowStyles.Add(New
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 80F))
Me.tableLayoutPanel1.RowStyles.Add(New
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 20F))
Me.tableLayoutPanel1.Size = New System.Drawing.Size(514, 411)
Me.tableLayoutPanel1.TabIndex = 15
'
' attributesDemoControl1
'

Me.tableLayoutPanel1.SetColumnSpan(Me.attributesDemoControl1, 2)
Me.attributesDemoControl1.DataMember = ""
Me.attributesDemoControl1.DataSource = Me.bindingSource1
Me.attributesDemoControl1.Dock = System.Windows.Forms.DockStyle.Fill
Me.attributesDemoControl1.Location = New System.Drawing.Point(13, 13)
Me.attributesDemoControl1.Name = "attributesDemoControl1"
Me.attributesDemoControl1.Padding = New System.Windows.Forms.Padding(10)
Me.attributesDemoControl1.Size = New System.Drawing.Size(488, 306)
Me.attributesDemoControl1.TabIndex = 0
Me.attributesDemoControl1.Threshold = 200000F
Me.attributesDemoControl1.TitleText = "TITLE"
'
' Form1
'

Me.BackColor = System.Drawing.SystemColors.Control
Me.ClientSize = New System.Drawing.Size(514, 430)
Me.Controls.Add(tableLayoutPanel1)
Me.Name = "Form1"
Me.Text = "Form1"
 CType(Me.bindingSource1, System.ComponentModel.ISupportInitialize).EndInit()
 CType(Me.performanceCounter1, System.ComponentModel.ISupportInitialize).EndInit()
 CType(Me.timer1, System.ComponentModel.ISupportInitialize).EndInit()
 CType(Me.numericUpDown1, System.ComponentModel.ISupportInitialize).EndInit()
Me.groupBox1.ResumeLayout(False)
Me.groupBox2.ResumeLayout(False)
```

```

Me.GroupBox2.ResumeLayout(False)
Me.tableLayoutPanel1.ResumeLayout(False)
Me.ResumeLayout(false)
Me.PerformLayout()
End Sub
End Class

```

The first code example is the `AttributesDemoControl` implementation. The second code example demonstrates a form that uses the `AttributesDemoControl`.

Class-level Attributes

Some attributes are applied at the class level. The following code example shows the attributes that are commonly applied to a Windows Forms control.

```

// This control demonstrates a simple logging capability.
[ComplexBindingProperties("DataSource", "DataMember")]
[DefaultBindingProperty("TitleText")]
[DefaultEvent("ThresholdExceeded")]
[DefaultProperty("Threshold")]
[HelpKeywordAttribute(typeof(UserControl))]
[ToolboxItem("System.Windows.Forms.Design.AutoSizeToolboxItem, System.Design")]
public class AttributesDemoControl : UserControl
{

```

```

' This control demonstrates a simple logging capability.
<ComplexBindingProperties("DataSource", "DataMember"), _
DefaultBindingProperty("TitleText"), _
DefaultEvent("ThresholdExceeded"), _
DefaultProperty("Threshold"), _
HelpKeywordAttribute(GetType(UserControl)), _
ToolboxItem("System.Windows.Forms.Design.AutoSizeToolboxItem, System.Design")> _
Public Class AttributesDemoControl
    Inherits UserControl

```

TypeConverter Attribute

[TypeConverterAttribute](#) is another commonly used class-level attribute. The following code example shows its use for the `LogEntry` class. This example also shows an implementation of a [TypeConverter](#) for the `LogEntry` type, called `LogEntryTypeConverter`.

```

// This class encapsulates a log entry. It is a parameterized
// type (also known as a template class). The parameter type T
// defines the type of data being logged. For threshold detection
// to work, this type must implement the IComparable interface.
[TypeConverter("LogEntryTypeConverter")]
public class LogEntry<T> where T : IComparable
{
    private T entryValue;
    private DateTime entryTimeValue;

    public LogEntry(
        T value,
        DateTime time)
    {
        this.entryValue = value;
        this.entryTimeValue = time;
    }

    public T Entry
    {
        get

```

```

        {
            return this.entryValue;
        }
    }

    public DateTime EntryTime
    {
        get
        {
            return this.entryTimeValue;
        }
    }

    // This is the TypeConverter for the LogEntry class.
    public class LogEntryTypeConverter : TypeConverter
    {
        public override bool CanConvertFrom(
            ITypeDescriptorContext context,
            Type sourceType)
        {
            if (sourceType == typeof(string))
            {
                return true;
            }

            return base.CanConvertFrom(context, sourceType);
        }

        public override object ConvertFrom(
            ITypeDescriptorContext context,
            System.Globalization.CultureInfo culture,
            object value)
        {
            if (value is string)
            {
                string[] v = ((string)value).Split(new char[] { '|' });

                Type paramType = typeof(T);
                T entryValue = (T)paramType.InvokeMember(
                    "Parse",
                    BindingFlags.Static | BindingFlags.Public | BindingFlags.InvokeMethod,
                    null,
                    null,
                    new string[] { v[0] },
                    culture);

                return new LogEntry<T>(
                    entryValue,
                    DateTime.Parse(v[2]));
            }

            return base.ConvertFrom(context, culture, value);
        }

        public override object ConvertTo(
            ITypeDescriptorContext context,
            System.Globalization.CultureInfo culture,
            object value,
            Type destinationType)
        {
            if (destinationType == typeof(string))
            {
                LogEntry<T> le = value as LogEntry<T>;

                string stringRepresentation =
                    String.Format("{0} | {1}",
                    le.Entry,
                    le.EntryTime);
            }
        }
    }
}

```

```

        return stringRepresentation;
    }

    return base.ConvertTo(context, culture, value, destinationType);
}
}
}

```

```

' This class encapsulates a log entry. It is a parameterized
' type (also known as a template class). The parameter type T
' defines the type of data being logged. For threshold detection
' to work, this type must implement the IComparable interface.
<TypeConverter("LogEntryTypeConverter")> _
Public Class LogEntry(Of T As IComparable)

    Private entryValue As T

    Private entryTimeValue As DateTime

    Public Sub New(ByVal value As T, ByVal time As DateTime)
        Me.entryValue = value
        Me.entryTimeValue = time
    End Sub

    Public ReadOnly Property Entry() As T
        Get
            Return Me.entryValue
        End Get
    End Property

    Public ReadOnly Property EntryTime() As DateTime
        Get
            Return Me.entryTimeValue
        End Get
    End Property

    ' This is the TypeConverter for the LogEntry class.
    Public Class LogEntryTypeConverter
        Inherits TypeConverter

        Public Overrides Function CanConvertFrom( _
            ByVal context As ITypeDescriptorContext, _
            ByVal sourceType As Type) As Boolean
            If sourceType Is GetType(String) Then
                Return True
            End If

            Return MyBase.CanConvertFrom(context, sourceType)
        End Function

        Public Overrides Function ConvertFrom( _
            ByVal context As ITypeDescriptorContext, _
            ByVal culture As System.Globalization.CultureInfo, _
            ByVal value As Object) As Object
            If TypeOf value Is String Then
                Dim v As String() = CStr(value).Split(New Char() {"|"})
                Dim paramType As Type = GetType(T)
                Dim entryValue As T = CType(paramType.InvokeMember("Parse", BindingFlags.Static Or
BindingFlags.Public Or BindingFlags.InvokeMethod, Nothing, Nothing, New String() {v(0)}), culture), T)

                Return New LogEntry(Of T)(entryValue, DateTime.Parse(v(2))) '
            End If
        End Function
    End Class
End Class

```

```

        Return MyBase.ConvertFrom(context, culture, value)
    End Function

    Public Overrides Function ConvertTo(ByVal context As ITypeDescriptorContext, ByVal culture As
System.Globalization.CultureInfo, ByVal value As Object, ByVal destinationType As Type) As Object
        If destinationType Is GetType(String) Then

            Dim le As LogEntry(Of T) = CType(value, LogEntry(Of T))

            Dim stringRepresentation As String = String.Format("{0} | {1}", le.Entry, le.EntryTime)

            Return stringRepresentation
        End If

        Return MyBase.ConvertTo(context, culture, value, destinationType)
    End Function
End Class
End Class

```

Member-level Attributes

Some attributes are applied at the member level. The following code examples show some attributes that are commonly applied to properties of Windows Forms controls.

```

[Category("Appearance")]
[Description("The title of the log data.")]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)]
[Localizable(true)]
[HelpKeywordAttribute("AttributesDemoControlLibrary.AttributesDemoControl.TitleText")]
public string TitleText
{
    get
    {
        return this.label1.Text;
    }

    set
    {
        this.label1.Text = value;
    }
}

```

```

<Category("Appearance"), _
Description("The title of the log data."), _
DesignerSerializationVisibility(DesignerSerializationVisibility.Visible), Localizable(True), _
HelpKeywordAttribute("AttributesDemoControlLibrary.AttributesDemoControl.TitleText")> _
Public Property TitleText() As String
    Get
        Return Me.label1.Text
    End Get

    Set(ByVal value As String)
        Me.label1.Text = value
    End Set
End Property

```

AmbientValue Attribute

The following example demonstrates the [AmbientValueAttribute](#) and shows code that supports its interaction with the design environment. This interaction is called *ambience*.

```
[AmbientValue(typeof(Color), "Empty")]
[Category("Appearance")]
[DefaultValue(typeof(Color), "White")]
[Description("The color used for painting alert text.")]
public Color AlertForeColor
{
    get
    {
        if (this.alertForeColorValue == Color.Empty &&
            this.Parent != null)
        {
            return Parent.ForeColor;
        }

        return this.alertForeColorValue;
    }

    set
    {
        this.alertForeColorValue = value;
    }
}

// This method is used by designers to enable resetting the
// property to its default value.
public void ResetAlertForeColor()
{
    this.AlertForeColor = AttributesDemoControl.defaultAlertForeColorValue;
}

// This method indicates to designers whether the property
// value is different from the ambient value, in which case
// the designer should persist the value.
private bool ShouldSerializeAlertForeColor()
{
    return (this.alertForeColorValue != AttributesDemoControl.ambientColorValue);
}
```

```

<AmbientValue(GetType(Color), "Empty"), _
Category("Appearance"), _
DefaultValue(GetType(Color), "White"), _
Description("The color used for painting alert text.")> _
Public Property AlertForeColor() As Color
    Get
        If Me.alertForeColorValue = Color.Empty AndAlso (Me.Parent IsNot Nothing) Then
            Return Parent.ForeColor
        End If

        Return Me.alertForeColorValue
    End Get

    Set(ByVal value As Color)
        Me.alertForeColorValue = value
    End Set
End Property

' This method is used by designers to enable resetting the
' property to its default value.
Public Sub ResetAlertForeColor()
    Me.AlertForeColor = AttributesDemoControl.defaultAlertForeColorValue
End Sub

' This method indicates to designers whether the property
' value is different from the ambient value, in which case
' the designer should persist the value.
Private Function ShouldSerializeAlertForeColor() As Boolean
    Return Me.alertForeColorValue <> AttributesDemoControl.ambientColorValue
End Function

```

Databinding Attributes

The following examples demonstrate an implementation of complex data binding. The class-level [ComplexBindingPropertiesAttribute](#), shown previously, specifies the `DataSource` and `DataMember` properties to use for data binding. The [AttributeProviderAttribute](#) specifies the type to which the `DataSource` property will bind.

```

[Category("Data")]
[Description("Indicates the source of data for the control.")]
[RefreshProperties(RefreshProperties.Repaint)]
[AttributeProvider(typeof(IListSource))]
public object DataSource
{
    get
    {
        return this.dataGridView1.DataSource;
    }

    set
    {
        this.dataGridView1.DataSource = value;
    }
}

```

```

<Category("Data"), _
Description("Indicates the source of data for the control."), _
RefreshProperties(RefreshProperties.Repaint), _
AttributeProvider(GetType(IListSource))> _
Public Property DataSource() As Object
    Get
        Return Me.dataGridView1.DataSource
    End Get

    Set(ByVal value As Object)
        Me.dataGridView1.DataSource = value
    End Set
End Property

```

```

[Category("Data")]
[Description("Indicates a sub-list of the data source to show in the control.")]
public string DataMember
{
    get
    {
        return this.dataGridView1.DataMember;
    }

    set
    {
        this.dataGridView1.DataMember = value;
    }
}

```

```

<Category("Data"), _
Description("Indicates a sub-list of the data source to show in the control.")> _
Public Property DataMember() As String
    Get
        Return Me.dataGridView1.DataMember
    End Get

    Set(ByVal value As String)
        Me.dataGridView1.DataMember = value
    End Set
End Property

```

Compiling the Code

- The form that hosts the `AttributesDemoControl` requires a reference to the `AttributesDemoControl` assembly in order to build.

See Also

[IComparable](#)
[DataGridView](#)

[Developing Custom Windows Forms Controls with the .NET Framework](#)

[Attributes in Windows Forms Controls](#)

[How to: Serialize Collections of Standard Types with the DesignerSerializationVisibilityAttribute](#)

Custom Control Painting and Rendering

5/4/2018 • 1 min to read • [Edit Online](#)

Custom painting of controls is one of the many complicated tasks made easy by the .NET Framework. When authoring a custom control, you have many options regarding your control's graphical appearance. If you are authoring a control that inherits from the `Control`, you must provide code that allows your control to render its graphical representation. If you are creating a user control by inheriting from the `UserControl`, or are inheriting from one of the Windows Forms controls, you may override the standard graphical representation and provide your own graphics code. If you want to provide custom rendering for the constituent controls of a `UserControl` you are authoring, your options become more limited, but still allow a wide range of graphical possibilities for your controls and applications.

In This Section

[Rendering a Windows Forms Control](#)

Shows how to program the logic that displays a control.

[User-Drawn Controls](#)

Gives an overview of the steps involved in writing and overriding rendering code for your control.

[Constituent Controls](#)

Describes how to implement custom rendering code for constituent controls in your user controls and forms.

[How to: Make Your Control Invisible at Run Time](#)

Shows how to use the `Visible` property to hide and show a control.

[How to: Give Your Control a Transparent Background](#)

Shows how to use the `SetStyle` method to create a background color that is opaque, transparent, or partially transparent.

[Rendering Controls with Visual Styles](#)

Shows how to render controls using visual styles in operating systems that support them.

Reference

[Control](#)

Describes this class and has links to all of its members.

[UserControl](#)

Describes this class and has links to all of its members.

[OnPaint](#)

Describes this method.

Related Sections

[How to: Create Graphics Objects for Drawing](#)

Introduces GDI+ graphics functionality from a Visual Studio perspective and gives links to more information.

[Varieties of Custom Controls](#)

Describes the kinds of custom controls you can author.

Rendering a Windows Forms Control

5/4/2018 • 5 min to read • [Edit Online](#)

Rendering refers to the process of creating a visual representation on a user's screen. Windows Forms uses GDI (the new Windows graphics library) for rendering. The managed classes that provide access to GDI are in the [System.Drawing](#) namespace and its subnamespaces.

The following elements are involved in control rendering:

- The drawing functionality provided by the base class [System.Windows.Forms.Control](#).
- The essential elements of the GDI graphics library.
- The geometry of the drawing region.
- The procedure for freeing graphics resources.

Drawing Functionality Provided by Control

The base class [Control](#) provides drawing functionality through its [Paint](#) event. A control raises the [Paint](#) event whenever it needs to update its display. For more information about events in the .NET Framework, see [Handling and Raising Events](#).

The event data class for the [Paint](#) event, [PaintEventArgs](#), holds the data needed for drawing a control — a handle to a graphics object and a rectangle object that represents the region to draw in. These objects are shown in bold in the following code fragment.

```
Public Class PaintEventArgs
    Inherits EventArgs
    Implements IDisposable

    Public ReadOnly Property ClipRectangle() As System.Drawing.Rectangle
        ...
    End Property

    Public ReadOnly Property Graphics() As System.Drawing.Graphics
        ...
    End Property
    ' Other properties and methods.
    ...
End Class
```

```
public class PaintEventArgs : EventArgs, IDisposable {
    public System.Drawing.Rectangle ClipRectangle {get;}
    public System.Drawing.Graphics Graphics {get;}
    // Other properties and methods.
    ...
}
```

[Graphics](#) is a managed class that encapsulates drawing functionality, as described in the discussion of GDI later in this topic. The [ClipRectangle](#) is an instance of the [Rectangle](#) structure and defines the available area in which a control can draw. A control developer can compute the [ClipRectangle](#) using the [ClipRectangle](#) property of a control, as described in the discussion of geometry later in this topic.

A control must provide rendering logic by overriding the [OnPaint](#) method that it inherits from [Control](#). [OnPaint](#)

gets access to a graphics object and a rectangle to draw in through the [Graphics](#) and the [ClipRectangle](#) properties of the [PaintEventArgs](#) instance passed to it.

```
Protected Overridable Sub OnPaint(pe As PaintEventArgs)
```

```
protected virtual void OnPaint(PaintEventArgs pe);
```

The [OnPaint](#) method of the base [Control](#) class does not implement any drawing functionality but merely invokes the event delegates that are registered with the [Paint](#) event. When you override [OnPaint](#), you should typically invoke the [OnPaint](#) method of the base class so that registered delegates receive the [Paint](#) event. However, controls that paint their entire surface should not invoke the base class's [OnPaint](#), as this introduces flicker. For an example of overriding the [OnPaint](#) event, see the [How to: Create a Windows Forms Control That Shows Progress](#).

NOTE

Do not invoke [OnPaint](#) directly from your control; instead, invoke the [Invalidate](#) method (inherited from [Control](#)) or some other method that invokes [Invalidate](#). The [Invalidate](#) method in turn invokes [OnPaint](#). The [Invalidate](#) method is overloaded, and, depending on the arguments supplied to [Invalidate](#) `e`, a control redraws either some or all of its screen area.

The base [Control](#) class defines another method that is useful for drawing — the [OnPaintBackground](#) method.

```
Protected Overridable Sub OnPaintBackground(pevent As PaintEventArgs)
```

```
protected virtual void OnPaintBackground(PaintEventArgs pevent);
```

[OnPaintBackground](#) paints the background (and thereby the shape) of the window and is guaranteed to be fast, while [OnPaint](#) paints the details and might be slower because individual paint requests are combined into one [Paint](#) event that covers all areas that have to be redrawn. You might want to invoke the [OnPaintBackground](#) if, for instance, you want to draw a gradient-colored background for your control.

While [OnPaintBackground](#) has an event-like nomenclature and takes the same argument as the `onPaint` method, [OnPaintBackground](#) is not a true event method. There is no [PaintBackground](#) event and [OnPaintBackground](#) does not invoke event delegates. When overriding the [OnPaintBackground](#) method, a derived class is not required to invoke the [OnPaintBackground](#) method of its base class.

GDI+ Basics

The [Graphics](#) class provides methods for drawing various shapes such as circles, triangles, arcs, and ellipses, as well as methods for displaying text. The [System.Drawing](#) namespace and its subnamespaces contain classes that encapsulate graphics elements such as shapes (circles, rectangles, arcs, and others), colors, fonts, brushes, and so on. For more information about GDI, see [Using Managed Graphics Classes](#). The essentials of GDI are also described in the [How to: Create a Windows Forms Control That Shows Progress](#).

Geometry of the Drawing Region

The [ClientRect](#) property of a control specifies the rectangular region available to the control on the user's screen, while the [ClipRectangle](#) property of [PaintEventArgs](#) specifies the area that is actually painted. (Remember that painting is done in the [Paint](#) event method that takes a [PaintEventArgs](#) instance as its argument). A control might need to paint only a portion of its available area, as is the case when a small section of the control's display changes. In those situations, a control developer must compute the actual rectangle to draw in and pass that to

Invalidate. The overloaded versions of **Invalidate** that take a **Rectangle** or **Region** as an argument use that argument to generate the **ClipRectangle** property of **PaintEventArgs**.

The following code fragment shows how the **FlashTrackBar** custom control computes the rectangular area to draw in. The **client** variable denotes the **ClipRectangle** property. For a complete sample, see [How to: Create a Windows Forms Control That Shows Progress](#).

```
Rectangle invalid = new Rectangle(
    client.X + min,
    client.Y,
    max - min,
    client.Height);

Invalidate(invalid);
```

```
Dim invalid As Rectangle = New Rectangle( _
    client.X + lmin, _
    client.Y, _
    lmax - lmin, _
    client.Height)

Invalidate(invalid)
```

Freeing Graphics Resources

Graphics objects are expensive because they use system resources. Such objects include instances of the **System.Drawing.Graphics** class as well as instances of **System.Drawing.Brush**, **System.Drawing.Pen**, and other graphics classes. It is important that you create a graphics resource only when you need it and release it soon as you are finished using it. If you create a type that implements the **IDisposable** interface, call its **Dispose** method when you are finished with it in order to free resources.

The following code fragment shows how the **FlashTrackBar** custom control creates and releases a **Brush** resource. For the complete source code, see [How to: Create a Windows Forms Control That Shows Progress](#).

```
private Brush baseBackground = null;
```

```
Private baseBackground As Brush
```

```
base.OnPaint(e);
if (baseBackground == null) {
    if (showGradient) {
        baseBackground = new LinearGradientBrush(new Point(0, 0),
                                                new Point(ClientSize.Width, 0),
                                                StartColor,
                                                EndColor);
    }
    else if (BackgroundImage != null) {
        baseBackground = new TextureBrush(BackgroundImage);
    }
    else {
        baseBackground = new SolidBrush(BackColor);
    }
}
```

```
 MyBase.OnPaint(e)

If (baseBackground Is Nothing) Then

    If (myShowGradient) Then
        baseBackground = New LinearGradientBrush(New Point(0, 0), _
                                                New Point(ClientSize.Width, 0), _
                                                StartColor, _
                                                EndColor)
    ElseIf (BackgroundImage IsNot Nothing) Then
        baseBackground = New TextureBrush(BackgroundImage)
    Else
        baseBackground = New SolidBrush(BackColor)
    End If

End If
```

```
protected override void OnResize(EventArgs e) {
    base.OnResize(e);
    if (baseBackground != null) {
        baseBackground.Dispose();
        baseBackground = null;
    }
}
```

```
Protected Overrides Sub OnResize(ByVal e As EventArgs)
    MyBase.OnResize(e)
    If (baseBackground IsNot Nothing) Then
        baseBackground.Dispose()
        baseBackground = Nothing
    End If
End Sub
```

See Also

[How to: Create a Windows Forms Control That Shows Progress](#)

User-Drawn Controls

5/4/2018 • 3 min to read • [Edit Online](#)

The .NET Framework provides you with the ability to easily develop your own controls. You can create a user control, which is a set of standard controls bound together by code, or you can design your own control from the ground up. You can even use inheritance to create a control that inherits from an existing control and add to its inherent functionality. Whatever approach you use, the .NET Framework provides the functionality to draw a custom graphical interface for any control you create.

Painting of a control is accomplished by the execution of code in the control's [OnPaint](#) method. The single argument of the [OnPaint](#) method is a [PaintEventArgs](#) object that provides all of the information and functionality required to render your control. The [PaintEventArgs](#) provides as properties two principal objects that will be used in the rendering of your control:

- [ClipRectangle](#) object - the rectangle that represents the part of the control that will be drawn. This can be the entire control, or part of the control depending on how the control is drawn.
- [Graphics](#) object - encapsulates several graphics-oriented objects and methods that provide the functionality necessary to draw your control.

For more information on the [Graphics](#) object and how to use it, see [How to: Create Graphics Objects for Drawing](#).

The [OnPaint](#) event is fired whenever the control is drawn or refreshed on the screen, and the [ClipRectangle](#) object represents the rectangle in which painting will take place. If the entire control needs to be refreshed, the [ClipRectangle](#) will represent the size of the entire control. If only part of the control needs to be refreshed, however, the [ClipRectangle](#) object will represent only the region that needs to be redrawn. An example of such a case would be when a control was partially obscured by another control or form in the user interface.

When inheriting from the [Control](#) class, you must override the [OnPaint](#) method and provide graphics-rendering code within. If you want to provide a custom graphical interface to a user control or an inherited control, you can also do so by overriding the [OnPaint](#) method. An example is shown below:

```
Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
    ' Call the OnPaint method of the base class.
    MyBase.OnPaint(e)

    ' Declare and instantiate a drawing pen.
    Using myPen As System.Drawing.Pen = New System.Drawing.Pen(Color.Aqua)
        ' Draw an aqua rectangle in the rectangle represented by the control.
        e.Graphics.DrawRectangle(myPen, New Rectangle(Me.Location, Me.Size))
    End Using
End Sub
```

```
protected override void OnPaint(PaintEventArgs e)
{
    // Call the OnPaint method of the base class.
    base.OnPaint(e);

    // Declare and instantiate a new pen.
    using (System.Drawing.Pen myPen = new System.Drawing.Pen(Color.Aqua))
    {
        // Draw an aqua rectangle in the rectangle represented by the control.
        e.Graphics.DrawRectangle(myPen, new Rectangle(this.Location,
            this.Size));
    }
}
```

The preceding example demonstrates how to render a control with a very simple graphical representation. It calls the [OnPaint](#) method of the base class, it creates a [Pen](#) object with which to draw, and finally draws an ellipse in the rectangle determined by the [Location](#) and [Size](#) of the control. Although most rendering code will be significantly more complicated than this, this example demonstrates the use of the [Graphics](#) object contained within the [PaintEventArgs](#) object. Note that if you are inheriting from a class that already has a graphical representation, such as [UserControl](#) or [Button](#), and you do not wish to incorporate that representation into your rendering, you should not call your base class's [OnPaint](#) method.

The code in the [OnPaint](#) method of your control will execute when the control is first drawn, and whenever it is refreshed. To ensure that your control is redrawn every time it is resized, add the following line to the constructor of your control:

```
SetStyle(ControlStyles.ResizeRedraw, True)
```

```
SetStyle(ControlStyles.ResizeRedraw, true);
```

NOTE

Use the [Control.Region](#) property to implement a non-rectangular control.

See Also

[Region](#)

[ControlStyles](#)

[Graphics](#)

[OnPaint](#)

[PaintEventArgs](#)

[How to: Create Graphics Objects for Drawing](#)

[Constituent Controls](#)

[Varieties of Custom Controls](#)

Constituent Controls

5/4/2018 • 1 min to read • [Edit Online](#)

The controls that make up a user control, or *constituent controls* as they are termed, are relatively inflexible when it comes to custom graphics rendering. All Windows Forms controls handle their own rendering through their own `OnPaint` method. Because this method is protected, it is not accessible to the developer, and thus cannot be prevented from executing when the control is painted. This does not mean, however, that you cannot add code to affect the appearance of constituent controls. Additional rendering can be accomplished by adding an event handler. For example, suppose you were authoring a `UserControl` with a button named `MyButton`. If you wished to have additional rendering beyond what was provided by the `Button`, you would add code to your user control similar to the following:

```
Public Sub MyPaint(ByVal sender as Object, e as PaintEventArgs) Handles _
    MyButton.Paint
    'Additional rendering code goes here
End Sub
```

```
// Add the event handler to the button's Paint event.
MyButton.Paint += 
    new System.Windows.Forms.PaintEventHandler (this.MyPaint);
// Create the custom painting method.
protected void MyPaint (object sender,
System.Windows.Forms.PaintEventArgs e)
{
    // Additional rendering code goes here.
}
```

NOTE

Some Windows Forms controls, such as `TextBox`, are painted directly by Windows. In these instances, the `OnPaint` method is never called, and thus the above example will never be called.

This creates a method that executes every time the `MyButton.Paint` event executes, thereby adding additional graphical representation to your control. Note that this does not prevent the execution of `MyButton.OnPaint`, and thus all of the painting usually performed by a button will still be performed in addition to your custom painting. For details about GDI+ technology and custom rendering, see the [Creating Graphical Images with GDI+](#). If you wish to have a unique representation of your control, your best course of action is to create an inherited control, and to write custom rendering code for it. For details, see [User-Drawn Controls](#).

See Also

[UserController](#)
[OnPaint](#)
[User-Drawn Controls](#)
[How to: Create Graphics Objects for Drawing](#)
[Varieties of Custom Controls](#)

How to: Make Your Control Invisible at Run Time

5/4/2018 • 1 min to read • [Edit Online](#)

There are times when you might want to create a user control that is invisible at run time. For example, a control that is an alarm clock might be invisible except when the alarm was sounding. This is easily accomplished by setting the `Visible` property. If the `Visible` property is `true`, your control will appear as normal. If `false`, your control will be hidden. Although code in your control may still run while invisible, you will not be able to interact with the control through the user interface. If you want to create an invisible control that still responds to user input (for example, mouse clicks), you should create a transparent control. For more information, see [Giving Your Control a Transparent Background](#).

To make your control invisible at run time

1. Set the `Visible` property to `false`.

```
' To set the Visible property from within your object's own code.  
Me.Visible = False  
' To set the Visible property from another object.  
myControl1.Visible = False
```

```
// To set the Visible property from within your object's own code.  
this.Visible = false;  
// To set the Visible property from another object.  
myControl1.Visible = false;
```

See Also

[Visible](#)

[Developing Custom Windows Forms Controls with the .NET Framework](#)

[How to: Give Your Control a Transparent Background](#)

How to: Give Your Control a Transparent Background

5/4/2018 • 1 min to read • [Edit Online](#)

In earlier versions of the .NET Framework, controls didn't support setting transparent backcolors without first setting the [SetStyle](#) method in the forms's constructor. In the current framework version, the backcolor for most controls can be set to [Transparent](#) in the **Properties** window at design time, or in code in the form's constructor.

NOTE

Windows Forms controls do not support true transparency. The background of a transparent Windows Forms control is painted by its parent.

NOTE

The [Button](#) control doesn't support a transparent backcolor even when the [BackColor](#) property is set to [Transparent](#).

To give your control a transparent backcolor

- In the Properties window, choose the [BackColor](#) property and set it to [Transparent](#)

See Also

[FromArgb](#)

[Developing Custom Windows Forms Controls with the .NET Framework](#)

[Using Managed Graphics Classes](#)

[How to: Draw Opaque and Semitransparent Lines](#)

Rendering Controls with Visual Styles

5/4/2018 • 3 min to read • [Edit Online](#)

The .NET Framework provides support for rendering controls and other Windows user interface (UI) elements using visual styles in operating systems that support them. This topic discusses the several levels of support in the .NET Framework for rendering controls and other UI elements with the current visual style of the operating system.

Rendering Classes for Common Controls

Rendering a control refers to drawing the user interface of a control. The [System.Windows.Forms](#) namespace provides the [ControlPaint](#) class for rendering some common Windows Forms controls. However, this class draws controls in the classic Windows style, which can make it difficult to maintain a consistent UI experience when drawing custom controls in applications with visual styles enabled.

The .NET Framework 2.0 includes classes in the [System.Windows.Forms](#) namespace that render the parts and states of common controls with visual styles. Each of these classes includes `static` methods for drawing the control or parts of the control in a particular state with the current visual style of the operating system.

Some of these classes are designed to draw the related control regardless of whether visual styles are available. If visual styles are enabled, then the class members will draw the related control with visual styles; if visual styles are disabled, then the class members will draw the control in the classic Windows style. These classes include:

- [ButtonRenderer](#)
- [CheckBoxRenderer](#)
- [GroupBoxRenderer](#)
- [RadioButtonRenderer](#)

Other classes can only draw the related control when visual styles are available, and their members will throw an exception if visual styles are disabled. These classes include:

- [ComboBoxRenderer](#)
- [ProgressBarRenderer](#)
- [ScrollBarRenderer](#)
- [TabRenderer](#)
- [TextBoxRenderer](#)
- [TrackBarRenderer](#)

For more information on using these classes to draw a control, see [How to: Use a Control Rendering Class](#).

Visual Style Element and Rendering Classes

The [System.Windows.Forms.VisualStyles](#) namespace includes classes that can be used to draw and get information about any control or UI element that is supported by visual styles. Supported controls include common controls that have a rendering class in the [System.Windows.Forms](#) namespace (see the previous section), as well as other controls, such as tab controls and rebar controls. Other supported UI elements include the parts of the **Start** menu, the taskbar, and the nonclient area of windows.

The main classes of the [System.Windows.Forms.VisualStyles](#) namespace are [VisualStyleElement](#) and [VisualStyleRenderer](#). [VisualStyleElement](#) is a foundation class for identifying any control or user interface element supported by visual styles. In addition to [VisualStyleElement](#) itself, the [System.Windows.Forms.VisualStyles](#) namespace includes many nested classes of [VisualStyleElement](#) with `static` properties that return a [VisualStyleElement](#) for every state of a control, control part, or other UI element supported by visual styles.

[VisualStyleRenderer](#) provides the methods that draw and get information about each [VisualStyleElement](#) defined by the current visual style of the operating system. Information that can be retrieved about an element includes its default size, background type, and color definitions. [VisualStyleRenderer](#) wraps the functionality of the visual styles (UxTheme) API from the Windows Shell portion of the Windows Platform SDK. For more information, see [Using Windows XP Visual Styles](#).

For more information about using [VisualStyleRenderer](#) and [VisualStyleElement](#), see [How to: Render a Visual Style Element](#).

Enabling Visual Styles

To enable visual styles for an application written for the .NET Framework version 1.0, programmers must include an application manifest that specifies that ComCtl32.dll version 6 or later will be used to draw controls.

Applications built with the .NET Framework version 1.1 or later can use the [Application.EnableVisualStyles](#) method of the [Application](#) class.

Checking for Visual Styles Support

The [RenderWithVisualStyles](#) property of the [Application](#) class indicates whether the current application is drawing controls with visual styles. When painting a custom control, you can check the value of [RenderWithVisualStyles](#) to determine whether you should render your control with or without visual styles. The following table lists the four conditions that must exist for [RenderWithVisualStyles](#) to return `true`.

CONDITION	NOTES
The operating system supports visual styles.	To verify this condition separately, use the IsSupportedByOS property of the VisualStyleInformation class.
The user has enabled visual styles in the operating system.	To verify this condition separately, use the IsEnabledByUser property of the VisualStyleInformation class.
Visual styles are enabled in the application.	Visual styles can be enabled in an application by calling the Application.EnableVisualStyles method or by using an application manifest that specifies that ComCtl32.dll version 6 or later will be used to draw controls.
Visual styles are being used to draw the client area of application windows.	To verify this condition separately, use the VisualStyleState property of the Application class and verify that it has the value VisualStyleState.ClientAreaEnabled or VisualStyleState.ClientAndNonClientAreasEnabled .

To determine when a user enables or disables visual styles, or switches from one visual style to another, check for the [UserPreferenceCategoryVisualStyle](#) value in the handlers for the [SystemEvents.UserPreferenceChanging](#) or [SystemEvents.UserPreferenceChanged](#) events.

IMPORTANT

If you want to use [VisualStyleRenderer](#) to render a control or UI element when the user enables or switches visual styles, make sure that you do this when handling the [UserPreferenceChanged](#) event instead of the [UserPreferenceChanging](#) event. An exception will be thrown if you use the [VisualStyleRenderer](#) class when handling [UserPreferenceChanging](#).

See Also

[Custom Control Painting and Rendering](#)

How to: Use a Control Rendering Class

5/4/2018 • 1 min to read • [Edit Online](#)

This example demonstrates how to use the [ComboBoxRenderer](#) class to render the drop-down arrow of a combo box control. The example consists of the [OnPaint](#) method of a simple custom control. The [ComboBoxRenderer::IsSupported](#) property is used to determine whether visual styles are enabled in the client area of application windows. If visual styles are active, then the [ComboBoxRenderer::DrawDropDownButton](#) method will render the drop-down arrow with visual styles; otherwise, the [ControlPaint::DrawComboButton](#) method will render the drop-down arrow in the classic Windows style.

Example

```
// Render the drop-down arrow with or without visual styles.  
protected:  
    virtual void OnPaint(PaintEventArgs^ e) override  
    {  
        __super::OnPaint(e);  
  
        if (!ComboBoxRenderer::IsSupported)  
        {  
            ControlPaint::DrawComboButton(e->Graphics,  
                this->ClientRectangle, ButtonState::Normal);  
        }  
        else  
        {  
            ComboBoxRenderer::DrawDropDownButton(e->Graphics,  
                this->ClientRectangle, ComboBoxState::Normal);  
        }  
    }  
}
```

```
// Render the drop-down arrow with or without visual styles.  
protected override void OnPaint(PaintEventArgs e)  
{  
    base.OnPaint(e);  
  
    if (!ComboBoxRenderer::IsSupported)  
    {  
        ControlPaint::DrawComboButton(e.Graphics,  
            this.ClientRectangle, ButtonState.Normal);  
    }  
    else  
    {  
        ComboBoxRenderer::DrawDropDownButton(e.Graphics,  
            this.ClientRectangle, ComboBoxState.Normal);  
    }  
}
```

```
' Render the drop-down arrow with or without visual styles.  
Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)  
    MyBase.OnPaint(e)  
  
    If Not ComboBoxRenderer.IsSupported Then  
        ControlPaint.DrawComboButton(e.Graphics, _  
            Me.ClientRectangle, ButtonState.Normal)  
    Else  
        ComboBoxRenderer.DrawDropDownButton(e.Graphics, _  
            Me.ClientRectangle, ComboBoxState.Normal)  
    End If  
End Sub
```

Compiling the Code

This example requires:

- A custom control derived from the [Control](#) class.
- A [Form](#) that hosts the custom control.
- References to the [System](#), [System.Drawing](#), [System.Windows.Forms](#), and [System.Windows.Forms.VisualStyles](#) namespaces.

See Also

[Rendering Controls with Visual Styles](#)

How to: Render a Visual Style Element

5/4/2018 • 2 min to read • [Edit Online](#)

The [System.Windows.Forms.VisualStyles](#) namespace exposes [VisualStyleElement](#) objects that represent the Windows user interface (UI) elements supported by visual styles. This topic demonstrates how to use the [VisualStyleRenderer](#) class to render the [VisualStyleElement](#) that represents the **Log Off** and **Shut Down** buttons of the Start menu.

To render a visual style element

1. Create a [VisualStyleRenderer](#) and set it to the element you want to draw. Note the use of the [Application.RenderWithVisualStyles](#) property and the [VisualStyleRenderer.IsElementDefined](#) method; the [VisualStyleRenderer](#) constructor will throw an exception if visual styles are disabled or an element is undefined.

```
private:  
    VisualStyleRenderer^ renderer;  
    VisualStyleElement^ element;  
  
public:  
    CustomControl()  
    {  
        this->Location = Point(50, 50);  
        this->Size = System::Drawing::Size(200, 200);  
        this->BackColor = SystemColors::ActiveBorder;  
        this->element =  
            VisualStyleElement::StartPanel::LogOffButtons::Normal;  
        if (Application::RenderWithVisualStyles &&  
            VisualStyleRenderer::IsElementDefined(element))  
        {  
            renderer = gcnew VisualStyleRenderer(element);  
        }  
    }  
}
```

```
private VisualStyleRenderer renderer = null;  
private readonly VisualStyleElement element =  
    VisualStyleElement.StartPanel.LogOffButtons.Normal;  
  
public CustomControl()  
{  
    this.Location = new Point(50, 50);  
    this.Size = new Size(200, 200);  
    this.BackColor = SystemColors.ActiveBorder;  
  
    if (Application.RenderWithVisualStyles &&  
        VisualStyleRenderer.IsElementDefined(element))  
    {  
        renderer = new VisualStyleRenderer(element);  
    }  
}
```

```

Private renderer As VisualStyleRenderer = Nothing
Private element As VisualStyleElement = _
    VisualStyleElement.StartPanel.LogOffButtons.Normal

Public Sub New()
    Me.Location = New Point(50, 50)
    Me.Size = New Size(200, 200)
    Me.BackColor = SystemColors.ActiveBorder

    If Application.RenderWithVisualStyles And _
        VisualStyleRenderer.IsElementDefined(element) Then
        renderer = New VisualStyleRenderer(element)
    End If
End Sub

```

- Call the [DrawBackground](#) method to render the element that the [VisualStyleRenderer](#) currently represents.

```

protected:
    virtual void OnPaint(PaintEventArgs^ e) override
    {
        // Draw the element if the renderer has been set.
        if (renderer != nullptr)
        {
            renderer->DrawBackground(e->Graphics, this->ClientRectangle);
        }

        // Visual styles are disabled or the element is undefined,
        // so just draw a message.
        else
        {
            this->Text = "Visual styles are disabled.";
            TextRenderer::DrawText(e->Graphics, this->Text, this->Font,
                Point(0, 0), this->ForeColor);
        }
    }
}

```

```

protected override void OnPaint(PaintEventArgs e)
{
    // Draw the element if the renderer has been set.
    if (renderer != null)
    {
        renderer.DrawBackground(e.Graphics, this.ClientRectangle);
    }

    // Visual styles are disabled or the element is undefined,
    // so just draw a message.
    else
    {
        this.Text = "Visual styles are disabled.";
        TextRenderer.DrawText(e.Graphics, this.Text, this.Font,
            new Point(0, 0), this.ForeColor);
    }
}

```

```
Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
    ' Draw the element if the renderer has been set.
    If (renderer IsNot Nothing) Then
        renderer.DrawBackground(e.Graphics, Me.ClientRectangle)

        ' Visual styles are disabled or the element is undefined,
        ' so just draw a message.
    Else
        Me.Text = "Visual styles are disabled."
        TextRenderer.DrawText(e.Graphics, Me.Text, Me.Font, _
            New Point(0, 0), Me.ForeColor)
    End If
End Sub
```

Compiling the Code

This example requires:

- A custom control derived from the [Control](#) class.
- A [Form](#) that hosts the custom control.
- References to the [System](#), [System.Drawing](#), [System.Windows.Forms](#), and [System.Windows.Forms.VisualStyles](#) namespaces.

See Also

[Rendering Controls with Visual Styles](#)

Layout in Windows Forms Controls

5/4/2018 • 1 min to read • [Edit Online](#)

Precise placement of controls on your form is a high priority for many applications. The `System.Windows.Forms` namespace gives you many layout tools to accomplish this.

In This Section

[AutoSize Property Overview](#)

Describes the `AutoSize` property and its role in layout.

[Margin and Padding in Windows Forms Controls](#)

Describes the `Margin` and `Padding` properties and their roles in layout.

[How to: Align a Control to the Edges of Forms](#)

Demonstrates how to use the `Dock` property to align your control to the edge of the form it occupies.

[How to: Create a Border Around a Windows Forms Control Using Padding](#)

Demonstrates how to use the `Padding` property to outline a control.

[How to: Implement a Custom Layout Engine](#)

Demonstrates how to implement a `LayoutEngine` for arranging Windows Forms controls.

Reference

[TableLayoutPanel](#)

Provides reference documentation for the `TableLayoutPanel` control.

[FlowLayoutPanel](#)

Provides reference documentation for the `FlowLayoutPanel` control.

See Also

[How to: Anchor and Dock Child Controls in a FlowLayoutPanel Control](#)

[How to: Anchor and Dock Child Controls in a TableLayoutPanel Control](#)

[How to: Design a Windows Forms Layout that Responds Well to Localization](#)

[AutoSize Behavior in the TableLayoutPanel Control](#)

AutoSize Property Overview

5/4/2018 • 3 min to read • [Edit Online](#)

The [AutoSize](#) property enables a control to change its size, if necessary, to attain the value specified by the [PreferredSize](#) property. You adjust the sizing behavior for specific controls by setting the [AutoSizeMode](#) property.

AutoSize Behavior

Only some controls support the [AutoSize](#) property. In addition, some controls that support the [AutoSize](#) property also support the [AutoSizeMode](#) property.

The [AutoSize](#) property produces somewhat different behavior, depending on the specific control type and the value of the [AutoSizeMode](#) property, if the property exists. The following table describes the behaviors that are always true and provides a brief description of each:

ALWAYS TRUE BEHAVIOR	DESCRIPTION
Automatic sizing is a run-time feature.	This means it never grows or shrinks a control and then has no further effect.
If a control changes size, the value of its Location property always remains constant.	When a control's contents cause it to grow, the control grows toward the right and downward. Controls do not grow to the left.
The Dock and Anchor properties are honored when AutoSize is <code>true</code> .	<p>The value of the control's Location property is adjusted to the correct value.</p> <p>Note The Label control is the exception to this rule. When you set the value of a docked Label control's AutoSize property to <code>true</code>, the Label control will not stretch.</p>
A control's MaximumSize and MinimumSize properties are always honored, regardless of the value of its AutoSize property.	The MaximumSize and MinimumSize properties are not affected by the AutoSize property.
There is no minimum size set by default.	This means that if a control is set to shrink under AutoSize and it has no contents, the value of its Size property is 0,0. In this case, your control will shrink to a point, and it will not be readily visible.
If a control does not implement the GetPreferredSize method, the GetPreferredSize method returns last value assigned to the Size property.	This means that setting AutoSize to <code>true</code> will have no effect.
A control in a TableLayoutPanel cell always shrinks to fit in the cell until its MinimumSize is reached.	This size is enforced as a maximum size. This is not the case when the cell is part of an AutoSize row or column.

AutoSizeMode Property

The [AutoSizeMode](#) property provides more fine-grained control over the default [AutoSize](#) behavior. The [AutoSizeMode](#) property specifies how a control sizes itself to its content. The content, for example, could be the text for a [Button](#) control or the child controls for a container.

The following table shows the [AutoSizeMode](#) settings and a description of the behavior each setting elicits.

AUTOSIZemode SETTING	BEHAVIOR
GrowAndShrink	<p>The control grows or shrinks to encompass its contents.</p> <p>The MinimumSize and MaximumSize values are honored, but the current value of the Size property is ignored.</p> <p>This is the same behavior as controls with the AutoSize property and no AutoSizeMode property.</p>
GrowOnly	<p>The control grows as much as necessary to encompass its contents, but it will not shrink smaller than the value specified by its Size property.</p> <p>This is the default value for AutoSizeMode.</p>

Controls That Support the AutoSize Property

The following table lists the controls that support the [AutoSize](#) and [AutoSizeMode](#) properties.

AUTOSIZE SUPPORT	CONTROL TYPE
<ul style="list-style-type: none"> - AutoSize property supported. - No AutoSizeMode property. 	CheckBox DomainUpDown Label LinkLabel MaskedTextBox (TextBox base) NumericUpDown RadioButton TextBox TrackBar
<ul style="list-style-type: none"> - AutoSize property supported. - AutoSizeMode property supported. 	Button CheckedListBox FlowLayoutPanel Form GroupBox Panel TableLayoutPanel

AUTOSIZE SUPPORT	CONTROL TYPE
- No AutoSize property.	CheckedListBox
	ComboBox
	DataGridView
	DateTimePicker
	ListBox
	ListView
	MaskedTextBox
	MonthCalendar
	ProgressBar
	PropertyGrid
	RichTextBox
	SplitContainer
	TabControl
	TabPage
	TreeView
	WebBrowser
	ScrollBar

AutoSize in the Design Environment

The following table describes the sizing behavior of a control at design time, based on the value of its [AutoSize](#) and [AutoSizeMode](#) properties.

Override the [SelectionRules](#) property to determine whether a given control is in a user-resizable state. In the following table, "cannot" means [Moveable](#) only, "can" means [AllSizeable](#) and [Moveable](#).

AUTOSIZE SETTINGS	DESIGN-TIME SIZING GESTURE
<ul style="list-style-type: none"> - AutoSize = <code>true</code> - No AutoSizeMode property. 	<p>The user cannot resize the control at design time, except for the following controls:</p> <ul style="list-style-type: none"> - TextBox - MaskedTextBox - RichTextBox - TrackBar
<ul style="list-style-type: none"> - AutoSize = <code>true</code> - AutoSizeMode = <code>GrowAndShrink</code> 	<p>The user cannot resize the control at design time.</p>
<ul style="list-style-type: none"> - AutoSize = <code>true</code> - AutoSizeMode = <code>GrowOnly</code> 	<p>The user can resize the control at design time. When the Size property is set, the user can only increase the size of the control.</p>

AUTOSIZE SETTINGS	DESIGN-TIME SIZING GESTURE
- <code>AutoSize = false</code> , or <code>AutoSize</code> property is hidden.	User can resize the control at design time.

NOTE

To maximize productivity, the Windows Forms Designer shadows the `AutoSize` property for the `Form` class. At design time, the form behaves as though the `AutoSize` property is set to `false`, regardless of its actual setting. At runtime, no special accommodation is made, and the `AutoSize` property is applied as specified by the property setting.

See Also

[AutoSize](#)

[PreferredSize](#)

[GetPreferredSize](#)

How to: Align a Control to the Edges of Forms

5/4/2018 • 1 min to read • [Edit Online](#)

You can make your control align to the edge of your forms by setting the [Dock](#) property. This property designates where your control resides in the form. The [Dock](#) property can be set to the following values:

SETTING	EFFECT ON YOUR CONTROL
Bottom	Docks to the bottom of the form.
Fill	Fills all remaining space in the form.
Left	Docks to the left side of the form.
None	Does not dock anywhere, and it appears at the location specified by its Location property.
Right	Docks to the right side of the form.
Top	Docks to the top of the form.

There is design-time support for this feature in Visual Studio.

To set the Dock property for your control at run time

1. Set the [Dock](#) property to the appropriate value in code.

```
' To set the Dock property internally.  
Me.Dock = DockStyle.Top  
' To set the Dock property from another object.  
UserControl1.Dock = DockStyle.Top
```

```
// To set the Dock property internally.  
this.Dock = DockStyle.Top;  
// To set the Dock property from another object.  
UserControl1.Dock = DockStyle.Top;
```

See Also

[Control.Dock](#)

[Control.Anchor](#)

[Developing Custom Windows Forms Controls with the .NET Framework](#)

[How to: Anchor and Dock Child Controls in a FlowLayoutPanel Control](#)

[How to: Anchor and Dock Child Controls in a TableLayoutPanel Control](#)

[AutoSize Property Overview](#)

Margin and Padding in Windows Forms Controls

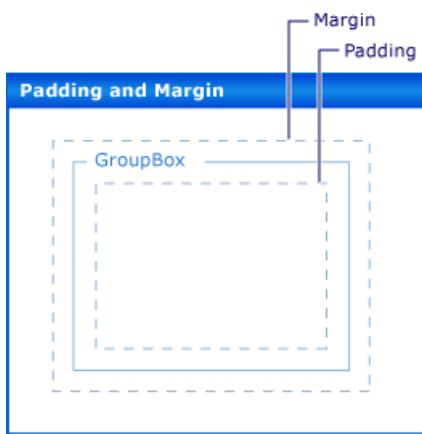
5/4/2018 • 1 min to read • [Edit Online](#)

Precise placement of controls on your form is a high priority for many applications. The [System.Windows.Forms](#) namespace gives you many layout features to accomplish this. Two of the most important are the [Margin](#) and [Padding](#) properties.

The [Margin](#) property defines the space around the control that keeps other controls a specified distance from the control's borders.

The [Padding](#) property defines the space in the interior of a control that keeps the control's content (for example, the value of its [Text](#) property) a specified distance from the control's borders.

The following illustration shows the [Padding](#) and [Margin](#) properties on a control.



There is design-time support for this feature in Visual Studio. Also see [Walkthrough: Laying Out Windows Forms Controls with Padding, Margins, and the AutoSize Property](#).

See Also

[AutoSize](#)
[Margin](#)
[Padding](#)
[LayoutEngine](#)
[TableLayoutPanel](#)
[FlowLayoutPanel](#)

How to: Create a Border Around a Windows Forms Control Using Padding

5/4/2018 • 2 min to read • [Edit Online](#)

The following code example demonstrates how to create a border or outline around a `RichTextBox` control. The example sets the value of a `Panel` control's `Padding` property to 5 and sets the `Dock` property of a child `RichTextBox` control to `Fill`. The `BackColor` of the `Panel` control is set to `Blue`, which creates a blue border around the `RichTextBox` control.

Example

```
using System;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;

namespace MarginAndPadding
{
    public class Form1 : Form
    {
        private Panel panel1;
        private RichTextBox richTextBox1;
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        // This code example demonstrates using the Padding property to
        // create a border around a RichTextBox control.
        public Form1()
        {
            InitializeComponent();

            this.panel1.BackColor = System.Drawing.Color.Blue;
            this.panel1.Padding = new System.Windows.Forms.Padding(5);
            this.panel1.Dock = System.Windows.Forms.DockStyle.Fill;

            this.richTextBox1.BorderStyle = System.Windows.Forms.BorderStyle.None;
            this.richTextBox1.Dock = System.Windows.Forms.DockStyle.Fill;
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
    }
}
```

```

/// </summary>
private void InitializeComponent()
{
    this.panel1 = new System.Windows.Forms.Panel();
    this.richTextBox1 = new System.Windows.Forms.RichTextBox();
    this.panel1.SuspendLayout();
    this.SuspendLayout();
    //
    // panel1
    //
    this.panel1.Controls.Add(this.richTextBox1);
    this.panel1.Location = new System.Drawing.Point(20, 20);
    this.panel1.Name = "panel1";
    this.panel1.Size = new System.Drawing.Size(491, 313);
    this.panel1.TabIndex = 0;
    //
    // richTextBox1
    //
    this.richTextBox1.Location = new System.Drawing.Point(5, 5);
    this.richTextBox1.Name = "richTextBox1";
    this.richTextBox1.Size = new System.Drawing.Size(481, 303);
    this.richTextBox1.TabIndex = 0;
    this.richTextBox1.Text = "";
    //
    // Form1
    //
    this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
    this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
    this.ClientSize = new System.Drawing.Size(531, 353);
    this.Controls.Add(this.panel1);
    this.Name = "Form1";
    this.Padding = new System.Windows.Forms.Padding(20);
    this.Text = "Form1";
    this.panel1.ResumeLayout(false);
    this.ResumeLayout(false);
}

#endregion
}

static class Program
{
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.Run(new Form1());
    }
}
}

```

```

Imports System
Imports System.ComponentModel
Imports System.Drawing
Imports System.Windows.Forms

Public Class Form1
Inherits Form
Private panel1 As Panel
Private richTextBox1 As RichTextBox

```

```

'<summary>
' Required designer variable.
'</summary>
Private components As System.ComponentModel.IContainer = Nothing

' This code example demonstrates using the Padding property to
' create a border around a RichTextBox control.

Public Sub New()
    InitializeComponent()

    Me.panel1.BackColor = System.Drawing.Color.Blue
    Me.panel1.Padding = New System.Windows.Forms.Padding(5)
    Me.panel1.Dock = System.Windows.Forms.DockStyle.Fill

    Me.richTextBox1.BorderStyle = System.Windows.Forms.BorderStyle.None
    Me.richTextBox1.Dock = System.Windows.Forms.DockStyle.Fill
End Sub

'<summary>
' Clean up any resources being used.
'</summary>
'<param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
Protected Overrides Sub Dispose(disposing As Boolean)
    If disposing AndAlso (components IsNot Nothing) Then
        components.Dispose()
    End If
    MyBase.Dispose(disposing)
End Sub

#Region "Windows Form Designer generated code"

'<summary>
' Required method for Designer support - do not modify
' the contents of this method with the code editor.
'</summary>
Private Sub InitializeComponent()
    Me.panel1 = New System.Windows.Forms.Panel()
    Me.richTextBox1 = New System.Windows.Forms.RichTextBox()
    Me.panel1.SuspendLayout()
    Me.SuspendLayout()

    ' panel1
    ' 

    Me.panel1.Controls.Add(Me.richTextBox1)
    Me.panel1.Location = New System.Drawing.Point(20, 20)
    Me.panel1.Name = "panel1"
    Me.panel1.Size = New System.Drawing.Size(491, 313)
    Me.panel1.TabIndex = 0
    ' 

    ' richTextBox1
    ' 

    Me.richTextBox1.Location = New System.Drawing.Point(5, 5)
    Me.richTextBox1.Name = "richTextBox1"
    Me.richTextBox1.Size = New System.Drawing.Size(481, 303)
    Me.richTextBox1.TabIndex = 0
    Me.richTextBox1.Text = ""

    ' Form1
    ' 

    Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0F, 13.0F)
    Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
    Me.ClientSize = New System.Drawing.Size(531, 353)
    Me.Controls.Add(panel1)
    Me.Name = "Form1"
    Me.Padding = New System.Windows.Forms.Padding(20)
    Me.Text = "Form1"
    Me.panel1.ResumeLayout(False)
    Me.ResumeLayout(False)
End Sub

```

```
#End Region

End Class

Public Class Program

'<summary>
'</summary>
'</summary>
<STAThread()> _
Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())
End Sub
End Class
```

See Also

[Padding](#)

[Margin and Padding in Windows Forms Controls](#)

How to: Implement a Custom Layout Engine

5/4/2018 • 4 min to read • [Edit Online](#)

The following code example demonstrates how to create a custom layout engine that performs a simple flow layout. It implements a panel control named `DemoFlowPanel`, which overrides the `LayoutEngine` property to provide an instance of the `DemoFlowLayout` class.

Example

```
#using <System.Drawing.dll>
#using <System.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Collections::Generic;
using namespace System::Drawing;
using namespace System::Text;
using namespace System::Windows::Forms;
using namespace System::Windows::Forms::Layout;

// This class demonstrates a simple custom layout engine.
public ref class DemoFlowLayout : public LayoutEngine
{
public:
    virtual bool Layout(Object^ container,
    LayoutEventArgs^ layoutEventArgs) override
    {
        Control^ parent = nullptr;
        try
        {
            parent = (Control ^) container;
        }
        catch (InvalidCastException^ ex)
        {
            throw gcnew ArgumentException(
                "The parameter 'container' must be a control", "container", ex);
        }
        // Use DisplayRectangle so that parent.Padding is honored.
        Rectangle parentDisplayRectangle = parent->DisplayRectangle;
        Point nextControlLocation = parentDisplayRectangle.Location;

        for each (Control^ currentControl in parent->Controls)
        {
            // Only apply layout to visible controls.
            if (!currentControl->Visible)
            {
                continue;
            }

            // Respect the margin of the control:
            // shift over the left and the top.
            nextControlLocation.Offset(currentControl->Margin.Left,
                currentControl->Margin.Top);

            // Set the location of the control.
            currentControl->Location = nextControlLocation;

            // Set the autosized controls to their
            // autosized heights.
            if (currentControl->AutoSize)
        }
    }
}
```

```

    {
        currentControl->Size = currentControl->GetPreferredSize(
            parentDisplayRectangle.Size);
    }

    // Move X back to the display rectangle origin.
    nextControlLocation.X = parentDisplayRectangle.X;

    // Increment Y by the height of the control
    // and the bottom margin.
    nextControlLocation.Y += currentControl->Height +
        currentControl->Margin.Bottom;
}

// Optional: Return whether or not the container's
// parent should perform layout as a result of this
// layout. Some layout engines return the value of
// the container's AutoSize property.

return false;
}
};

// This class demonstrates a simple custom layout panel.
// It overrides the LayoutEngine property of the Panel
// control to provide a custom layout engine.
public ref class DemoFlowLayout : public Panel
{
private:
    DemoFlowLayout^ layoutEngine;

public:
    DemoFlowLayout()
    {
        layoutEngine = gcnew DemoFlowLayout();
    }

public:
    virtual property System::Windows::Forms::Layout::LayoutEngine^ LayoutEngine
    {
        System::Windows::Forms::Layout::LayoutEngine^ get() override
        {
            if (layoutEngine == nullptr)
            {
                layoutEngine = gcnew DemoFlowLayout();
            }

            return layoutEngine;
        }
    }
};

```

```

using System;
using System.Collections.Generic;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Windows.Forms.Layout;

// This class demonstrates a simple custom layout panel.
// It overrides the LayoutEngine property of the Panel
// control to provide a custom layout engine.
public class DemoFlowLayout : LayoutEngine
{
    private DemoFlowLayout layoutEngine;

    public DemoFlowLayout()
    {
    }
}

```

```

    public DemoFlowLayout()
    {
    }

    public override LayoutEngine LayoutEngine
    {
        get
        {
            if (layoutEngine == null)
            {
                layoutEngine = new DemoFlowLayout();
            }

            return layoutEngine;
        }
    }
}

// This class demonstrates a simple custom layout engine.
public class DemoFlowLayout : LayoutEngine
{
    public override bool Layout(
        object container,
        LayoutEventArgs layoutEventArgs)
    {
        Control parent = container as Control;

        // Use DisplayRectangle so that parent.Padding is honored.
        Rectangle parentDisplayRectangle = parent.DisplayRectangle;
        Point nextControlLocation = parentDisplayRectangle.Location;

        foreach (Control c in parent.Controls)
        {
            // Only apply layout to visible controls.
            if (!c.Visible)
            {
                continue;
            }

            // Respect the margin of the control:
            // shift over the left and the top.
            nextControlLocation.Offset(c.Margin.Left, c.Margin.Top);

            // Set the location of the control.
            c.Location = nextControlLocation;

            // Set the autosized controls to their
            // autosized heights.
            if (c.AutoSize)
            {
                c.Size = c.GetPreferredSize(parentDisplayRectangle.Size);
            }

            // Move X back to the display rectangle origin.
            nextControlLocation.X = parentDisplayRectangle.X;

            // Increment Y by the height of the control
            // and the bottom margin.
            nextControlLocation.Y += c.Height + c.Margin.Bottom;
        }

        // Optional: Return whether or not the container's
        // parent should perform layout as a result of this
        // layout. Some layout engines return the value of
        // the container's AutoSize property.

        return false;
    }
}

```

```

Imports System
Imports System.Collections.Generic
Imports System.Drawing
Imports System.Text
Imports System.Windows.Forms
Imports System.Windows.Forms.Layout

' This class demonstrates a simple custom layout panel.
' It overrides the LayoutEngine property of the Panel
' control to provide a custom layout engine.
Public Class DemoFlowLayoutPanel
    Inherits Panel

    Private layoutEng As DemoFlowLayout

    Public Sub New()
        End Sub

    Public Overrides ReadOnly Property LayoutEngine() As LayoutEngine
        Get
            If layoutEng Is Nothing Then
                layoutEng = New DemoFlowLayout()
            End If

            Return layoutEng
        End Get
    End Property
End Class

' This class demonstrates a simple custom layout engine.
Public Class DemoFlowLayout
    Inherits LayoutEngine

    Public Overrides Function Layout( _
        ByVal container As Object, _
        ByVal layoutEventArgs As LayoutEventArgs) As Boolean

        Dim parent As Control = container

        ' Use DisplayRectangle so that parent.Padding is honored.
        Dim parentDisplayRectangle As Rectangle = parent.DisplayRectangle
        Dim nextControlLocation As Point = parentDisplayRectangle.Location

        Dim c As Control
        For Each c In parent.Controls

            ' Only apply layout to visible controls.
            If c.Visible <> True Then
                Continue For
            End If

            ' Respect the margin of the control:
            ' shift over the left and the top.
            nextControlLocation.Offset(c.Margin.Left, c.Margin.Top)

            ' Set the location of the control.
            c.Location = nextControlLocation

            ' Set the autosized controls to their
            ' autosized heights.
            If c.AutoSize Then
                c.Size = c.GetPreferredSize(parentDisplayRectangle.Size)
            End If

            ' Move X back to the display rectangle origin.
            nextControlLocation.X = parentDisplayRectangle.X
        Next
    End Function
End Class

```

```
' Increment Y by the height of the control
' and the bottom margin.
nextControlLocation.Y += c.Height + c.Margin.Bottom
Next c

' Optional: Return whether or not the container's
' parent should perform layout as a result of this
' layout. Some layout engines return the value of
' the container's AutoSize property.
Return False

End Function
End Class
```

See Also

[LayoutEngine](#)

[Control.LayoutEngine](#)

Multithreading in Windows Forms Controls

5/4/2018 • 1 min to read • [Edit Online](#)

In many applications, you can make your user interface (UI) more responsive by performing time-consuming operations on another thread. A number of tools are available for multithreading your Windows Forms controls, including the [System.Threading](#) namespace, the [Control.BeginInvoke](#) method, and the [BackgroundWorker](#) component.

NOTE

The [BackgroundWorker](#) component replaces and adds functionality to the [System.Threading](#) namespace and the [Control.BeginInvoke](#) method; however, these are retained for both backward compatibility and future use, if you choose. For more information, see [BackgroundWorker Component Overview](#).

In This Section

[How to: Make Thread-Safe Calls to Windows Forms Controls](#)

Shows how to make thread-safe calls to Windows Forms controls.

[How to: Use a Background Thread to Search for Files](#)

Shows how to use the [System.Threading](#) namespace and the [BeginInvoke](#) method to search for files asynchronously.

Reference

[BackgroundWorker](#)

Documents a component that encapsulates a worker thread for asynchronous operations.

[LoadAsync](#)

Documents how to load a sound asynchronously.

[LoadAsync](#)

Documents how to load an image asynchronously.

Related Sections

[How to: Run an Operation in the Background](#)

Shows how to perform a time-consuming operation with the [BackgroundWorker](#) component.

[BackgroundWorker Component Overview](#)

Provides topics that describe how to use the [BackgroundWorker](#) component for asynchronous operations.

How to: Make Thread-Safe Calls to Windows Forms Controls

5/4/2018 • 20 min to read • [Edit Online](#)

If you use multithreading to improve the performance of your Windows Forms applications, you must make sure that you make calls to your controls in a thread-safe way.

Access to Windows Forms controls is not inherently thread safe. If you have two or more threads manipulating the state of a control, it is possible to force the control into an inconsistent state. Other thread-related bugs are possible, such as race conditions and deadlocks. It is important to make sure that access to your controls is performed in a thread-safe way.

It is unsafe to call a control from a thread other than the one that created the control without using the [Invoke](#) method. The following is an example of a call that is not thread safe.

```
// This event handler creates a thread that calls a
// Windows Forms control in an unsafe way.
private void setTextUnsafeBtn_Click(
    object sender,
    EventArgs e)
{
    this.demoThread =
        new Thread(new ThreadStart(this.ThreadProcUnsafe));

    this.demoThread.Start();
}

// This method is executed on the worker thread and makes
// an unsafe call on the TextBox control.
private void ThreadProcUnsafe()
{
    this.textBox1.Text = "This text was set unsafely.";
}
```

```
' This event handler creates a thread that calls a
' Windows Forms control in an unsafe way.
Private Sub setTextUnsafeBtn_Click( _
    ByVal sender As Object, _
    ByVal e As EventArgs) Handles setTextUnsafeBtn.Click

    Me.demoThread = New Thread( _
        New ThreadStart(AddressOf Me.ThreadProcUnsafe))

    Me.demoThread.Start()
End Sub

' This method is executed on the worker thread and makes
' an unsafe call on the TextBox control.
Private Sub ThreadProcUnsafe()
    Me.textBox1.Text = "This text was set unsafely."
End Sub
```

```

// This event handler creates a thread that calls a
// Windows Forms control in an unsafe way.
private:
    void setTextUnsafeBtn_Click(Object^ sender, EventArgs^ e)
    {
        this->demoThread =
            gcnew Thread(gcnew ThreadStart(this,&Form1::ThreadProcUnsafe));

        this->demoThread->Start();
    }

// This method is executed on the worker thread and makes
// an unsafe call on the TextBox control.
private:
    void ThreadProcUnsafe()
    {
        this->textBox1->Text = "This text was set unsafely.";
    }

```

The .NET Framework helps you detect when you are accessing your controls in a manner that is not thread safe. When you are running your application in the debugger, and a thread other than the one which created a control tries to call that control, the debugger raises an [InvalidOperationException](#) with the message, "Control *control name* accessed from a thread other than the thread it was created on."

This exception occurs reliably during debugging and, under some circumstances, at run time. You might see this exception when you debug applications that you wrote with the .NET Framework prior to the .NET Framework 2.0. You are strongly advised to fix this problem when you see it, but you can disable it by setting the [CheckForIllegalCrossThreadCalls](#) property to `false`. This causes your control to run like it would run under Visual Studio .NET 2003 and the .NET Framework 1.1.

NOTE

If you are using ActiveX controls on a form, you may receive the cross-thread [InvalidOperationException](#) when you run under the debugger. When this occurs, the ActiveX control does not support multithreading. For more information about using ActiveX controls with Windows Forms, see [Windows Forms and Unmanaged Applications](#). If you are using Visual Studio, you can prevent this exception by disabling the Visual Studio hosting process, see [How to: Disable the Hosting Process](#).

Making Thread-Safe Calls to Windows Forms Controls

To make a thread-safe call to a Windows Forms control

1. Query the control's [InvokeRequired](#) property.
2. If [InvokeRequired](#) returns `true`, call [Invoke](#) with a delegate that makes the actual call to the control.
3. If [InvokeRequired](#) returns `false`, call the control directly.

In the following code example, a thread-safe call is implemented in the `ThreadProcSafe` method, which is executed by the background thread. If the `TextBox` control's [InvokeRequired](#) returns `true`, the `ThreadProcSafe` method creates an instance of `StringArgReturningVoidDelegate` and passes that to the form's [Invoke](#) method. This causes the `SetText` method to be called on the thread that created the `TextBox` control, and in this thread context the `Text` property is set directly.

```

// This event handler creates a thread that calls a
// Windows Forms control in a thread-safe way.
private void setTextSafeBtn_Click(
    object sender,
    EventArgs e)
{
    this.demoThread =
        new Thread(new ThreadStart(this.ThreadProcSafe));

    this.demoThread.Start();
}

// This method is executed on the worker thread and makes
// a thread-safe call on the TextBox control.
private void ThreadProcSafe()
{
    this.SetText("This text was set safely.");
}

```

```

' This event handler creates a thread that calls a
' Windows Forms control in a thread-safe way.
Private Sub setTextSafeBtn_Click( _
    ByVal sender As Object, _
    ByVal e As EventArgs) Handles setTextSafeBtn.Click

    Me.demoThread = New Thread( _
        New ThreadStart(AddressOf Me.ThreadProcSafe))

    Me.demoThread.Start()
End Sub

```

```

' This method is executed on the worker thread and makes
' a thread-safe call on the TextBox control.
Private Sub ThreadProcSafe()
    Me.SetText("This text was set safely.")
End Sub

```

```

// This event handler creates a thread that calls a
// Windows Forms control in a thread-safe way.
private:
    void setTextSafeBtn_Click(Object^ sender, EventArgs^ e)
    {
        this->demoThread =
            gcnew Thread(gcnew ThreadStart(this,&Form1::ThreadProcSafe));

        this->demoThread->Start();
    }

// This method is executed on the worker thread and makes
// a thread-safe call on the TextBox control.
private:
    void ThreadProcSafe()
    {
        this->SetText("This text was set safely.");
    }

```

```

// This delegate enables asynchronous calls for setting
// the text property on a TextBox control.
delegate void StringArgReturningVoidDelegate(string text);

```

```
' This delegate enables asynchronous calls for setting
' the text property on a TextBox control.
Delegate Sub StringArgReturningVoidDelegate([text] As String)
```

```
// This delegate enables asynchronous calls for setting
// the text property on a TextBox control.
delegate void StringArgReturningVoidDelegate(String^ text);
```

```
// This method demonstrates a pattern for making thread-safe
// calls on a Windows Forms control.
//
// If the calling thread is different from the thread that
// created the TextBox control, this method creates a
// StringArgReturningVoidDelegate and calls itself asynchronously using the
// Invoke method.
//
// If the calling thread is the same as the thread that created
// the TextBox control, the Text property is set directly.

private void SetText(string text)
{
    // InvokeRequired required compares the thread ID of the
    // calling thread to the thread ID of the creating thread.
    // If these threads are different, it returns true.
    if (this.textBox1.InvokeRequired)
    {
        StringArgReturningVoidDelegate d = new StringArgReturningVoidDelegate(SetText);
        this.Invoke(d, new object[] { text });
    }
    else
    {
        this.textBox1.Text = text;
    }
}
```

```
' This method demonstrates a pattern for making thread-safe
' calls on a Windows Forms control.
'
' If the calling thread is different from the thread that
' created the TextBox control, this method creates a
' StringArgReturningVoidDelegate and calls itself asynchronously using the
' Invoke method.
'
' If the calling thread is the same as the thread that created
' the TextBox control, the Text property is set directly.
```

```
Private Sub SetText(ByVal [text] As String)

    ' InvokeRequired required compares the thread ID of the
    ' calling thread to the thread ID of the creating thread.
    ' If these threads are different, it returns true.
    If Me.textBox1.InvokeRequired Then
        Dim d As New StringArgReturningVoidDelegate(AddressOf SetText)
        Me.Invoke(d, New Object() {[text]})
    Else
        Me.textBox1.Text = [text]
    End If
End Sub
```

```

// This method demonstrates a pattern for making thread-safe
// calls on a Windows Forms control.
//
// If the calling thread is different from the thread that
// created the TextBox control, this method creates a
// StringArgReturningVoidDelegate and calls itself asynchronously using the
// Invoke method.
//
// If the calling thread is the same as the thread that created
// the TextBox control, the Text property is set directly.

private:
    void SetText(String^ text)
    {
        // InvokeRequired required compares the thread ID of the
        // calling thread to the thread ID of the creating thread.
        // If these threads are different, it returns true.
        if (this->textBox1->InvokeRequired)
        {
            StringArgReturningVoidDelegate^ d =
                gcnew StringArgReturningVoidDelegate(this, &Form1::SetText);
            this->Invoke(d, gcnew array<Object^> { text });
        }
        else
        {
            this->textBox1->Text = text;
        }
    }
}

```

Making Thread-Safe Calls by using BackgroundWorker

The preferred way to implement multithreading in your application is to use the [BackgroundWorker](#) component. The [BackgroundWorker](#) component uses an event-driven model for multithreading. The background thread runs your [DoWork](#) event handler, and the thread that creates your controls runs your [ProgressChanged](#) and [RunWorkerCompleted](#) event handlers. You can call your controls from your [ProgressChanged](#) and [RunWorkerCompleted](#) event handlers.

To make thread-safe calls by using BackgroundWorker

1. Create a method to do the work that you want done in the background thread. Do not call controls created by the main thread in this method.
2. Create a method to report the results of your background work after it finishes. You can call controls created by the main thread in this method.
3. Bind the method created in step 1 to the [DoWork](#) event of an instance of [BackgroundWorker](#), and bind the method created in step 2 to the same instance's [RunWorkerCompleted](#) event.
4. To start the background thread, call the [RunWorkerAsync](#) method of the [BackgroundWorker](#) instance.

In the following code example, the [DoWork](#) event handler uses [Sleep](#) to simulate work that takes some time. It does not call the form's [TextBox](#) control. The [TextBox](#) control's [Text](#) property is set directly in the [RunWorkerCompleted](#) event handler.

```

// This BackgroundWorker is used to demonstrate the
// preferred way of performing asynchronous operations.
private BackgroundWorker backgroundWorker1;

```

```
' This BackgroundWorker is used to demonstrate the
' preferred way of performing asynchronous operations.
Private WithEvents backgroundWorker1 As BackgroundWorker
```

```
// This BackgroundWorker is used to demonstrate the
// preferred way of performing asynchronous operations.
private:
    BackgroundWorker^ backgroundWorker1;
```

```
// This event handler starts the form's
// BackgroundWorker by calling RunWorkerAsync.
//
// The Text property of the TextBox control is set
// when the BackgroundWorker raises the RunWorkerCompleted
// event.
private void setTextBackgroundWorkerBtn_Click(
    object sender,
    EventArgs e)
{
    this.backgroundWorker1.RunWorkerAsync();
}

// This event handler sets the Text property of the TextBox
// control. It is called on the thread that created the
// TextBox control, so the call is thread-safe.
//
// BackgroundWorker is the preferred way to perform asynchronous
// operations.

private void backgroundWorker1_RunWorkerCompleted(
    object sender,
    RunWorkerCompletedEventArgs e)
{
    this.textBox1.Text =
        "This text was set safely by BackgroundWorker.";
}
```

```

' This event handler starts the form's
' BackgroundWorker by calling RunWorkerAsync.
'

' The Text property of the TextBox control is set
' when the BackgroundWorker raises the RunWorkerCompleted
' event.
Private Sub setTextBackgroundWorkerBtn_Click( _
ByVal sender As Object, _
ByVal e As EventArgs) Handles setTextBackgroundWorkerBtn.Click
    Me.backgroundWorker1.RunWorkerAsync()
End Sub

' This event handler sets the Text property of the TextBox
' control. It is called on the thread that created the
' TextBox control, so the call is thread-safe.
'

' BackgroundWorker is the preferred way to perform asynchronous
' operations.
Private Sub backgroundWorker1_RunWorkerCompleted( _
ByVal sender As Object, _
ByVal e As RunWorkerCompletedEventArgs) _
Handles backgroundWorker1.RunWorkerCompleted
    Me.textBox1.Text = _
        "This text was set safely by BackgroundWorker."
End Sub

```

```

// This event handler starts the form's
// BackgroundWorker by calling RunWorkerAsync.
//
// The Text property of the TextBox control is set
// when the BackgroundWorker raises the RunWorkerCompleted
// event.
private:
    void setTextBackgroundWorkerBtn_Click(Object^ sender, EventArgs^ e)
    {
        this->backgroundWorker1->RunWorkerAsync();
    }

    // This event handler sets the Text property of the TextBox
    // control. It is called on the thread that created the
    // TextBox control, so the call is thread-safe.
    //
    // BackgroundWorker is the preferred way to perform asynchronous
    // operations.

private:
    void backgroundWorker1_RunWorkerCompleted(
        Object^ sender,
        RunWorkerCompletedEventArgs^ e)
    {
        this->textBox1->Text =
            "This text was set safely by BackgroundWorker.";
    }

```

You can also report the progress of a background task by using the [ProgressChanged](#) event. For an example that incorporates that event, see [BackgroundWorker](#).

Example

The following code example is a complete Windows Forms application that consists of a form with three buttons and one text box. The first button demonstrates unsafe cross-thread access, the second button demonstrates safe access by using [Invoke](#), and the third button demonstrates safe access by using [BackgroundWorker](#).

NOTE

For instructions on how to run the example, see [How to: Compile and Run a Complete Windows Forms Code Example Using Visual Studio](#). This example requires references to the System.Drawing and System.Windows.Forms assemblies.

```
using System;
using System.ComponentModel;
using System.Threading;
using System.Windows.Forms;

namespace CrossThreadDemo
{
    public class Form1 : Form
    {
        // This delegate enables asynchronous calls for setting
        // the text property on a TextBox control.
        delegate void StringArgReturningVoidDelegate(string text);

        // This thread is used to demonstrate both thread-safe and
        // unsafe ways to call a Windows Forms control.
        private Thread demoThread = null;

        // This BackgroundWorker is used to demonstrate the
        // preferred way of performing asynchronous operations.
        private BackgroundWorker backgroundWorker1;

        private TextBox textBox1;
        private Button setTextUnsafeBtn;
        private Button setTextSafeBtn;
        private Button setTextBackgroundWorkerBtn;

        private System.ComponentModel.IContainer components = null;

        public Form1()
        {
            InitializeComponent();
        }

        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        // This event handler creates a thread that calls a
        // Windows Forms control in an unsafe way.
        private void setTextUnsafeBtn_Click(
            object sender,
            EventArgs e)
        {
            this.demoThread =
                new Thread(new ThreadStart(this.ThreadProcUnsafe));

            this.demoThread.Start();
        }

        // This method is executed on the worker thread and makes
        // an unsafe call on the TextBox control.
        private void ThreadProcUnsafe()
        {
            this.textBox1.Text = "This text was set unsafely.";
        }
    }
}
```

```

// This event handler creates a thread that calls a
// Windows Forms control in a thread-safe way.
private void setTextSafeBtn_Click(
    object sender,
    EventArgs e)
{
    this.demoThread =
        new Thread(new ThreadStart(this.ThreadProcSafe));

    this.demoThread.Start();
}

// This method is executed on the worker thread and makes
// a thread-safe call on the TextBox control.
private void ThreadProcSafe()
{
    this.SetText("This text was set safely.");
}

// This method demonstrates a pattern for making thread-safe
// calls on a Windows Forms control.
//
// If the calling thread is different from the thread that
// created the TextBox control, this method creates a
// StringArgReturningVoidDelegate and calls itself asynchronously using the
// Invoke method.
//
// If the calling thread is the same as the thread that created
// the TextBox control, the Text property is set directly.

private void SetText(string text)
{
    // InvokeRequired required compares the thread ID of the
    // calling thread to the thread ID of the creating thread.
    // If these threads are different, it returns true.
    if (this.textBox1.InvokeRequired)
    {
        StringArgReturningVoidDelegate d = new StringArgReturningVoidDelegate(SetText);
        this.Invoke(d, new object[] { text });
    }
    else
    {
        this.textBox1.Text = text;
    }
}

// This event handler starts the form's
// BackgroundWorker by calling RunWorkerAsync.
//
// The Text property of the TextBox control is set
// when the BackgroundWorker raises the RunWorkerCompleted
// event.
private void setTextBackgroundWorkerBtn_Click(
    object sender,
    EventArgs e)
{
    this.backgroundWorker1.RunWorkerAsync();
}

// This event handler sets the Text property of the TextBox
// control. It is called on the thread that created the
// TextBox control, so the call is thread-safe.
//
// BackgroundWorker is the preferred way to perform asynchronous
// operations.

private void backgroundWorker1_RunWorkerCompleted(
    object sender,
    RunWorkerCompletedEventArgs e)
{
}

```

```

        RunWorkerCompletedEventArgs e)
{
    this.textBox1.Text =
        "This text was set safely by BackgroundWorker.";
}

#region Windows Form Designer generated code

private void InitializeComponent()
{
    this.textBox1 = new System.Windows.Forms.TextBox();
    this.setTextUnsafeBtn = new System.Windows.Forms.Button();
    this.setTextSafeBtn = new System.Windows.Forms.Button();
    this.setTextBackgroundWorkerBtn = new System.Windows.Forms.Button();
    this.backgroundWorker1 = new System.ComponentModel.BackgroundWorker();
    this.SuspendLayout();
    //
    // textBox1
    //
    this.textBox1.Location = new System.Drawing.Point(12, 12);
    this.textBox1.Name = "textBox1";
    this.textBox1.Size = new System.Drawing.Size(240, 20);
    this.textBox1.TabIndex = 0;
    //
    // setTextUnsafeBtn
    //
    this.setTextUnsafeBtn.Location = new System.Drawing.Point(15, 55);
    this.setTextUnsafeBtn.Name = "setTextUnsafeBtn";
    this.setTextUnsafeBtn.TabIndex = 1;
    this.setTextUnsafeBtn.Text = "Unsafe Call";
    this.setTextUnsafeBtn.Click += new System.EventHandler(this.setTextUnsafeBtn_Click);
    //
    // setTextSafeBtn
    //
    this.setTextSafeBtn.Location = new System.Drawing.Point(96, 55);
    this.setTextSafeBtn.Name = "setTextSafeBtn";
    this.setTextSafeBtn.TabIndex = 2;
    this.setTextSafeBtn.Text = "Safe Call";
    this.setTextSafeBtn.Click += new System.EventHandler(this.setTextSafeBtn_Click);
    //
    // setTextBackgroundWorkerBtn
    //
    this.setTextBackgroundWorkerBtn.Location = new System.Drawing.Point(177, 55);
    this.setTextBackgroundWorkerBtn.Name = "setTextBackgroundWorkerBtn";
    this.setTextBackgroundWorkerBtn.TabIndex = 3;
    this.setTextBackgroundWorkerBtn.Text = "Safe BW Call";
    this.setTextBackgroundWorkerBtn.Click += new
System.EventHandler(this.setTextBackgroundWorkerBtn_Click);
    //
    // backgroundWorker1
    //
    this.backgroundWorker1.RunWorkerCompleted += new
System.ComponentModel.RunWorkerCompletedEventHandler(this.backgroundWorker1_RunWorkerCompleted);
    //
    // Form1
    //
    this.ClientSize = new System.Drawing.Size(268, 96);
    this.Controls.Add(this.setTextBackgroundWorkerBtn);
    this.Controls.Add(this.setTextSafeBtn);
    this.Controls.Add(this.setTextUnsafeBtn);
    this.Controls.Add(this.textBox1);
    this.Name = "Form1";
    this.Text = "Form1";
    this.ResumeLayout(false);
    this.PerformLayout();
}

#endregion

```

```

[STAThread]
static void Main()
{
    Application.EnableVisualStyles();
    Application.Run(new Form1());
}

}

```

```

Imports System
Imports System.ComponentModel
Imports System.Threading
Imports System.Windows.Forms

Public Class Form1
    Inherits Form

    ' This delegate enables asynchronous calls for setting
    ' the Text property on a TextBox control.
    Delegate Sub StringArgReturningVoidDelegate([text] As String)

    ' This thread is used to demonstrate both thread-safe and
    ' unsafe ways to call a Windows Forms control.
    Private demoThread As Thread = Nothing

    ' This BackgroundWorker is used to demonstrate the
    ' preferred way of performing asynchronous operations.
    Private WithEvents backgroundWorker1 As BackgroundWorker

    Private textBox1 As TextBox
    Private WithEvents setTextUnsafeBtn As Button
    Private WithEvents setTextSafeBtn As Button
    Private WithEvents setTextBackgroundWorkerBtn As Button

    Private components As System.ComponentModel.IContainer = Nothing

    Public Sub New()
        InitializeComponent()
    End Sub

    Protected Overrides Sub Dispose(disposing As Boolean)
        If disposing AndAlso (components IsNot Nothing) Then
            components.Dispose()
        End If
        MyBase.Dispose(disposing)
    End Sub

    ' This event handler creates a thread that calls a
    ' Windows Forms control in an unsafe way.
    Private Sub setTextUnsafeBtn_Click(
        ByVal sender As Object,
        ByVal e As EventArgs) Handles setTextUnsafeBtn.Click

        Me.demoThread = New Thread( _
        New ThreadStart(AddressOf Me.ThreadProcUnsafe))

        Me.demoThread.Start()
    End Sub

    ' This method is executed on the worker thread and makes
    ' an unsafe call on the TextBox control.
    Private Sub ThreadProcUnsafe()
        Me.textBox1.Text = "This text was set unsafely."
    End Sub

```

```

' This event handler creates a thread that calls a
' Windows Forms control in a thread-safe way.
Private Sub setTextSafeBtn_Click( _
    ByVal sender As Object, _
    ByVal e As EventArgs) Handles setTextSafeBtn.Click

    Me.demoThread = New Thread( _
        New ThreadStart(AddressOf Me.ThreadProcSafe))

    Me.demoThread.Start()
End Sub

' This method is executed on the worker thread and makes
' a thread-safe call on the TextBox control.
Private Sub ThreadProcSafe()
    Me.SetText("This text was set safely.")
End Sub

' This method demonstrates a pattern for making thread-safe
' calls on a Windows Forms control.
'

' If the calling thread is different from the thread that
' created the TextBox control, this method creates a
' StringArgReturningVoidDelegate and calls itself asynchronously using the
' Invoke method.
'

' If the calling thread is the same as the thread that created
' the TextBox control, the Text property is set directly.

Private Sub SetText(ByVal [text] As String)

    ' InvokeRequired required compares the thread ID of the
    ' calling thread to the thread ID of the creating thread.
    ' If these threads are different, it returns true.
    If Me.textBox1.InvokeRequired Then
        Dim d As New StringArgReturningVoidDelegate(AddressOf SetText)
        Me.Invoke(d, New Object() {[text]})
    Else
        Me.textBox1.Text = [text]
    End If
End Sub

' This event handler starts the form's
' BackgroundWorker by calling RunWorkerAsync.
'

' The Text property of the TextBox control is set
' when the BackgroundWorker raises the RunWorkerCompleted
' event.
Private Sub setTextBackgroundWorkerBtn_Click( _
    ByVal sender As Object, _
    ByVal e As EventArgs) Handles setTextBackgroundWorkerBtn.Click
    Me.backgroundWorker1.RunWorkerAsync()
End Sub

' This event handler sets the Text property of the TextBox
' control. It is called on the thread that created the
' TextBox control, so the call is thread-safe.
'

' BackgroundWorker is the preferred way to perform asynchronous
' operations.
Private Sub backgroundWorker1_RunWorkerCompleted( _
    ByVal sender As Object, _
    ByVal e As RunWorkerCompletedEventArgs) _
Handles backgroundWorker1.RunWorkerCompleted
    Me.textBox1.Text = _
        "This text was set safely by BackgroundWorker."
End Sub

#Region "Windows Form Designer generated code"

```

```

Private Sub InitializeComponent()
    Me.textBox1 = New System.Windows.Forms.TextBox()
    Me.setTextUnsafeBtn = New System.Windows.Forms.Button()
    Me.setTextSafeBtn = New System.Windows.Forms.Button()
    Me.setTextBackgroundWorkerBtn = New System.Windows.Forms.Button()
    Me.backgroundWorker1 = New System.ComponentModel.BackgroundWorker()
    Me.SuspendLayout()

    ' textBox1

    Me.textBox1.Location = New System.Drawing.Point(12, 12)
    Me.textBox1.Name = "textBox1"
    Me.textBox1.Size = New System.Drawing.Size(240, 20)
    Me.textBox1.TabIndex = 0

    ' setTextUnsafeBtn

    Me.setTextUnsafeBtn.Location = New System.Drawing.Point(15, 55)
    Me.setTextUnsafeBtn.Name = "setTextUnsafeBtn"
    Me.setTextUnsafeBtn.TabIndex = 1
    Me.setTextUnsafeBtn.Text = "Unsafe Call"

    ' setTextSafeBtn

    Me.setTextSafeBtn.Location = New System.Drawing.Point(96, 55)
    Me.setTextSafeBtn.Name = "setTextSafeBtn"
    Me.setTextSafeBtn.TabIndex = 2
    Me.setTextSafeBtn.Text = "Safe Call"

    ' setTextBackgroundWorkerBtn

    Me.setTextBackgroundWorkerBtn.Location = New System.Drawing.Point(177, 55)
    Me.setTextBackgroundWorkerBtn.Name = "setTextBackgroundWorkerBtn"
    Me.setTextBackgroundWorkerBtn.TabIndex = 3
    Me.setTextBackgroundWorkerBtn.Text = "Safe BW Call"

    ' backgroundWorker1

    ' Form1

    Me.ClientSize = New System.Drawing.Size(268, 96)
    Me.Controls.Add(setTextBackgroundWorkerBtn)
    Me.Controls.Add(setTextSafeBtn)
    Me.Controls.Add(setTextUnsafeBtn)
    Me.Controls.Add(textBox1)
    Me.Name = "Form1"
    Me.Text = "Form1"
    Me.ResumeLayout(False)
    Me.PerformLayout()
End Sub 'InitializeComponent

#End Region

<STAThread()> _
Shared Sub Main()
    Application.EnableVisualStyles()
    Application.Run(New Form1())
End Sub
End Class

```

```

#using <System.dll>
#using <System.Windows.Forms.dll>
#using <System.Drawing.dll>

using namespace System;

```

```

using namespace System::ComponentModel;
using namespace System::Threading;
using namespace System::Windows::Forms;

namespace CrossThreadDemo
{
    public ref class Form1 : public Form
    {
        // This delegate enables asynchronous calls for setting
        // the text property on a TextBox control.
        delegate void StringArgReturningVoidDelegate(String^ text);

        // This thread is used to demonstrate both thread-safe and
        // unsafe ways to call a Windows Forms control.
        private:
            Thread^ demoThread;

        // This BackgroundWorker is used to demonstrate the
        // preferred way of performing asynchronous operations.
        private:
            BackgroundWorker^ backgroundWorker1;

        private:
            TextBox^ textBox1;
        private:
            Button^ setTextUnsafeBtn;
        private:
            Button^ setTextSafeBtn;
        private:
            Button^ setTextBackgroundWorkerBtn;

        private:
            System::ComponentModel::.IContainer^ components;

        public:
            Form1()
            {
                components = nullptr;
                InitializeComponent();
            }

        protected:
            ~Form1()
            {
                if (components != nullptr)
                {
                    delete components;
                }
            }

        // This event handler creates a thread that calls a
        // Windows Forms control in an unsafe way.
        private:
            void setTextUnsafeBtn_Click(Object^ sender, EventArgs^ e)
            {
                this->demoThread =
                    gcnew Thread(gcnew ThreadStart(this,&Form1::ThreadProcUnsafe));

                this->demoThread->Start();
            }

        // This method is executed on the worker thread and makes
        // an unsafe call on the TextBox control.
        private:
            void ThreadProcUnsafe()
            {
                this->textBox1->Text = "This text was set unsafely.";
            }
    }
}

```

```

// This event handler creates a thread that calls a
// Windows Forms control in a thread-safe way.
private:
    void setTextSafeBtn_Click(Object^ sender, EventArgs^ e)
    {
        this->demoThread =
            gcnew Thread(gcnew ThreadStart(this,&Form1::ThreadProcSafe));

        this->demoThread->Start();
    }

// This method is executed on the worker thread and makes
// a thread-safe call on the TextBox control.
private:
    void ThreadProcSafe()
    {
        this->SetText("This text was set safely.");
    }

// This method demonstrates a pattern for making thread-safe
// calls on a Windows Forms control.
//
// If the calling thread is different from the thread that
// created the TextBox control, this method creates a
// StringArgReturningVoidDelegate and calls itself asynchronously using the
// Invoke method.
//
// If the calling thread is the same as the thread that created
// the TextBox control, the Text property is set directly.

private:
    void SetText(String^ text)
    {
        // InvokeRequired required compares the thread ID of the
        // calling thread to the thread ID of the creating thread.
        // If these threads are different, it returns true.
        if (this->textBox1->InvokeRequired)
        {
            StringArgReturningVoidDelegate^ d =
                gcnew StringArgReturningVoidDelegate(this, &Form1::SetText);
            this->Invoke(d, gcnew array<Object^> { text });
        }
        else
        {
            this->textBox1->Text = text;
        }
    }

// This event handler starts the form's
// BackgroundWorker by calling RunWorkerAsync.
//
// The Text property of the TextBox control is set
// when the BackgroundWorker raises the RunWorkerCompleted
// event.
private:
    void setTextBackgroundWorkerBtn_Click(Object^ sender, EventArgs^ e)
    {
        this->backgroundWorker1->RunWorkerAsync();
    }

// This event handler sets the Text property of the TextBox
// control. It is called on the thread that created the
// TextBox control, so the call is thread-safe.
//
// BackgroundWorker is the preferred way to perform asynchronous
// operations.

private:
    void backgroundWorker1_RunWorkerCompleted(

```

```

Object^ sender,
RunWorkerCompletedEventArgs^ e)
{
    this->textBox1->Text =
        "This text was set safely by BackgroundWorker.";
}

#pragma region Windows Form Designer generated code

private:
    void InitializeComponent()
    {
        this->textBox1 = gcnew System::Windows::Forms::TextBox();
        this->setTextUnsafeBtn = gcnew System::Windows::Forms::Button();
        this->setTextSafeBtn = gcnew System::Windows::Forms::Button();
        this->setTextBackgroundWorkerBtn =
            gcnew System::Windows::Forms::Button();
        this->backgroundWorker1 =
            gcnew System::ComponentModel::BackgroundWorker();
        this->SuspendLayout();
        //
        // textBox1
        //
        this->textBox1->Location = System::Drawing::Point(12, 12);
        this->textBox1->Name = "textBox1";
        this->textBox1->Size = System::Drawing::Size(240, 20);
        this->textBox1->TabIndex = 0;
        //
        // setTextUnsafeBtn
        //
        this->setTextUnsafeBtn->Location = System::Drawing::Point(15, 55);
        this->setTextUnsafeBtn->Name = "setTextUnsafeBtn";
        this->setTextUnsafeBtn->TabIndex = 1;
        this->setTextUnsafeBtn->Text = "Unsafe Call";
        this->setTextUnsafeBtn->Click +=
            gcnew System::EventHandler(
                this,&Form1::setTextUnsafeBtn_Click);
        //
        // setTextSafeBtn
        //
        this->setTextSafeBtn->Location = System::Drawing::Point(96, 55);
        this->setTextSafeBtn->Name = "setTextSafeBtn";
        this->setTextSafeBtn->TabIndex = 2;
        this->setTextSafeBtn->Text = "Safe Call";
        this->setTextSafeBtn->Click +=
            gcnew System::EventHandler(this,&Form1::setTextSafeBtn_Click);
        //
        // setTextBackgroundWorkerBtn
        //
        this->setTextBackgroundWorkerBtn->Location =
            System::Drawing::Point(177, 55);
        this->setTextBackgroundWorkerBtn->Name =
            "setTextBackgroundWorkerBtn";
        this->setTextBackgroundWorkerBtn->TabIndex = 3;
        this->setTextBackgroundWorkerBtn->Text = "Safe BW Call";
        this->setTextBackgroundWorkerBtn->Click +=
            gcnew System::EventHandler(
                this,&Form1::setTextBackgroundWorkerBtn_Click);
        //
        // backgroundWorker1
        //
        this->backgroundWorker1->RunWorkerCompleted +=
            gcnew System::ComponentModel::RunWorkerCompletedEventHandler(
                this,&Form1::backgroundWorker1_RunWorkerCompleted);
        //
        // Form1
        //
        this->ClientSize = System::Drawing::Size(268, 96);
        this->Controls->Add(this->setTextBackgroundWorkerBtn);
    }
}

```

```

        this->Controls->Add(this->setTextSafeBtn);
        this->Controls->Add(this->setTextUnsafeBtn);
        this->Controls->Add(this->textBox1);
        this->Name = "Form1";
        this->Text = "Form1";
        this->ResumeLayout(false);
        this->PerformLayout();

    }

    #pragma endregion
};

}

[STAThread]
int main()
{
    Application::EnableVisualStyles();
    Application::Run(gcnew CrossThreadDemo::Form1());
}

```

When you run the application and click the **Unsafe Call** button, you immediately see "Written by the main thread" in the text box. Two seconds later, when the unsafe call is attempted, the Visual Studio debugger indicates that an exception occurred. The debugger stops at the line in the background thread that attempted to write directly to the text box. You will have to restart the application to test the other two buttons. When you click the **Safe Call** button, "Written by the main thread" appears in the text box. Two seconds later, the text box is set to "Written by the background thread (Invoke)", which indicates that the [Invoke](#) method was called. When you click the **Safe BW Call** button, "Written by the main thread" appears in the text box. Two seconds later, the text box is set to "Written by the main thread after the background thread completed", which indicates that the handler for the [RunWorkerCompleted](#) event of [BackgroundWorker](#) was called.

Robust Programming

Caution

When you use multithreading of any sort, your code can be exposed to very serious and complex bugs. For more information, see [Managed Threading Best Practices](#) before you implement any solution that uses multithreading.

See Also

[BackgroundWorker](#)

[How to: Run an Operation in the Background](#)

[How to: Implement a Form That Uses a Background Operation](#)

[Developing Custom Windows Forms Controls with the .NET Framework](#)

[Windows Forms and Unmanaged Applications](#)

How to: Use a Background Thread to Search for Files

5/4/2018 • 14 min to read • [Edit Online](#)

The [BackgroundWorker](#) component replaces and adds functionality to the [System.Threading](#) namespace; however, the [System.Threading](#) namespace is retained for both backward compatibility and future use, if you choose. For more information, see [BackgroundWorker Component Overview](#).

Windows Forms uses the single-threaded apartment (STA) model because Windows Forms is based on native Win32 windows that are inherently apartment-threaded. The STA model implies that a window can be created on any thread, but it cannot switch threads once created, and all function calls to it must occur on its creation thread. Outside Windows Forms, classes in the .NET Framework use the free threading model. For information about threading in the .NET Framework, see [Threading](#).

The STA model requires that any methods on a control that need to be called from outside the control's creation thread must be marshaled to (executed on) the control's creation thread. The base class [Control](#) provides several methods ([Invoke](#), [BeginInvoke](#), and [EndInvoke](#)) for this purpose. [Invoke](#) makes synchronous method calls; [BeginInvoke](#) makes asynchronous method calls.

If you use multithreading in your control for resource-intensive tasks, the user interface can remain responsive while a resource-intensive computation executes on a background thread.

The following sample ([DirectorySearcher](#)) shows a multithreaded Windows Forms control that uses a background thread to recursively search a directory for files matching a specified search string and then populates a list box with the search result. The key concepts illustrated by the sample are as follows:

- [DirectorySearcher](#) starts a new thread to perform the search. The thread executes the [ThreadProcedure](#) method that in turn calls the helper [RecurseDirectory](#) method to do the actual search and to populate the list box. However, populating the list box requires a cross-thread call, as explained in the next two bulleted items.
- [DirectorySearcher](#) defines the [AddFiles](#) method to add files to a list box; however, [RecurseDirectory](#) cannot directly invoke [AddFiles](#) because [AddFiles](#) can execute only in the STA thread that created [DirectorySearcher](#).
- The only way [RecurseDirectory](#) can call [AddFiles](#) is through a cross-thread call — that is, by calling [Invoke](#) or [BeginInvoke](#) to marshal [AddFiles](#) to the creation thread of [DirectorySearcher](#). [RecurseDirectory](#) uses [BeginInvoke](#) so that the call can be made asynchronously.
- Marshaling a method requires the equivalent of a function pointer or callback. This is accomplished using delegates in the .NET Framework. [BeginInvoke](#) takes a delegate as an argument. [DirectorySearcher](#) therefore defines a delegate ([FileListDelegate](#)), binds [AddFiles](#) to an instance of [FileListDelegate](#) in its constructor, and passes this delegate instance to [BeginInvoke](#). [DirectorySearcher](#) also defines an event delegate that is marshaled when the search is completed.

```
Option Strict
Option Explicit

Imports System
Imports System.IO
Imports System.Threading
Imports System.Windows.Forms

Namespace Microsoft.Samples.DirectorySearcher
    ' <summary>
```

```

' This class is a Windows Forms control that implements a simple directory searcher.
' You provide, through code, a search string and it will search directories on
' a background thread, populating its list box with matches.
' </summary>
Public Class DirectorySearcher
    Inherits Control
    ' Define a special delegate that handles marshaling
    ' lists of file names from the background directory search
    ' thread to the thread that contains the list box.
    Delegate Sub FileListDelegate(files() As String, startIndex As Integer, count As Integer)

    Private _listBox As ListBox
    Private _searchCriteria As String
    Private _searching As Boolean
    Private _deferSearch As Boolean
    Private _searchThread As Thread
    Private _fileListDelegate As FileListDelegate
    Private _onSearchComplete As EventHandler

    Public Sub New()
        _listBox = New ListBox()
        _listBox.Dock = DockStyle.Fill

        Controls.Add(_listBox)

        _fileListDelegate = New FileListDelegate(AddressOf AddFiles)
        _onSearchComplete = New EventHandler(AddressOf OnSearchComplete)
    End Sub

    Public Property SearchCriteria() As String
        Get
            Return _searchCriteria
        End Get
        Set
            ' If currently searching, abort
            ' the search and restart it after
            ' setting the new criteria.
            '
            Dim wasSearching As Boolean = Searching

            If wasSearching Then
                StopSearch()
            End If

            _listBox.Items.Clear()
            _searchCriteria = value

            If wasSearching Then
                BeginSearch()
            End If
        End Set
    End Property

    Public ReadOnly Property Searching() As Boolean
        Get
            Return _searching
        End Get
    End Property

    Public Event SearchComplete As EventHandler

    ' <summary>
    ' This method is called from the background thread. It is called through
    ' a BeginInvoke call so that it is always marshaled to the thread that
    ' owns the list box control.
    ' </summary>
    ' <param name="files"></param>
    ' <param name="startIndex"></param>
    ' <param name="count"></param>

```

```

    <param name="Count"></param>
Private Sub AddFiles(files() As String, startIndex As Integer, count As Integer)
    While count > 0
        count -= 1
        _listBox.Items.Add(files((startIndex + count)))
    End While
End Sub

Public Sub BeginSearch()
    ' Create the search thread, which
    ' will begin the search.
    ' If already searching, do nothing.
    '

    If Searching Then
        Return
    End If

    ' Start the search if the handle has
    ' been created. Otherwise, defer it until the
    ' handle has been created.
    If IsHandleCreated Then
        _searchThread = New Thread(New ThreadStart(AddressOf ThreadProcedure))
        _searching = True
        _searchThread.Start()
    Else
        _deferSearch = True
    End If
End Sub

Protected Overrides Sub OnHandleDestroyed(e As EventArgs)
    ' If the handle is being destroyed and you are not
    ' recreating it, then abort the search.
    If Not RecreatingHandle Then
        StopSearch()
    End If
    MyBase.OnHandleDestroyed(e)
End Sub

Protected Overrides Sub OnHandleCreated(e As EventArgs)
    MyBase.OnHandleCreated(e)
    If _deferSearch Then
        _deferSearch = False
        BeginSearch()
    End If
End Sub

' <summary>
' This method is called by the background thread when it has
' finished the search.
' </summary>
' <param name="sender"></param>
' <param name="e"></param>
Private Sub OnSearchComplete(sender As Object, e As EventArgs)
    RaiseEvent SearchComplete(sender, e)
End Sub

Public Sub StopSearch()
    If Not _searching Then
        Return
    End If

    If _searchThread.IsAlive Then
        _searchThread.Abort()
        _searchThread.Join()
    End If

    _searchThread = Nothing
    _searching = False
End Sub

```

```

' <summary>
' Recurses the given path, adding all files on that path to
' the list box. After it finishes with the files, it
' calls itself once for each directory on the path.
' </summary>
' <param name="searchPath"></param>
Private Sub RecurseDirectory(searchPath As String)
    ' Split searchPath into a directory and a wildcard specification.
    '

    Dim directoryPath As String = Path.GetDirectoryName(searchPath)
    Dim search As String = Path.GetFileName(searchPath)

    ' If a directory or search criteria are not specified, then return.
    '

    If directoryPath Is Nothing Or search Is Nothing Then
        Return
    End If

    Dim files() As String

    ' File systems like NTFS that have
    ' access permissions might result in exceptions
    ' when looking into directories without permission.
    ' Catch those exceptions and return.
    Try
        files = Directory.GetFiles(directoryPath, search)
    Catch e As UnauthorizedAccessException
        Return
    Catch e As DirectoryNotFoundException
        Return
    End Try

    ' Perform a BeginInvoke call to the list box
    ' in order to marshal to the correct thread. It is not
    ' very efficient to perform this marshal once for every
    ' file, so batch up multiple file calls into one
    ' marshal invocation.
    Dim startingIndex As Integer = 0
    While startingIndex < files.Length
        ' Batch up 20 files at once, unless at the
        ' end.
        '

        Dim count As Integer = 20
        If count + startingIndex >= files.Length Then
            count = files.Length - startingIndex
        End If
        ' Begin the cross-thread call. Because you are passing
        ' immutable objects into this invoke method, you do not have to
        ' wait for it to finish. If these were complex objects, you would
        ' have to either create new instances of them or
        ' wait for the thread to process this invoke before modifying
        ' the objects.
        Dim r As IAsyncResult = BeginInvoke(_fileListDelegate, New Object() {files, startingIndex, count})
        startingIndex += count
    End While
    ' Now that you have finished the files in this directory, recurse
    ' for each subdirectory.
    Dim directories As String() = Directory.GetDirectories(directoryPath)
    Dim d As String
    For Each d In directories
        RecurseDirectory(Path.Combine(d, search))
    Next d
End Sub

'<summary>
' This is the actual thread procedure. This method runs in a background
' thread to scan directories. When finished, it simply exits.
'</summary>

```

```

Private Sub ThreadProcedure()
    ' Get the search string. Individual
    ' field assigns are atomic in .NET, so you do not
    ' need to use any thread synchronization to grab
    ' the string value here.
    Try
        Dim localSearch As String = SearchCriteria

        ' Now, search the file system.
        ' 

        RecurseDirectory(localSearch)
    Finally
        ' You are done with the search, so update.
        ' 

        _searching = False

        ' Raise an event that notifies the user that
        ' the search has terminated.
        ' You do not have to do this through a marshaled call, but
        ' marshaling is recommended for the following reason:
        ' Users of this control do not know that it is
        ' multithreaded, so they expect its events to
        ' come back on the same thread as the control.
        BeginInvoke(_onSearchComplete, New Object() {Me, EventArgs.Empty})
    End Try
End Sub
End Class
End Namespace

```

```

namespace Microsoft.Samples.DirectorySearcher
{
    using System;
    using System.IO;
    using System.Threading;
    using System.Windows.Forms;

    /// <summary>
    /// This class is a Windows Forms control that implements a simple directory searcher.
    /// You provide, through code, a search string and it will search directories on
    /// a background thread, populating its list box with matches.
    /// </summary>
    public class DirectorySearcher : Control
    {
        // Define a special delegate that handles marshaling
        // lists of file names from the background directory search
        // thread to the thread that contains the list box.
        private delegate void FileListDelegate(string[] files, int startIndex, int count);

        private ListBox listBox;
        private string searchCriteria;
        private bool searching;
        private bool deferSearch;
        private Thread searchThread;
        private FileListDelegate fileListDelegate;
        private EventHandler onSearchComplete;

        public DirectorySearcher()
        {
            listBox = new ListBox();
            listBox.Dock = DockStyle.Fill;

            Controls.Add(listBox);

            fileListDelegate = new FileListDelegate(AddFiles);
            onSearchComplete = new EventHandler(OnSearchComplete);
        }
    }
}

```

```

public string SearchCriteria
{
    get
    {
        return searchCriteria;
    }
    set
    {
        // If currently searching, abort
        // the search and restart it after
        // setting the new criteria.
        //
        bool wasSearching = Searching;

        if (wasSearching)
        {
            StopSearch();
        }

        listBox.Items.Clear();
        searchCriteria = value;

        if (wasSearching)
        {
            BeginSearch();
        }
    }
}

public bool Searching
{
    get
    {
        return searching;
    }
}

public event EventHandler SearchComplete;

/// <summary>
/// This method is called from the background thread. It is called through
/// a BeginInvoke call so that it is always marshaled to the thread that
/// owns the list box control.
/// </summary>
/// <param name="files"></param>
/// <param name="startIndex"></param>
/// <param name="count"></param>
private void AddFiles(string[] files, int startIndex, int count)
{
    while(count-- > 0)
    {
        listBox.Items.Add(files[startIndex + count]);
    }
}

public void BeginSearch()
{
    // Create the search thread, which
    // will begin the search.
    // If already searching, do nothing.
    //
    if (Searching)
    {
        return;
    }

    // Start the search if the handle has
    // been created. Otherwise, defer it until the
}

```

```

// handle has been created.
if (IsHandleCreated)
{
    searchThread = new Thread(new ThreadStart(ThreadProcedure));
    searching = true;
    searchThread.Start();
}
else
{
    deferSearch = true;
}
}

protected override void OnHandleDestroyed(EventArgs e)
{
    // If the handle is being destroyed and you are not
    // recreating it, then abort the search.
    if (!RecreatingHandle)
    {
        StopSearch();
    }
    base.OnHandleDestroyed(e);
}

protected override void OnHandleCreated(EventArgs e)
{
    base.OnHandleCreated(e);
    if (deferSearch)
    {
        deferSearch = false;
        BeginSearch();
    }
}

/// <summary>
/// This method is called by the background thread when it has finished
/// the search.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void OnSearchComplete(object sender, EventArgs e)
{
    if (SearchComplete != null)
    {
        SearchComplete(sender, e);
    }
}

public void StopSearch()
{
    if (!searching)
    {
        return;
    }

    if (searchThread.IsAlive)
    {
        searchThread.Abort();
        searchThread.Join();
    }

    searchThread = null;
    searching = false;
}

/// <summary>
/// Recurses the given path, adding all files on that path to
/// the list box. After it finishes with the files, it
/// calls itself once for each directory on the path.

```

```

/// </summary>
/// <param name="searchPath"></param>
private void RecurseDirectory(string searchPath)
{
    // Split searchPath into a directory and a wildcard specification.
    //
    string directory = Path.GetDirectoryName(searchPath);
    string search = Path.GetFileName(searchPath);

    // If a directory or search criteria are not specified, then return.
    //
    if (directory == null || search == null)
    {
        return;
    }

    string[] files;

    // File systems like NTFS that have
    // access permissions might result in exceptions
    // when looking into directories without permission.
    // Catch those exceptions and return.
    try
    {
        files = Directory.GetFiles(directory, search);
    }
    catch(UnauthorizedAccessException)
    {
        return;
    }
    catch(DirectoryNotFoundException)
    {
        return;
    }

    // Perform a BeginInvoke call to the list box
    // in order to marshal to the correct thread. It is not
    // very efficient to perform this marshal once for every
    // file, so batch up multiple file calls into one
    // marshal invocation.
    int startingIndex = 0;

    while(startingIndex < files.Length)
    {
        // Batch up 20 files at once, unless at the
        // end.
        //
        int count = 20;
        if (count + startingIndex >= files.Length)
        {
            count = files.Length - startingIndex;
        }

        // Begin the cross-thread call. Because you are passing
        // immutable objects into this invoke method, you do not have to
        // wait for it to finish. If these were complex objects, you would
        // have to either create new instances of them or
        // wait for the thread to process this invoke before modifying
        // the objects.
        IAsyncResult r = BeginInvoke(fileListDelegate, new object[] {files, startingIndex, count});
        startingIndex += count;
    }

    // Now that you have finished the files in this directory, recurse for
    // each subdirectory.
    string[] directories = Directory.GetDirectories(directory);
    foreach(string d in directories)
    {
        RecurseDirectory(Path.Combine(d, search));
    }
}

```

```

        }

    /// <summary>
    /// This is the actual thread procedure. This method runs in a background
    /// thread to scan directories. When finished, it simply exits.
    /// </summary>
    private void ThreadProcedure()
    {
        // Get the search string. Individual
        // field assigns are atomic in .NET, so you do not
        // need to use any thread synchronization to grab
        // the string value here.
        try
        {
            string localSearch = SearchCriteria;

            // Now, search the file system.
            //
            RecurseDirectory(localSearch);
        }
        finally
        {
            // You are done with the search, so update.
            //
            searching = false;

            // Raise an event that notifies the user that
            // the search has terminated.
            // You do not have to do this through a marshaled call, but
            // marshaling is recommended for the following reason:
            // Users of this control do not know that it is
            // multithreaded, so they expect its events to
            // come back on the same thread as the control.
            BeginInvoke(onSearchComplete, new object[] {this, EventArgs.Empty});
        }
    }
}
}
}

```

Using the Multithreaded Control on a Form

The following example shows how the multithreaded `DirectorySearcher` control can be used on a form.

```

Option Explicit
Option Strict

Imports Microsoft.Samples.DirectorySearcher
Imports System
Imports System.Drawing
Imports System.Collections
Imports System.ComponentModel
Imports System.Windows.Forms
Imports System.Data

Namespace SampleUsage

    ' <summary>
    '     Summary description for Form1.
    ' </summary>
    Public Class Form1
        Inherits System.Windows.Forms.Form
        Private WithEvents directorySearcher As DirectorySearcher
        Private searchText As System.Windows.Forms.TextBox
        Private searchLabel As System.Windows.Forms.Label
        Private WithEvents searchButton As System.Windows.Forms.Button

```

```

Public Sub New()
    '
    ' Required for Windows Forms designer support.
    '

    InitializeComponent()
    '
    ' Add any constructor code after InitializeComponent call here.
    '

End Sub

#Region "Windows Form Designer generated code"
' <summary>
'     Required method for designer support. Do not modify
'     the contents of this method with the code editor.
' </summary>
Private Sub InitializeComponent()
    Me.directorySearcher = New Microsoft.Samples.DirectorySearcher.DirectorySearcher()
    Me.searchButton = New System.Windows.Forms.Button()
    Me.searchText = New System.Windows.Forms.TextBox()
    Me.searchLabel = New System.Windows.Forms.Label()
    Me.directorySearcher.Anchor = System.Windows.Forms.AnchorStyles.Top Or
System.Windows.Forms.AnchorStyles.Bottom Or System.Windows.Forms.AnchorStyles.Left Or
System.Windows.Forms.AnchorStyles.Right
    Me.directorySearcher.Location = New System.Drawing.Point(8, 72)
    Me.directorySearcher.SearchCriteria = Nothing
    Me.directorySearcher.Size = New System.Drawing.Size(271, 173)
    Me.directorySearcher.TabIndex = 2
    Me.searchButton.Location = New System.Drawing.Point(8, 16)
    Me.searchButton.Size = New System.Drawing.Size(88, 40)
    Me.searchButton.TabIndex = 0
    Me.searchButton.Text = "&Search"
    Me.searchText.Anchor = System.Windows.Forms.AnchorStyles.Top Or System.Windows.Forms.AnchorStyles.Left
Or System.Windows.Forms.AnchorStyles.Right
    Me.searchText.Location = New System.Drawing.Point(104, 24)
    Me.searchText.Size = New System.Drawing.Size(175, 20)
    Me.searchText.TabIndex = 1
    Me.searchText.Text = "c:\*.*"
    Me.searchLabel.ForeColor = System.Drawing.Color.Red
    Me.searchLabel.Location = New System.Drawing.Point(104, 48)
    Me.searchLabel.Size = New System.Drawing.Size(176, 16)
    Me.searchLabel.TabIndex = 3
    Me.ClientSize = New System.Drawing.Size(291, 264)
    Me.Controls.AddRange(New System.Windows.Forms.Control() {Me.searchLabel, Me.directorySearcher,
Me.searchText, Me.searchButton})
    Me.Text = "Search Directories"
End Sub
#End Region

' <summary>
'     The main entry point for the application.
' </summary>
<STAThread()> _
Shared Sub Main()
    Application.Run(New Form1())
End Sub

Private Sub searchButton_Click(sender As Object, e As System.EventArgs) Handles searchButton.Click
    directorySearcher.SearchCriteria = searchText.Text
    searchLabel.Text = "Searching..."
    directorySearcher.BeginSearch()
End Sub

Private Sub directorySearcher_SearchComplete(sender As Object, e As System.EventArgs) Handles
directorySearcher.SearchComplete
    searchLabel.Text = String.Empty
End Sub
End Class
End Namespace

```

```
namespace SampleUsage
{
    using Microsoft.Samples.DirectorySearcher;
    using System;
    using System.Drawing;
    using System.Collections;
    using System.ComponentModel;
    using System.Windows.Forms;
    using System.Data;

    /// <summary>
    ///     Summary description for Form1.
    /// </summary>
    public class Form1 : System.Windows.Forms.Form
    {
        private DirectorySearcher directorySearcher;
        private System.Windows.Forms.TextBox searchText;
        private System.Windows.Forms.Label searchLabel;
        private System.Windows.Forms.Button searchButton;

        public Form1()
        {
            //
            // Required for Windows Forms designer support.
            //
            InitializeComponent();

            //
            // Add any constructor code after InitializeComponent call here.
            //
        }

        #region Windows Form Designer generated code
        /// <summary>
        ///     Required method for designer support. Do not modify
        ///     the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.directorySearcher = new Microsoft.Samples.DirectorySearcher.DirectorySearcher();
            this.searchButton = new System.Windows.Forms.Button();
            this.searchText = new System.Windows.Forms.TextBox();
            this.searchLabel = new System.Windows.Forms.Label();
            this.directorySearcher.Anchor = (((System.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Bottom)
| System.Windows.Forms.AnchorStyles.Left)
| System.Windows.Forms.AnchorStyles.Right);
            this.directorySearcher.Location = new System.Drawing.Point(8, 72);
            this.directorySearcher.SearchCriteria = null;
            this.directorySearcher.Size = new System.Drawing.Size(271, 173);
            this.directorySearcher.TabIndex = 2;
            this.directorySearcher.SearchComplete += new
System.EventHandler(this.directorySearcher_SearchComplete);
            this.searchButton.Location = new System.Drawing.Point(8, 16);
            this.searchButton.Size = new System.Drawing.Size(88, 40);
            this.searchButton.TabIndex = 0;
            this.searchButton.Text = "&Search";
            this.searchButton.Click += new System.EventHandler(this.searchButton_Click);
            this.searchText.Anchor = ((System.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Left)
| System.Windows.Forms.AnchorStyles.Right);
            this.searchText.Location = new System.Drawing.Point(104, 24);
            this.searchText.Size = new System.Drawing.Size(175, 20);
            this.searchText.TabIndex = 1;
            this.searchText.Text = "c:\\*.cs";
            this.searchLabel.ForeColor = System.Drawing.Color.Red;
        }
    }
}
```

```
this.searchLabel.Location = new System.Drawing.Point(104, 48);
this.searchLabel.Size = new System.Drawing.Size(176, 16);
this.searchLabel.TabIndex = 3;
this.ClientSize = new System.Drawing.Size(291, 264);
this.Controls.AddRange(new System.Windows.Forms.Control[] {this.searchLabel,
    this.directorySearcher,
    this.searchText,
    this.searchButton});
this.Text = "Search Directories";

}

#endregion

/// <summary>
///     The main entry point for the application.
/// </summary>
[STAThread]
static void Main()
{
    Application.Run(new Form1());
}

private void searchButton_Click(object sender, System.EventArgs e)
{
    directorySearcher.SearchCriteria = searchText.Text;
    searchLabel.Text = "Searching...";
    directorySearcher.BeginSearch();
}

private void directorySearcher_SearchComplete(object sender, System.EventArgs e)
{
    searchLabel.Text = string.Empty;
}
}
```

See Also

[BackgroundWorker](#)

[Developing Custom Windows Forms Controls with the .NET Framework](#)

[Event-based Asynchronous Pattern Overview](#)

Windows Forms Controls by Function

5/4/2018 • 4 min to read • [Edit Online](#)

Windows Forms offers controls and components that perform a number of functions. The following table lists the Windows Forms controls and components according to general function. In addition, where multiple controls exist that serve the same function, the recommended control is listed with a note regarding the control it superseded. In a separate subsequent table, the superseded controls are listed with their recommended replacements.

NOTE

The following tables do not list every control or component you can use in Windows Forms; for a more comprehensive list, see [Controls to Use on Windows Forms](#)

Recommended Controls and Components by Function

FUNCTION	CONTROL	DESCRIPTION
Data display	DataGridView control	The DataGridView control provides a customizable table for displaying data. The DataGridView class enables customization of cells, rows, columns, and borders. Note: The DataGridView control provides numerous basic and advanced features that are missing in the DataGrid control. For more information, see Differences Between the Windows Forms DataGridView and DataGrid Controls
Data binding and navigation	BindingSource component	Simplifies binding controls on a form to data by providing currency management, change notification, and other services.
	BindingNavigator control	Provides a toolbar-type interface to navigate and manipulate data on a form.
Text editing	TextBox control	Displays text entered at design time that can be edited by users at run time, or changed programmatically.
	RichTextBox control	Enables text to be displayed with formatting in plain text or rich-text format (RTF).
	MaskedTextBox control	Constrains the format of user input
Information display (read-only)	Label control	Displays text that users cannot directly edit.

FUNCTION	CONTROL	DESCRIPTION
	LinkLabel control	Displays text as a Web-style link and triggers an event when the user clicks the special text. Usually the text is a link to another window or a Web site.
	StatusStrip control	Displays information about the application's current state using a framed area, usually at the bottom of a parent form.
	ProgressBar control	Displays the current progress of an operation to the user.
Web page display	WebBrowser control	Enables the user to navigate Web pages inside your form.
Selection from a list	CheckedListBox control	Displays a scrollable list of items, each accompanied by a check box.
	ComboBox control	Displays a drop-down list of items.
	DomainUpDown control	Displays a list of text items that users can scroll through with up and down buttons.
	ListBox control	Displays a list of text and graphical items (icons).
	ListView control	Displays items in one of four different views. Views include text only, text with small icons, text with large icons, and a details view.
	NumericUpDown control	Displays a list of numerals that users can scroll through with up and down buttons.
	TreeView control	Displays a hierarchical collection of node objects that can consist of text with optional check boxes or icons.
Graphics display	PictureBox control	Displays graphical files, such as bitmaps and icons, in a frame.
Graphics storage	ImageList control	Serves as a repository for images. ImageList controls and the images they contain can be reused from one application to the next.
Value setting	CheckBox control	Displays a check box and a label for text. Generally used to set options.
	CheckedListBox control	Displays a scrollable list of items, each accompanied by a check box.

FUNCTION	CONTROL	DESCRIPTION
	RadioButton control	Displays a button that can be turned on or off.
	TrackBar control	Allows users to set values on a scale by moving a "thumb" along a scale.
Date setting	DateTimePicker control	Displays a graphical calendar to allow users to select a date or a time.
	MonthCalendar control	Displays a graphical calendar to allow users to select a range of dates.
Dialog boxes	ColorDialog control	Displays the color picker dialog box that allows users to set the color of an interface element.
	FontDialog control	Displays a dialog box that allows users to set a font and its attributes.
	OpenFileDialog control	Displays a dialog box that allows users to navigate to and select a file.
	PrintDialog control	Displays a dialog box that allows users to select a printer and set its attributes.
	PrintPreviewDialog control	Displays a dialog box that displays how a control PrintDocument component will appear when printed.
	FolderBrowserDialog control	Displays a dialog that allows users to browse, create, and eventually select a folder
	SaveFileDialog control	Displays a dialog box that allows users to save a file.
Menu controls	MenuStrip control	Creates custom menus. Note: The MenuStrip is designed to replace the MainMenu control.
	ContextMenuStrip control	Creates custom context menus. Note: The ContextMenuStrip is designed to replace the ContextMenu control.
Commands	Button control	Starts, stops, or interrupts a process.
	LinkLabel control	Displays text as a Web-style link and triggers an event when the user clicks the special text. Usually the text is a link to another window or a Web site.

FUNCTION	CONTROL	DESCRIPTION
	NotifyIcon control	Displays an icon in the status notification area of the taskbar that represents an application running in the background.
	ToolStrip control	Creates toolbars that can have a Microsoft Windows XP, Microsoft Office, Microsoft Internet Explorer, or custom look and feel, with or without themes, and with support for overflow and run-time item reordering. Note: The ToolStrip control is designed to replace the ToolBar control.
User Help	HelpProvider component	Provides pop-up or online Help for controls.
	ToolTip component	Provides a pop-up window that displays a brief description of a control's purpose when the user rests the pointer on the control.
Grouping other controls	Panel control	Groups a set of controls on an unlabeled, scrollable frame.
	GroupBox control	Groups a set of controls (such as radio buttons) on a labeled, nonscrollable frame.
	TabControl control	Provides a tabbed page for organizing and accessing grouped objects efficiently.
	SplitContainer control	Provides two panels separated by a movable bar. Note: The SplitContainer control is designed to replace the Splitter control.
	TableLayoutPanel control	Represents a panel that dynamically lays out its contents in a grid composed of rows and columns.
	FlowLayoutPanel control	Represents a panel that dynamically lays out its contents horizontally or vertically.
Audio	SoundPlayer control	Plays sound files in the .wav format. Sounds can be loaded or played asynchronously.

Superseded Controls and Components by Function

FUNCTION	SUPERSEDED CONTROL	RECOMMENDED REPLACEMENT
Data display	DataGrid	DataGridView

FUNCTION	SUPERSEDED CONTROL	RECOMMENDED REPLACEMENT
Information Display (Read-only controls)	StatusBar	StatusStrip
Menu controls	ContextMenu	ContextMenuStrip
	MainMenu	MenuStrip
Commands	ToolBar	ToolStrip
	StatusBar	StatusStrip
Form layout	Splitter	SplitContainer

See Also

[Controls to Use on Windows Forms](#)

[Developing Custom Windows Forms Controls with the .NET Framework](#)

Developing Windows Forms Controls at Design Time

5/4/2018 • 3 min to read • [Edit Online](#)

For control authors, the .NET Framework provides a wealth of control authoring technology. Authors are no longer limited to designing composite controls that act as a collection of preexisting controls. Through inheritance, you can create your own controls from preexisting composite controls or preexisting Windows Forms controls. You can also design your own controls that implement custom painting. These options enable a great deal of flexibility to the design and functionality of the visual interface. To take advantage of these features, you should be familiar with object-based programming concepts.

NOTE

It is not necessary to have a thorough understanding of inheritance, but you may find it useful to refer to [Inheritance basics \(Visual Basic\)](#).

If you want to create custom controls to use on Web Forms, see [Developing Custom ASP.NET Server Controls](#).

In This Section

[Walkthrough: Authoring a Composite Control with Visual Basic](#)

Shows how to create a simple composite control in Visual Basic.

[Walkthrough: Authoring a Composite Control with Visual C#](#)

Shows how to create a simple composite control in C#.

[Walkthrough: Inheriting from a Windows Forms Control with Visual Basic](#)

Shows how to create a simple Windows Forms control using inheritance in Visual Basic.

[Walkthrough: Inheriting from a Windows Forms Control with Visual C#](#)

Shows how to create a simple Windows Forms control using inheritance in C#.

[Walkthrough: Performing Common Tasks Using Smart Tags on Windows Forms Controls](#)

Shows how to use the smart tag feature on Windows Forms controls.

[Walkthrough: Serializing Collections of Standard Types with the DesignerSerializationVisibilityAttribute](#)

Shows how to use the `DesignerSerializationVisibilityAttribute.Content` attribute to serialize a collection.

[Walkthrough: Debugging Custom Windows Forms Controls at Design Time](#)

Shows how to debug the design-time behavior of a Windows Forms control.

[Walkthrough: Creating a Windows Forms Control That Takes Advantage of Visual Studio Design-Time Features](#)

Shows how to tightly integrate a composite control into the design environment.

[How to: Author Controls for Windows Forms](#)

Provides an overview of considerations for implementing a Windows Forms control.

[How to: Author Composite Controls](#)

Shows how to create a control by inheriting from a composite control.

[How to: Inherit from the UserControl Class](#)

Provides an overview of the procedure for creating a composite control.

[How to: Inherit from Existing Windows Forms Controls](#)

Shows how to create an extended control by inheriting from the [Button](#) control class.

[How to: Inherit from the Control Class](#)

Provides an overview of creating an extended control.

[How to: Align a Control to the Edges of Forms at Design Time](#)

Shows how to use the [Dock](#) property to align your control to the edge of the form it occupies.

[How to: Display a Control in the Choose Toolbox Items Dialog Box](#)

Shows the procedure for installing your control so that it appears in the **Customize Toolbox** dialog box.

[How to: Provide a Toolbox Bitmap for a Control](#)

Shows how to use the [ToolboxBitmapAttribute](#) to display an icon next to your custom control in the **Toolbox**.

[How to: Test the Run-Time Behavior of a UserControl](#)

Shows how to use the **UserControl Test Container** to test the behavior of a composite control.

[Design-Time Errors in the Windows Forms Designer](#)

Explains the meaning and use of the Design-Time Error List that appears in Microsoft Visual Studio when the Windows Forms designer fails to load.

[Troubleshooting Control and Component Authoring](#)

Shows how to diagnose and fix common issues that can occur when you author a custom component or control.

Reference

[System.Windows.Forms.Control](#)

Describes this class and has links to all of its members.

[System.Windows.Forms.UserControl](#)

Describes this class and has links to all of its members.

Related Sections

[Developing Custom Windows Forms Controls with the .NET Framework](#)

Discusses how to create your own custom controls with the .NET Framework.

[Language Independence and Language-Independent Components](#)

Introduces the common language runtime, which is designed to simplify the creation and use of components. An important aspect of this simplification is enhanced interoperability between components written using different programming languages. The Common Language Specification (CLS) makes it possible to create tools and components that work with multiple programming languages.

[Walkthrough: Automatically Populating the Toolbox with Custom Components](#)

Describes how to enable your component or control to be displayed in the **Customize Toolbox** dialog box.

Walkthrough: Authoring a Composite Control with Visual Basic

5/4/2018 • 15 min to read • [Edit Online](#)

Composite controls provide a means by which custom graphical interfaces can be created and reused. A composite control is essentially a component with a visual representation. As such, it might consist of one or more Windows Forms controls, components, or blocks of code that can extend functionality by validating user input, modifying display properties, or performing other tasks required by the author. Composite controls can be placed on Windows Forms in the same manner as other controls. In the first part of this walkthrough, you create a simple composite control called `ctlClock`. In the second part of the walkthrough, you extend the functionality of `ctlClock` through inheritance.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

Creating the Project

When you create a new project, you specify its name to set the root namespace, assembly name, and project name, and ensure that the default component will be in the correct namespace.

To create the `ctlClockLib` control library and the `ctlClock` control

1. On the **File** menu, point to **New**, and then click **Project** to open the **New Project** dialog box.
2. From the list of Visual Basic projects, select the **Windows Control Library** project template, type `ctlClockLib` in the **Name** box, and then click **OK**.

The project name, `ctlClockLib`, is also assigned to the root namespace by default. The root namespace is used to qualify the names of components in the assembly. For example, if two assemblies provide components named `ctlClock`, you can specify your `ctlClock` component using `ctlClockLib.ctlClock`.

3. In Solution Explorer, right-click **UserControl1.vb**, and then click **Rename**. Change the file name to `ctlClock.vb`. Click the **Yes** button when you are asked if you want to rename all references to the code element "UserControl1".

NOTE

By default, a composite control inherits from the `UserControl` class provided by the system. The `UserControl` class provides functionality required by all composite controls, and implements standard methods and properties.

4. On the **File** menu, click **Save All** to save the project.

Adding Windows Controls and Components to the Composite Control

A visual interface is an essential part of your composite control. This visual interface is implemented by the addition of one or more Windows controls to the designer surface. In the following demonstration, you will incorporate Windows controls into your composite control and write code to implement functionality.

To add a Label and a Timer to your composite control

1. In Solution Explorer, right-click **ctlClock.vb**, and then click **View Designer**.
2. In the Toolbox, expand the **Common Controls** node, and then double-click **Label**.
A **Label** control named **Label1** is added to your control on the designer surface.
3. In the designer, click **Label1**. In the Properties window, set the following properties.

PROPERTY	CHANGE TO
Name	lblDisplay
Text	(blank space)
TextAlign	MiddleCenter
FontSize	14

4. In the **Toolbox**, expand the **Components** node, and then double-click **Timer**.

Because a **Timer** is a component, it has no visual representation at run time. Therefore, it does not appear with the controls on the designer surface, but rather in the Component Designer (a tray at the bottom of the designer surface).

5. In the Component Designer, click **Timer1**, and then set the **Interval** property to **1000** and the **Enabled** property to **True**.

The **Interval** property controls the frequency with which the timer component ticks. Each time **Timer1** ticks, it runs the code in the **Timer1_Tick** event. The interval represents the number of milliseconds between ticks.

6. In the Component Designer, double-click **Timer1** to go to the **Timer1_Tick** event for **ctlClock**.
7. Modify the code so that it resembles the following code sample. Be sure to change the access modifier from **Private** to **Protected**.

```
Protected Sub Timer1_Tick(ByVal sender As Object, ByVal e As _
    System.EventArgs) Handles Timer1.Tick
    ' Causes the label to display the current time.
    lblDisplay.Text = Format(Now, "hh:mm:ss")
End Sub
```

This code will cause the current time to be shown in **lblDisplay**. Because the interval of **Timer1** was set to **1000**, this event will occur every thousand milliseconds, thus updating the current time every second.

8. Modify the method to be overridable. For more information, see the "Inheriting from a User Control" section below.

```
Protected Overridable Sub Timer1_Tick(ByVal sender As Object, ByVal _
    e As System.EventArgs) Handles Timer1.Tick
```

9. On the **File** menu, click **Save All** to save the project.

Adding Properties to the Composite Control

Your clock control now encapsulates a [Label](#) control and a [Timer](#) component, each with its own set of inherent properties. While the individual properties of these controls will not be accessible to subsequent users of your control, you can create and expose custom properties by writing the appropriate blocks of code. In the following procedure, you will add properties to your control that enable the user to change the color of the background and text.

To add a property to your composite control

1. In Solution Explorer, right-click **ctlClock.vb**, and then click **View Code**.

The Code Editor for your control opens.

2. Locate the `Public Class ctlClock` statement. Beneath it, type the following code.

```
Private colFColor as Color  
Private colBColor as Color
```

These statements create the private variables that you will use to store the values for the properties you are about to create.

3. Insert the following code beneath the variable declarations from step 2.

```
' Declares the name and type of the property.  
Property ClockBackColor() as Color  
    ' Retrieves the value of the private variable colBColor.  
    Get  
        Return colBColor  
    End Get  
    ' Stores the selected value in the private variable colBColor, and  
    ' updates the background color of the label control lblDisplay.  
    Set(ByVal value as Color)  
        colBColor = value  
        lblDisplay.BackColor = colBColor  
    End Set  
  
End Property  
' Provides a similar set of instructions for the foreground color.  
Property ClockForeColor() as Color  
    Get  
        Return colFColor  
    End Get  
    Set(ByVal value as Color)  
        colFColor = value  
        lblDisplay.ForeColor = colFColor  
    End Set  
End Property
```

The preceding code makes two custom properties, `ClockForeColor` and `ClockBackColor`, available to subsequent users of this control by invoking the `Property` statement. The `Get` and `Set` statements provide for storage and retrieval of the property value, as well as code to implement functionality appropriate to the property.

4. On the **File** menu, click **Save All** to save the project.

Testing the Control

Controls are not stand-alone projects; they must be hosted in a container. Test your control's run-time behavior and exercise its properties with the **UserControl Test Container**. For more information, see [How to: Test the Run-Time Behavior of a UserControl](#).

To test your control

1. Press F5 to build the project and run your control in the **UserControl Test Container**.
2. In the test container's property grid, select the `ClockBackColor` property, and then click the drop-down arrow to display the color palette.
3. Choose a color by clicking it.

The background color of your control changes to the color you selected.

4. Use a similar sequence of events to verify that the `ClockForeColor` property is functioning as expected.
5. Click **Close** to close the **UserControl Test Container**.

In this section and the preceding sections, you have seen how components and Windows controls can be combined with code and packaging to provide custom functionality in the form of a composite control. You have learned to expose properties in your composite control, and how to test your control after it is complete. In the next section you will learn how to construct an inherited composite control using `ctlClock` as a base.

Inheriting from a Composite Control

In the previous sections, you learned how to combine Windows controls, components, and code into reusable composite controls. Your composite control can now be used as a base upon which other controls can be built. The process of deriving a class from a base class is called *inheritance*. In this section, you will create a composite control called `ctlAlarmClock`. This control will be derived from its parent control, `ctlClock`. You will learn to extend the functionality of `ctlClock` by overriding parent methods and adding new methods and properties.

The first step in creating an inherited control is to derive it from its parent. This action creates a new control that has all of the properties, methods, and graphical characteristics of the parent control, but can also act as a base for the addition of new or modified functionality.

To create the inherited control

1. In Solution Explorer, right-click `ctlClockLib`, point to **Add**, and then click **User Control**.

The **Add New Item** dialog box opens.

2. Select the **Inherited User Control** template.
 3. In the **Name** box, type `ctlAlarmClock.vb`, and then click **Add**.
- The **Inheritance Picker** dialog box appears.
4. Under **Component Name**, double-click `ctlClock`.
 5. In Solution Explorer, browse through the current projects.

NOTE

A file called `ctlAlarmClock.vb` has been added to the current project.

Adding the Alarm Properties

Properties are added to an inherited control in the same way they are added to a composite control. You will now use the property declaration syntax to add two properties to your control: `AlarmTime`, which will store the value of the date and time the alarm is to go off, and `AlarmSet`, which will indicate whether the alarm is set.

To add properties to your composite control

1. In Solution Explorer, right-click `ctlAlarmClock`, and then click **View Code**.
2. Locate the class declaration for the `ctlAlarmClock` control, which appears as `Public Class ctlAlarmClock`. In

the class declaration, insert the following code.

```
Private dteAlarmTime As Date
Private blnAlarmSet As Boolean
' These properties will be declared as Public to allow future
' developers to access them.
Public Property AlarmTime() As Date
    Get
        Return dteAlarmTime
    End Get
    Set(ByVal value as Date)
        dteAlarmTime = value
    End Set
End Property
Public Property AlarmSet() As Boolean
    Get
        Return blnAlarmSet
    End Get
    Set(ByVal value as Boolean)
        blnAlarmSet = value
    End Set
End Property
```

Adding to the Graphical Interface of the Control

Your inherited control has a visual interface that is identical to the control it inherits from. It possesses the same constituent controls as its parent control, but the properties of the constituent controls will not be available unless they were specifically exposed. You may add to the graphical interface of an inherited composite control in the same manner as you would add to any composite control. To continue adding to your alarm clock's visual interface, you will add a label control that will flash when the alarm is sounding.

To add the label control

1. In Solution Explorer, right-click **ctlAlarmClock**, and click **View Designer**.

The designer for **ctlAlarmClock** opens in the main window.

2. Click **lblDisplay** (the display portion of the control), and view the Properties window.

NOTE

While all the properties are displayed, they are dimmed. This indicates that these properties are native to **lblDisplay** and cannot be modified or accessed in the Properties window. By default, controls contained in a composite control are **Private**, and their properties are not accessible by any means.

NOTE

If you want subsequent users of your composite control to have access to its internal controls, declare them as **Public** or **Protected**. This will allow you to set and modify properties of controls contained within your composite control by using the appropriate code.

3. Add a **Label** control to your composite control.

4. Using the mouse, drag the **Label** control immediately beneath the display box. In the Properties window, set the following properties.

PROPERTY	SETTING
Name	lblAlarm

PROPERTY	SETTING
Text	Alarm!
TextAlign	MiddleCenter
Visible	False

Adding the Alarm Functionality

In the previous procedures, you added properties and a control that will enable alarm functionality in your composite control. In this procedure, you will add code to compare the current time to the alarm time and, if they are the same, to sound and flash an alarm. By overriding the `Timer1_Tick` method of `ctlClock` and adding additional code to it, you will extend the capability of `ctlAlarmClock` while retaining all of the inherent functionality of `ctlClock`.

To override the `Timer1_Tick` method of `ctlClock`

1. In Solution Explorer, right-click `ctlAlarmClock.vb`, and then click **View Code**.
2. Locate the `Private blnAlarmSet As Boolean` statement. Immediately beneath it, add the following statement.

```
Dim blnColorTicker as Boolean
```

3. Locate the `End Class` statement at the bottom of the page. Just before the `End Class` statement, add the following code.

```
Protected Overrides Sub Timer1_Tick(ByVal sender As Object, ByVal e _  
As System.EventArgs)  
    ' Calls the Timer1_Tick method of ctlClock.  
    MyBase.Timer1_Tick(sender, e)  
    ' Checks to see if the Alarm is set.  
    If AlarmSet = False Then  
        Exit Sub  
    End If  
    ' If the date, hour, and minute of the alarm time are the same as  
    ' now, flash and beep an alarm.  
    If AlarmTime.Date = Now.Date And AlarmTime.Hour = Now.Hour And _  
        AlarmTime.Minute = Now.Minute Then  
        ' Sounds an audible beep.  
        Beep()  
        ' Sets lblAlarmVisible to True, and changes the background color based on the  
        ' value of blnColorTicker. The background color of the label will  
        ' flash once per tick of the clock.  
        lblAlarm.Visible = True  
        If blnColorTicker = False Then  
            lblAlarm.BackColor = Color.PeachPuff  
            blnColorTicker = True  
        Else  
            lblAlarm.BackColor = Color.Plum  
            blnColorTicker = False  
        End If  
    Else  
        ' Once the alarm has sounded for a minute, the label is made  
        ' invisible again.  
        lblAlarm.Visible = False  
    End If  
End Sub
```

The addition of this code accomplishes several tasks. The `Overrides` statement directs the control to use this method in place of the method that was inherited from the base control. When this method is called, it

calls the method it overrides by invoking the `MyBase.Timer1_Tick` statement, ensuring that all of the functionality incorporated in the original control is reproduced in this control. It then runs additional code to incorporate the alarm functionality. A flashing label control will appear when the alarm occurs, and an audible beep will be heard.

NOTE

Because you are overriding an inherited event handler, you do not have to specify the event with the `Handles` keyword. The event is already hooked up. All you are overriding is the implementation of the handler.

Your alarm clock control is almost complete. The only thing that remains is to implement a way to turn it off. To do this, you will add code to the `lblAlarm_Click` method.

To implement the shutoff method

1. In Solution Explorer, right-click **ctlAlarmClock.vb**, and then click **View Designer**.
2. In the designer, double-click **lblAlarm**. The **Code Editor** opens to the `Private Sub lblAlarm_Click` line.
3. Modify this method so that it resembles the following code.

```
Private Sub lblAlarm_Click(ByVal sender As Object, ByVal e As _  
    System.EventArgs) Handles lblAlarm.Click  
    ' Turns off the alarm.  
    AlarmSet = False  
    ' Hides the flashing label.  
    lblAlarm.Visible = False  
End Sub
```

4. On the **File** menu, click **Save All** to save the project.

Using the Inherited Control on a Form

You can test your inherited control the same way you tested the base class control, `ctlClock`: Press F5 to build the project and run your control in the **UserControl Test Container**. For more information, see [How to: Test the Run-Time Behavior of a UserControl](#).

To put your control to use, you will need to host it on a form. As with a standard composite control, an inherited composite control cannot stand alone and must be hosted in a form or other container. Since `ctlAlarmClock` has a greater depth of functionality, additional code is required to test it. In this procedure, you will write a simple program to test the functionality of `ctlAlarmClock`. You will write code to set and display the `AlarmTime` property of `ctlAlarmClock`, and will test its inherent functions.

To build and add your control to a test form

1. In Solution Explorer, right-click **ctlClockLib**, and then click **Build**.
2. On the **File** menu, point to **Add**, and then click **New Project**.
3. Add a new **Windows Application** project to the solution, and name it `Test`.

The **Test** project is added to Solution Explorer.

4. In Solution Explorer, right-click the `Test` project node, and then click **Add Reference** to display the **Add Reference** dialog box.
5. Click the tab labeled **Projects**. The project `ctlClockLib` will be listed under **Project Name**. Double-click `ctlClockLib` to add the reference to the test project.
6. In Solution Explorer, right-click **Test**, and then click **Build**.
7. In the **Toolbox**, expand the **ctlClockLib Components** node.

8. Double-click **ctlAlarmClock** to add an instance of `ctlAlarmClock` to your form.
9. In the **Toolbox**, locate and double-click **DateTimePicker** to add a **DateTimePicker** control to your form, and then add a **Label** control by double-clicking **Label**.
10. Use the mouse to position the controls in a convenient place on the form.
11. Set the properties of these controls in the following manner.

CONTROL	PROPERTY	VALUE
<code>label1</code>	Text	(blank space)
	Name	<code>lblTest</code>
<code>dateTimePicker1</code>	Name	<code>dtpTest</code>
	Format	Time

12. In the designer, double-click **dtpTest**.

The **Code Editor** opens to `Private Sub dtpTest_ValueChanged`.

13. Modify the code so that it resembles the following.

```
Private Sub dtpTest_ValueChanged(ByVal sender As Object, ByVal e As _
    System.EventArgs) Handles dtpTest.ValueChanged
    ctlAlarmClock1.AlarmTime = dtpTest.Value
    ctlAlarmClock1.AlarmSet = True
    lblTest.Text = "Alarm Time is " & Format(ctlAlarmClock1.AlarmTime, _
        "hh:mm")
End Sub
```

14. In Solution Explorer, right-click **Test**, and then click **Set as StartUp Project**.

15. On the **Debug** menu, click **Start Debugging**.

The test program starts. Note that the current time is updated in the `ctlAlarmClock` control, and that the starting time is shown in the **DateTimePicker** control.

16. Click the **DateTimePicker** where the minutes of the hour are displayed.

17. Using the keyboard, set a value for minutes that is one minute greater than the current time shown by `ctlAlarmClock`.

The time for the alarm setting is shown in `lblTest`. Wait for the displayed time to reach the alarm setting time. When the displayed time reaches the time to which the alarm is set, the beep will sound and `lblAlarm` will flash.

18. Turn off the alarm by clicking `lblAlarm`. You may now reset the alarm.

This walkthrough has covered a number of key concepts. You have learned to create a composite control by combining controls and components into a composite control container. You have learned to add properties to your control, and to write code to implement custom functionality. In the last section, you learned to extend the functionality of a given composite control through inheritance, and to alter the functionality of host methods by overriding those methods.

See Also

[Varieties of Custom Controls](#)

[How to: Author Composite Controls](#)

[How to: Display a Control in the Choose Toolbox Items Dialog Box](#)

[Component Authoring Walkthroughs](#)

Walkthrough: Authoring a Composite Control with Visual C#

5/4/2018 • 14 min to read • [Edit Online](#)

Composite controls provide a means by which custom graphical interfaces can be created and reused. A composite control is essentially a component with a visual representation. As such, it might consist of one or more Windows Forms controls, components, or blocks of code that can extend functionality by validating user input, modifying display properties, or performing other tasks required by the author. Composite controls can be placed on Windows Forms in the same manner as other controls. In the first part of this walkthrough, you create a simple composite control called `ctlClock`. In the second part of the walkthrough, you extend the functionality of `ctlClock` through inheritance.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

Creating the Project

When you create a new project, you specify its name to set the root namespace, assembly name, and project name, and ensure that the default component will be in the correct namespace.

To create the `ctlClockLib` control library and the `ctlClock` control

1. On the **File** menu, point to **New**, and then click **Project** to open the **New Project** dialog box.
2. From the list of Visual C# projects, select the **Windows Forms Control Library** project template, type `ctlClockLib` in the **Name** box, and then click **OK**.

The project name, `ctlClockLib`, is also assigned to the root namespace by default. The root namespace is used to qualify the names of components in the assembly. For example, if two assemblies provide components named `ctlClock`, you can specify your `ctlClock` component using `ctlClockLib.ctlClock`.

3. In Solution Explorer, right-click **UserControl1.cs**, and then click **Rename**. Change the file name to `ctlClock.cs`. Click the **Yes** button when you are asked if you want to rename all references to the code element "UserControl1".

NOTE

By default, a composite control inherits from the `UserControl` class provided by the system. The `UserControl` class provides functionality required by all composite controls, and implements standard methods and properties.

4. On the **File** menu, click **Save All** to save the project.

Adding Windows Controls and Components to the Composite Control

A visual interface is an essential part of your composite control. This visual interface is implemented by the addition of one or more Windows controls to the designer surface. In the following demonstration, you will incorporate Windows controls into your composite control and write code to implement functionality.

To add a Label and a Timer to your composite control

1. In Solution Explorer, right-click **ctlClock.cs**, and then click **View Designer**.
2. In the **Toolbox**, expand the **Common Controls** node, and then double-click **Label**.
- A **Label** control named `label1` is added to your control on the designer surface.
3. In the designer, click **label1**. In the Properties window, set the following properties.

PROPERTY	CHANGE TO
Name	<code>lblDisplay</code>
Text	(blank space)
TextAlign	<code>MiddleCenter</code>
Font.Size	<code>14</code>

4. In the **Toolbox**, expand the **Components** node, and then double-click **Timer**.

Because a **Timer** is a component, it has no visual representation at run time. Therefore, it does not appear with the controls on the designer surface, but rather in the **Component Designer** (a tray at the bottom of the designer surface).

5. In the **Component Designer**, click **timer1**, and then set the **Interval** property to `1000` and the **Enabled** property to `true`.

The **Interval** property controls the frequency with which the **Timer** component ticks. Each time **timer1** ticks, it runs the code in the `timer1_Tick` event. The interval represents the number of milliseconds between ticks.

6. In the **Component Designer**, double-click **timer1** to go to the `timer1_Tick` event for **ctlClock**.
7. Modify the code so that it resembles the following code sample. Be sure to change the access modifier from `private` to `protected`.

```
protected void timer1_Tick(object sender, System.EventArgs e)
{
    // Causes the label to display the current time.
    lblDisplay.Text = DateTime.Now.ToString();
}
```

This code will cause the current time to be shown in `lblDisplay`. Because the interval of `timer1` was set to `1000`, this event will occur every thousand milliseconds, thus updating the current time every second.

8. Modify the method to be overridable with the `virtual` keyword. For more information, see the "Inheriting from a User Control" section below.

```
protected virtual void timer1_Tick(object sender, System.EventArgs e)
```

9. On the **File** menu, click **Save All** to save the project.

Adding Properties to the Composite Control

Your clock control now encapsulates a **Label** control and a **Timer** component, each with its own set of inherent

properties. While the individual properties of these controls will not be accessible to subsequent users of your control, you can create and expose custom properties by writing the appropriate blocks of code. In the following procedure, you will add properties to your control that enable the user to change the color of the background and text.

To add a property to your composite control

1. In Solution Explorer, right-click **ctlClock.cs**, and then click **View Code**.

The **Code Editor** for your control opens.

2. Locate the `public partial class ctlClock` statement. Beneath the opening brace (`{`), type the following code.

```
private Color colFColor;
private Color colBColor;
```

These statements create the private variables that you will use to store the values for the properties you are about to create.

3. Type the following code beneath the variable declarations from step 2.

```
// Declares the name and type of the property.
public Color ClockBackColor
{
    // Retrieves the value of the private variable colBColor.
    get
    {
        return colBColor;
    }
    // Stores the selected value in the private variable colBColor, and
    // updates the background color of the label control lblDisplay.
    set
    {
        colBColor = value;
        lblDisplay.BackColor = colBColor;
    }
}
// Provides a similar set of instructions for the foreground color.
public Color ClockForeColor
{
    get
    {
        return colFColor;
    }
    set
    {
        colFColor = value;
        lblDisplay.ForeColor = colFColor;
    }
}
```

The preceding code makes two custom properties, `ClockForeColor` and `ClockBackColor`, available to subsequent users of this control. The `get` and `set` statements provide for storage and retrieval of the property value, as well as code to implement functionality appropriate to the property.

4. On the **File** menu, click **Save All** to save the project.

Testing the Control

Controls are not stand-alone applications; they must be hosted in a container. Test your control's run-time behavior and exercise its properties with the **UserControl Test Container**. For more information, see [How to: Test a User Control](#).

Test the Run-Time Behavior of a UserControl.

To test your control

1. Press F5 to build the project and run your control in the **UserControl Test Container**.
2. In the test container's property grid, locate the `ClockBackColor` property, and then select the property to display the color palette.
3. Choose a color by clicking it.

The background color of your control changes to the color you selected.

4. Use a similar sequence of events to verify that the `ClockForeColor` property is functioning as expected.

In this section and the preceding sections, you have seen how components and Windows controls can be combined with code and packaging to provide custom functionality in the form of a composite control. You have learned to expose properties in your composite control, and how to test your control after it is complete. In the next section you will learn how to construct an inherited composite control using `ctlClock` as a base.

Inheriting from a Composite Control

In the previous sections, you learned how to combine Windows controls, components, and code into reusable composite controls. Your composite control can now be used as a base upon which other controls can be built. The process of deriving a class from a base class is called *inheritance*. In this section, you will create a composite control called `ctlAlarmClock`. This control will be derived from its parent control, `ctlClock`. You will learn to extend the functionality of `ctlClock` by overriding parent methods and adding new methods and properties.

The first step in creating an inherited control is to derive it from its parent. This action creates a new control that has all of the properties, methods, and graphical characteristics of the parent control, but can also act as a base for the addition of new or modified functionality.

To create the inherited control

1. In Solution Explorer, right-click **ctlClockLib**, point to **Add**, and then click **User Control**.

The **Add New Item** dialog box opens.

2. Select the **Inherited User Control** template.

3. In the **Name** box, type `ctlAlarmClock.cs`, and then click **Add**.

The **Inheritance Picker** dialog box appears.

4. Under **Component Name**, double-click **ctlClock**.

5. In Solution Explorer, browse through the current projects.

NOTE

A file called `ctlAlarmClock.cs` has been added to the current project.

Adding the Alarm Properties

Properties are added to an inherited control in the same way they are added to a composite control. You will now use the property declaration syntax to add two properties to your control: `AlarmTime`, which will store the value of the date and time the alarm is to go off, and `AlarmSet`, which will indicate whether the alarm is set.

To add properties to your composite control

1. In Solution Explorer, right-click **ctlAlarmClock**, and then click **View Code**.

2. Locate the `public class` statement. Note that your control inherits from `ctlClockLib.ctlClock`. Beneath the opening brace (`{`) statement, type the following code.

```
private DateTime dteAlarmTime;
private bool blnAlarmSet;
// These properties will be declared as public to allow future
// developers to access them.
public DateTime AlarmTime
{
    get
    {
        return dteAlarmTime;
    }
    set
    {
        dteAlarmTime = value;
    }
}
public bool AlarmSet
{
    get
    {
        return blnAlarmSet;
    }
    set
    {
        blnAlarmSet = value;
    }
}
```

Adding to the Graphical Interface of the Control

Your inherited control has a visual interface that is identical to the control it inherits from. It possesses the same constituent controls as its parent control, but the properties of the constituent controls will not be available unless they were specifically exposed. You may add to the graphical interface of an inherited composite control in the same manner as you would add to any composite control. To continue adding to your alarm clock's visual interface, you will add a label control that will flash when the alarm is sounding.

To add the label control

1. In Solution Explorer, right-click **ctlAlarmClock**, and then click **View Designer**.

The designer for `ctlAlarmClock` opens in the main window.

2. Click the display portion of the control, and view the Properties window.

NOTE

While all the properties are displayed, they are dimmed. This indicates that these properties are native to `lblDisplay` and cannot be modified or accessed in the Properties window. By default, controls contained in a composite control are `private`, and their properties are not accessible by any means.

NOTE

If you want subsequent users of your composite control to have access to its internal controls, declare them as `public` or `protected`. This will allow you to set and modify properties of controls contained within your composite control by using the appropriate code.

3. Add a **Label** control to your composite control.
4. Using the mouse, drag the **Label** control immediately beneath the display box. In the Properties window, set

the following properties.

PROPERTY	SETTING
Name	lblAlarm
Text	Alarm!
TextAlign	MiddleCenter
Visible	false

Adding the Alarm Functionality

In the previous procedures, you added properties and a control that will enable alarm functionality in your composite control. In this procedure, you will add code to compare the current time to the alarm time and, if they are the same, to flash an alarm. By overriding the `timer1_Tick` method of `ctlClock` and adding additional code to it, you will extend the capability of `ctlAlarmClock` while retaining all of the inherent functionality of `ctlClock`.

To override the `timer1_Tick` method of `ctlClock`

1. In the **Code Editor**, locate the `private bool bInAlarmSet;` statement. Immediately beneath it, add the following statement.

```
private bool bInColorTicker;
```

2. In the **Code Editor**, locate the closing brace (`}`) at the end of the class. Just before the brace, add the following code.

```

protected override void timer1_Tick(object sender, System.EventArgs e)
{
    // Calls the Timer1_Tick method of ctlClock.
    base.timer1_Tick(sender, e);
    // Checks to see if the alarm is set.
    if (AlarmSet == false)
        return;
    else
        // If the date, hour, and minute of the alarm time are the same as
        // the current time, flash an alarm.
    {
        if (AlarmTime.Date == DateTime.Now.Date && AlarmTime.Hour ==
            DateTime.Now.Hour && AlarmTime.Minute == DateTime.Now.Minute)
        {
            // Sets lblAlarmVisible to true, and changes the background color based on
            // the value of blnColorTicker. The background color of the label
            // will flash once per tick of the clock.
            lblAlarm.Visible = true;
            if (blnColorTicker == false)
            {
                lblAlarm.BackColor = Color.Red;
                blnColorTicker = true;
            }
            else
            {
                lblAlarm.BackColor = Color.Blue;
                blnColorTicker = false;
            }
        }
        else
        {
            // Once the alarm has sounded for a minute, the label is made
            // invisible again.
            lblAlarm.Visible = false;
        }
    }
}

```

The addition of this code accomplishes several tasks. The `override` statement directs the control to use this method in place of the method that was inherited from the base control. When this method is called, it calls the method it overrides by invoking the `base.timer1_Tick` statement, ensuring that all of the functionality incorporated in the original control is reproduced in this control. It then runs additional code to incorporate the alarm functionality. A flashing label control will appear when the alarm occurs.

Your alarm clock control is almost complete. The only thing that remains is to implement a way to turn it off. To do this, you will add code to the `lblAlarm_Click` method.

To implement the shutoff method

1. In Solution Explorer, right-click **ctlAlarmClock.cs**, and then click **View Designer**.

The designer opens.

2. Add a button to the control. Set the properties of the button as follows.

PROPERTY	VALUE
Name	<code>btnAlarmOff</code>
Text	Disable Alarm

3. In the designer, double-click **btnAlarmOff**.

The **Code Editor** opens to the `private void btnAlarmOff_Click` line.

4. Modify this method so that it resembles the following code.

```
private void btnAlarmOff_Click(object sender, System.EventArgs e)
{
    // Turns off the alarm.
    AlarmSet = false;
    // Hides the flashing label.
    lblAlarm.Visible = false;
}
```

5. On the **File** menu, click **Save All** to save the project.

Using the Inherited Control on a Form

You can test your inherited control the same way you tested the base class control, `ctlClock`. Press F5 to build the project and run your control in the **UserControl Test Container**. For more information, see [How to: Test the Run-Time Behavior of a UserControl](#).

To put your control to use, you will need to host it on a form. As with a standard composite control, an inherited composite control cannot stand alone and must be hosted in a form or other container. Since `ctlAlarmClock` has a greater depth of functionality, additional code is required to test it. In this procedure, you will write a simple program to test the functionality of `ctlAlarmClock`. You will write code to set and display the `AlarmTime` property of `ctlAlarmClock`, and will test its inherent functions.

To build and add your control to a test form

1. In Solution Explorer, right-click **ctlClockLib**, and then click **Build**.
2. Add a new **Windows Application** project to the solution, and name it `Test`.
3. In Solution Explorer, right-click the **References** node for your test project. Click **Add Reference** to display the **Add Reference** dialog box. Click the tab labeled **Projects**. Your `ctlClockLib` project will be listed under **Project Name**. Double-click the project to add the reference to the test project.
4. In Solution Explorer, right-click **Test**, and then click **Build**.
5. In the **Toolbox**, expand the **ctlClockLib Components** node.
6. Double-click **ctlAlarmClock** to add a copy of `ctlAlarmClock` to your form.
7. In the **Toolbox**, locate and double-click **DateTimePicker** to add a **DateTimePicker** control to your form, and then add a **Label** control by double-clicking **Label**.
8. Use the mouse to position the controls in a convenient place on the form.
9. Set the properties of these controls in the following manner.

CONTROL	PROPERTY	VALUE
<code>label1</code>	Text	(blank space)
	Name	<code>lblTest</code>
<code>dateTimePicker1</code>	Name	<code>dtpTest</code>
	Format	Time

10. In the designer, double-click `dtpTest`.

The **Code Editor** opens to `private void dtpTest_ValueChanged`.

11. Modify the code so that it resembles the following.

```
private void dtpTest_ValueChanged(object sender, System.EventArgs e)
{
    ctlAlarmClock1.AlarmTime = dtpTest.Value;
    ctlAlarmClock1.AlarmSet = true;
    lblTest.Text = "Alarm Time is " +
        ctlAlarmClock1.AlarmTime.ToShortTimeString();
}
```

12. In Solution Explorer, right-click **Test**, and then click **Set as StartUp Project**.

13. On the **Debug** menu, click **Start Debugging**.

The test program starts. Note that the current time is updated in the `ctlAlarmClock` control, and that the starting time is shown in the `DateTimePicker` control.

14. Click the `DateTimePicker` where the minutes of the hour are displayed.

15. Using the keyboard, set a value for minutes that is one minute greater than the current time shown by `ctlAlarmClock`.

The time for the alarm setting is shown in `lblTest`. Wait for the displayed time to reach the alarm setting time. When the displayed time reaches the time to which the alarm is set, the `lblAlarm` will flash.

16. Turn off the alarm by clicking `btnAlarmOff`. You may now reset the alarm.

This walkthrough has covered a number of key concepts. You have learned to create a composite control by combining controls and components into a composite control container. You have learned to add properties to your control, and to write code to implement custom functionality. In the last section, you learned to extend the functionality of a given composite control through inheritance, and to alter the functionality of host methods by overriding those methods.

See Also

[Varieties of Custom Controls](#)

[Programming with Components](#)

[Component Authoring Walkthroughs](#)

[How to: Display a Control in the Choose Toolbox Items Dialog Box](#)

[Walkthrough: Inheriting from a Windows Forms Control with Visual C#](#)

Walkthrough: Inheriting from a Windows Forms Control with Visual Basic

5/4/2018 • 5 min to read • [Edit Online](#)

With Visual Basic, you can create powerful custom controls through *inheritance*. Through inheritance you are able to create controls that retain all of the inherent functionality of standard Windows Forms controls but also incorporate custom functionality. In this walkthrough, you will create a simple inherited control called `ValueButton`. This button will inherit functionality from the standard Windows Forms `Button` control, and will expose a custom property called `ButtonValue`.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

Creating the Project

When you create a new project, you specify its name in order to set the root namespace, assembly name, and project name, and to ensure that the default component will be in the correct namespace.

To create the `ValueButtonLib` control library and the `ValueButton` control

1. On the **File** menu, point to **New** and then click **Project** to open the **New Project** dialog box.
2. Select the **Windows Forms Control Library** project template from the list of Visual Basic projects, and type `ValueButtonLib` in the **Name** box.

The project name, `ValueButtonLib`, is also assigned to the root namespace by default. The root namespace is used to qualify the names of components in the assembly. For example, if two assemblies provide components named `ValueButton`, you can specify your `ValueButton` component using `ValueButtonLib.ValueButton`. For more information, see [Namespaces in Visual Basic](#).

3. In **Solution Explorer**, right-click `UserControl1.vb`, then choose **Rename** from the shortcut menu. Change the file name to `ValueButton.vb`. Click the **Yes** button when you are asked if you want to rename all references to the code element 'UserControl1'.
4. In **Solution Explorer**, click the **Show All Files** button.
5. Open the `ValueButton.vb` node to display the designer-generated code file, `ValueButton.Designer.vb`. Open this file in the **Code Editor**.
6. Locate the `Class` statement, `Partial Public Class ValueButton`, and change the type from which this control inherits from `UserControl` to `Button`. This allows your inherited control to inherit all the functionality of the `Button` control.
7. Locate the `InitializeComponent` method and remove the line that assigns the `AutoSizeMode` property. This property does not exist in the `Button` control.
8. From the **File** menu, choose **Save All** to save the project.

Note that a visual designer is no longer available. Because the `Button` control does its own painting, you are unable to modify its appearance in the designer. Its visual representation will be exactly the same as that of

the class it inherits from (that is, [Button](#)) unless modified in the code.

NOTE

You can still add components, which have no UI elements, to the design surface.

Adding a Property to Your Inherited Control

One possible use of inherited Windows Forms controls is the creation of controls that are identical in appearance and behavior (look and feel) to standard Windows Forms controls, but expose custom properties. In this section, you will add a property called `ButtonValue` to your control.

To add the Value property

1. In **Solution Explorer**, right-click **ValueButton.vb**, and then click **View Code** from the shortcut menu.
2. Locate the `Public Class ValueButton` statement. Immediately beneath this statement, type the following code:

```
' Creates the private variable that will store the value of your
' property.
Private varValue As Integer
' Declares the property.
Property ButtonValue() As Integer
    ' Sets the method for retrieving the value of your property.
    Get
        Return varValue
    End Get
    ' Sets the method for setting the value of your property.
    Set(ByVal Value As Integer)
        varValue = Value
    End Set
End Property
```

This code sets the methods by which the `ButtonValue` property is stored and retrieved. The `Get` statement sets the value returned to the value that is stored in the private variable `varValue`, and the `Set` statement sets the value of the private variable by use of the `Value` keyword.

3. From the **File** menu, choose **Save All** to save the project.

Testing Your Control

Controls are not stand-alone projects; they must be hosted in a container. In order to test your control, you must provide a test project for it to run in. You must also make your control accessible to the test project by building (compiling) it. In this section, you will build your control and test it in a Windows Form.

To build your control

1. On the **Build** menu, click **Build Solution**.

The build should be successful with no compiler errors or warnings.

To create a test project

1. On the **File** menu, point to **Add** and then click **New Project** to open the **Add New Project** dialog box.
2. Select the Visual Basic projects node, and click **Windows Forms Application**.
3. In the **Name** box, type `Test`.
4. In **Solution Explorer**, click the **Show All Files** button.

5. In **Solution Explorer**, right-click the **References** node for your test project, then select **Add Reference** from the shortcut menu to display the **Add Reference** dialog box.
6. Click the **Projects** tab.
7. Click the tab labeled **Projects**. Your `ValueButtonLib` project will be listed under **Project Name**. Double-click the project to add the reference to the test project.
8. In **Solution Explorer**, right-click **Test** and select **Build**.

To add your control to the form

1. In **Solution Explorer**, right-click `Form1.vb` and choose **View Designer** from the shortcut menu.
2. In the **Toolbox**, click **ValueButtonLib Components**. Double-click **ValueButton**.
- A **ValueButton** appears on the form.
3. Right-click the **ValueButton** and select **Properties** from the shortcut menu.
4. In the **Properties** window, examine the properties of this control. Note that they are identical to the properties exposed by a standard button, except that there is an additional property, `ButtonValue`.
5. Set the `ButtonValue` property to `5`.
6. On the **All Windows Forms** tab of the **Toolbox**, double-click **Label** to add a **Label** control to your form.
7. Relocate the label to the center of the form.
8. Double-click `ValueButton1`.

The **Code Editor** opens to the `valueButton1_Click` event.

9. Type the following line of code.

```
Label1.Text = CStr(ValueButton1.ButtonValue)
```

10. In **Solution Explorer**, right-click **Test**, and choose **Set as Startup Project** from the shortcut menu.

11. From the **Debug** menu, select **Start Debugging**.

`Form1` appears.

12. Click `Valuebutton1`.

The numeral '5' is displayed in `Label1`, demonstrating that the `ButtonValue` property of your inherited control has been passed to `Label1` through the `ValueButton1_Click` method. Thus your **ValueButton** control inherits all the functionality of the standard Windows Forms button, but exposes an additional, custom property.

See Also

[Walkthrough: Authoring a Composite Control with Visual Basic](#)

[How to: Display a Control in the Choose Toolbox Items Dialog Box](#)

[Developing Custom Windows Forms Controls with the .NET Framework](#)

[Inheritance basics \(Visual Basic\)](#)

[Component Authoring Walkthroughs](#)

Walkthrough: Inheriting from a Windows Forms Control with Visual C#

5/4/2018 • 5 min to read • [Edit Online](#)

With Visual C# 2005, you can create powerful custom controls through *inheritance*. Through inheritance you are able to create controls that retain all of the inherent functionality of standard Windows Forms controls but also incorporate custom functionality. In this walkthrough, you will create a simple inherited control called `ValueButton`. This button will inherit functionality from the standard Windows Forms `Button` control, and will expose a custom property called `ButtonValue`.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

Creating the Project

When you create a new project, you specify its name in order to set the root namespace, assembly name, and project name, and to ensure that the default component will be in the correct namespace.

To create the `ValueButtonLib` control library and the `ValueButton` control

1. On the **File** menu, point to **New** and then click **Project** to open the **New Project** dialog box.
2. Select the **Windows Forms Control Library** project template from the list of Visual C# Projects, and type `ValueButtonLib` in the **Name** box.

The project name, `ValueButtonLib`, is also assigned to the root namespace by default. The root namespace is used to qualify the names of components in the assembly. For example, if two assemblies provide components named `ValueButton`, you can specify your `ValueButton` component using `ValueButtonLib.ValueButton`. For more information, see [Namespaces](#).

3. In **Solution Explorer**, right-click `UserControl1.cs`, then choose **Rename** from the shortcut menu. Change the file name to `ValueButton.cs`. Click the **Yes** button when you are asked if you want to rename all references to the code element '`userControl1`'.
4. In **Solution Explorer**, right-click `ValueButton.cs` and select **View Code**.
5. Locate the `class` statement line, `public partial class ValueButton`, and change the type from which this control inherits from `UserControl` to `Button`. This allows your inherited control to inherit all the functionality of the `Button` control.
6. In **Solution Explorer**, open the `ValueButton.cs` node to display the designer-generated code file, `ValueButton.Designer.cs`. Open this file in the **Code Editor**.
7. Locate the `InitializeComponent` method and remove the line that assigns the `AutoSizeMode` property. This property does not exist in the `Button` control.
8. From the **File** menu, choose **Save All** to save the project.

NOTE

A visual designer is no longer available. Because the **Button** control does its own painting, you are unable to modify its appearance in the designer. Its visual representation will be exactly the same as that of the class it inherits from (that is, **Button**) unless modified in the code. You can still add components, which have no UI elements, to the design surface.

Adding a Property to Your Inherited Control

One possible use of inherited Windows Forms controls is the creation of controls that are identical in look and feel of standard Windows Forms controls, but expose custom properties. In this section, you will add a property called **ButtonValue** to your control.

To add the Value property

1. In **Solution Explorer**, right-click **ValueButton.cs**, and then click **View Code** from the shortcut menu.
2. Locate the **class** statement. Immediately after the **{**, type the following code:

```
// Creates the private variable that will store the value of your
// property.
private int varValue;
// Declares the property.
public int ButtonValue
{
    // Sets the method for retrieving the value of your property.
    get
    {
        return varValue;
    }
    // Sets the method for setting the value of your property.
    set
    {
        varValue = value;
    }
}
```

This code sets the methods by which the **ButtonValue** property is stored and retrieved. The **get** statement sets the value returned to the value that is stored in the private variable **varValue**, and the **set** statement sets the value of the private variable by use of the **value** keyword.

3. From the **File** menu, choose **Save All** to save the project.

Testing Your Control

Controls are not stand-alone projects; they must be hosted in a container. In order to test your control, you must provide a test project for it to run in. You must also make your control accessible to the test project by building (compiling) it. In this section, you will build your control and test it in a Windows Form.

To build your control

1. On the **Build** menu, click **Build Solution**.

The build should be successful with no compiler errors or warnings.

To create a test project

1. On the **File** menu, point to **Add** and then click **New Project** to open the **Add New Project** dialog box.
2. Select the **Windows** node, beneath the **Visual C#** node, and click **Windows Forms Application**.
3. In the **Name** box, type **Test**.

4. In **Solution Explorer**, right-click the **References** node for your test project, then select **Add Reference** from the shortcut menu to display the **Add Reference** dialog box.
5. Click the tab labeled **Projects**. Your `ValueButtonLib` project will be listed under **Project Name**. Double-click the project to add the reference to the test project.
6. In **Solution Explorer**, right-click **Test** and select **Build**.

To add your control to the form

1. In **Solution Explorer**, right-click `Form1.cs` and choose **View Designer** from the shortcut menu.

2. In the **Toolbox**, click **ValueButtonLib Components**. Double-click **ValueButton**.

A **ValueButton** appears on the form.

3. Right-click the **ValueButton** and select **Properties** from the shortcut menu.
4. In the **Properties** window, examine the properties of this control. Note that they are identical to the properties exposed by a standard button, except that there is an additional property, `ButtonValue`.
5. Set the `ButtonValue` property to `5`.
6. In the **All Windows Forms** tab of the **Toolbox**, double-click **Label** to add a **Label** control to your form.
7. Relocate the label to the center of the form.
8. Double-click `valueButton1`.

The **Code Editor** opens to the `valueButton1_Click` event.

9. Insert the following line of code.

```
label1.Text = valueButton1.ButtonValue.ToString();
```

10. In **Solution Explorer**, right-click **Test**, and choose **Set as Startup Project** from the shortcut menu.

11. From the **Debug** menu, select **Start Debugging**.

`Form1` appears.

12. Click `valueButton1`.

The numeral '5' is displayed in `label1`, demonstrating that the `ButtonValue` property of your inherited control has been passed to `label1` through the `valueButton1_Click` method. Thus your **ValueButton** control inherits all the functionality of the standard Windows Forms button, but exposes an additional, custom property.

See Also

[Programming with Components](#)

[Component Authoring Walkthroughs](#)

[How to: Display a Control in the Choose Toolbox Items Dialog Box](#)

[Walkthrough: Authoring a Composite Control with Visual C#](#)

Walkthrough: Performing Common Tasks Using Smart Tags on Windows Forms Controls

5/4/2018 • 1 min to read • [Edit Online](#)

As you construct forms and controls for your Windows Forms application, there are many tasks you will perform repeatedly. These are some of the commonly performed tasks you will encounter:

- Adding or removing a tab on a [TabControl](#).
- Docking a control to its parent.
- Changing the orientation of a [SplitContainer](#) control.

To speed development, many controls offer smart tags, which are context-sensitive menus that allow you to perform common tasks like these in a single gesture at design time. These tasks are called *smart-tag verbs*.

Smart tags remain attached to a control instance for its lifetime in the designer and are always available.

Tasks illustrated in this walkthrough include:

- Creating a Windows Forms project
- Using smart tags
- Enabling and Disabling Smart Tags

When you are finished, you will have an understanding of the role played by these important layout features.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

Creating the Project

The first step is to create the project and set up the form.

To create the project

1. Create a Windows-based application project called "SmartTagsExample". For details, see [How to: Create a Windows Application Project](#).
2. Select the form in the **Windows Forms Designer**.

Using Smart Tags

Smart tags are always available at design time on controls that offer them.

To use smart tags

1. Drag a [TabControl](#) from the **Toolbox** onto your form. Note the smart-tag glyph (ⓘ) that appears on the side of the [TabControl](#).
2. Click the smart-tag glyph. In the shortcut menu that appears next to the glyph, select the **Add Tab** item. Observe that a new tab page is added to the [TabControl](#).

3. Drag a [TableLayoutPanel](#) control from the **Toolbox** onto your form.
4. Click the smart-tag glyph. In the shortcut menu that appears next to the glyph, select the **Add Column** item. Observe that a new column is added to the [TableLayoutPanel](#) control.
5. Drag a [SplitContainer](#) control from the **Toolbox** onto your form.
6. Click the smart-tag glyph. In the shortcut menu that appears next to the glyph, select the **Horizontal splitter orientation** item. Observe that the [SplitContainer](#) control's splitter bar is now oriented horizontally.

See Also

- [TextBox](#)
[TabControl](#)
[SplitContainer](#)
[DesignerActionList](#)
[Walkthrough: Adding Smart Tags to a Windows Forms Component](#)

Walkthrough: Serializing Collections of Standard Types with the DesignerSerializationVisibilityAttribute

5/4/2018 • 5 min to read • [Edit Online](#)

Your custom controls will sometimes expose a collection as a property. This walkthrough demonstrates how to use the [DesignerSerializationVisibilityAttribute](#) class to control how a collection is serialized at design time. Applying the [Content](#) value to your collection property ensures that the property will be serialized.

To copy the code in this topic as a single listing, see [How to: Serialize Collections of Standard Types with the DesignerSerializationVisibilityAttribute](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

Prerequisites

In order to complete this walkthrough, you will need:

- Sufficient permissions to be able to create and run Windows Forms application projects on the computer where Visual Studio is installed.

Creating a Control That Has a Serializable Collection

The first step is to create a control that has a serializable collection as a property. You can edit the contents of this collection using the **Collection Editor**, which you can access from the **Properties** window.

To create a control with a serializable collection

1. Create a Windows Control Library project called `SerializationDemoControlLib`. For more information, see [Windows Control Library Template](#).
2. Rename `UserControl1` to `SerializationDemoControl`. For more information, see [How to: Rename Identifiers](#).
3. In the **Properties** window, set the value of the `Padding.All` property to `10`.
4. Place a `TextBox` control in the `SerializationDemoControl`.
5. Select the `TextBox` control. In the **Properties** window, set the following properties.

PROPERTY	CHANGE TO
Multiline	<code>true</code>
Dock	<code>Fill</code>
ScrollBars	<code>Vertical</code>
ReadOnly	<code>true</code>

6. In the **Code Editor**, declare a string array field named `stringsValue` in `SerializationDemoControl`.

```
// This field backs the Strings property.  
private:  
    array<String^>^ stringsValue;
```

```
// This field backs the Strings property.  
private String[] stringsValue = new String[1];
```

```
' This field backs the Strings property.  
Private stringsValue(1) As String
```

7. Define the `Strings` property on the `SerializationDemoControl`.

NOTE

The `Content` value is used to enable serialization of the collection.

```
// When the DesignerSerializationVisibility attribute has  
// a value of "Content" or "Visible" the designer will  
// serialize the property. This property can also be edited  
// at design time with a CollectionEditor.  
public:  
    [DesignerSerializationVisibility(  
        DesignerSerializationVisibility::Content)]  
    property array<String^>^ Strings  
    {  
        array<String^>^ get()  
        {  
            return this->stringsValue;  
        }  
        void set(array<String^>^ value)  
        {  
            this->stringsValue = value;  
  
            // Populate the contained TextBox with the values  
            // in the stringsValue array.  
            StringBuilder^ sb =  
                gcnew StringBuilder(this->stringsValue->Length);  
  
            for (int i = 0; i < this->stringsValue->Length; i++)  
            {  
                sb->Append(this->stringsValue[i]);  
                sb->Append(Environment::NewLine);  
            }  
  
            this->demoControlTextBox->Text = sb->ToString();  
        }  
    }
```

```

// When the DesignerSerializationVisibility attribute has
// a value of "Content" or "Visible" the designer will
// serialize the property. This property can also be edited
// at design time with a CollectionEditor.
[DesignerSerializationVisibility(
    DesignerSerializationVisibility.Content )]
public String[] Strings
{
    get
    {
        return this.stringsValue;
    }
    set
    {
        this.stringsValue = value;

        // Populate the contained TextBox with the values
        // in the stringsValue array.
        StringBuilder sb =
            new StringBuilder(this.stringsValue.Length);

        for (int i = 0; i < this.stringsValue.Length; i++)
        {
            sb.Append(this.stringsValue[i]);
            sb.Append("\r\n");
        }

        this.textBox1.Text = sb.ToString();
    }
}

```

```

' When the DesignerSerializationVisibility attribute has
' a value of "Content" or "Visible" the designer will
' serialize the property. This property can also be edited
' at design time with a CollectionEditor.
<DesignerSerializationVisibility( _
    DesignerSerializationVisibility.Content)> _
Public Property Strings() As String()
    Get
        Return Me.stringsValue
    End Get
    Set(ByVal value As String())
        Me.stringsValue = Value

        ' Populate the contained TextBox with the values
        ' in the stringsValue array.
        Dim sb As New StringBuilder(Me.stringsValue.Length)

        Dim i As Integer
        For i = 0 To (Me.stringsValue.Length) - 1
            sb.Append(Me.stringsValue(i))
            sb.Append(ControlChars.Cr + ControlChars.Lf)
        Next i

        Me.textBox1.Text = sb.ToString()
    End Set
End Property

```

1. Press F5 to build the project and run your control in the **UserControl Test Container**.
2. Find the **Strings** property in the **PropertyGrid** of the **UserControl Test Container**. Click the **Strings** property, then click the ellipsis (...) button to open the **String Collection Editor**.
3. Enter several strings in the **String Collection Editor**. Separate them by pressing the ENTER key at the end

of each string. Click **OK** when you are finished entering strings.

NOTE

The strings you typed appear in the `TextBox` of the `SerializationDemoControl`.

Serializing a Collection Property

To test the serialization behavior of your control, you will place it on a form and change the contents of the collection with the **Collection Editor**. You can see the serialized collection state by looking at a special designer file, into which the **Windows Forms Designer** emits code.

To serialize a collection

1. Add a Windows Application project to the solution. Name the project `SerializationDemoControlTest`.
2. In the **Toolbox**, find the tab named **SerializationDemoControlLib Components**. In this tab, you will find the `SerializationDemoControl`. For more information, see [Walkthrough: Automatically Populating the Toolbox with Custom Components](#).
3. Place a `SerializationDemoControl` on your form.
4. Find the `Strings` property in the **Properties** window. Click the `Strings` property, then click the ellipsis (`[...]`) button to open the **String Collection Editor**.
5. Type several strings in the **String Collection Editor**. Separate them by pressing the ENTER key at the end of each string. Click **OK** when you are finished entering strings.

NOTE

The strings you typed appear in the `TextBox` of the `SerializationDemoControl`.

1. In **Solution Explorer**, click the **Show All Files** button.
2. Open the **Form1** node. Beneath it is a file called **Form1.Designer.cs** or **Form1.Designer.vb**. This is the file into which the **Windows Forms Designer** emits code representing the design-time state of your form and its child controls. Open this file in the **Code Editor**.
3. Open the region called **Windows Form Designer generated code** and find the section labeled **serializationDemoControl1**. Beneath this label is the code representing the serialized state of your control. The strings you typed in step 5 appear in an assignment to the `Strings` property. The following code examples in C# and Visual Basic, show code similar to what you will see if you typed the strings "red", "orange", and "yellow".

```
this.serializationDemoControl1.Strings = new string[] {  
    "red",  
    "orange",  
    "yellow"};
```

```
Me.serializationDemoControl1.Strings = New String() {"red", "orange", "yellow"}
```

4. In the **Code Editor**, change the value of the **DesignerSerializationVisibilityAttribute** on the `Strings` property to **Hidden**.

```
[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
```

```
<DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)> _
```

5. Rebuild the solution and repeat steps 3 and 4.

NOTE

In this case, the **Windows Forms Designer** emits no assignment to the `Strings` property.

Next Steps

Once you know how to serialize a collection of standard types, consider integrating your custom controls more deeply into the design-time environment. The following topics describe how to enhance the design-time integration of your custom controls:

- [Design-Time Architecture](#)
- [Attributes in Windows Forms Controls](#)
- [Designer Serialization Overview](#)
- [Walkthrough: Creating a Windows Forms Control That Takes Advantage of Visual Studio Design-Time Features](#)

See Also

[DesignerSerializationVisibilityAttribute](#)

[Designer Serialization Overview](#)

[How to: Serialize Collections of Standard Types with the DesignerSerializationVisibilityAttribute](#)

[Walkthrough: Automatically Populating the Toolbox with Custom Components](#)

Walkthrough: Debugging Custom Windows Forms Controls at Design Time

5/4/2018 • 5 min to read • [Edit Online](#)

When you create a custom control, you will often find it necessary to debug its design-time behavior. This is especially true if you are authoring a custom designer for your custom control. For details, see [Walkthrough: Creating a Windows Forms Control That Takes Advantage of Visual Studio Design-Time Features](#).

You can debug your custom controls using Visual Studio, just as you would debug any other .NET Framework classes. The difference is that you will debug a separate instance of Visual Studio that is running your custom control's code.

Tasks illustrated in this walkthrough include:

- Creating a Windows Forms project to host your custom control
- Creating a control library project
- Adding a property to your custom control
- Adding your custom control to the host form
- Setting up the project for design-time debugging
- Debugging your custom control at design time

When you are finished, you will have an understanding of the tasks necessary for debugging the design-time behavior of a custom control.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

Creating the Project

The first step is to create the application project. You will use this project to build the application that hosts the custom control.

To create the project

- Create a Windows Application project called "DebuggingExample". For details, see [How to: Create a Windows Application Project](#).

Creating a Control Library Project

The next step is to create the control library project and set up the custom control.

To create the control library project

1. Add a **Windows Control Library** project to the solution.
2. Add a new **UserControl** item to the DebugControlLibrary project. For details, see [NIB:How to: Add New Project Items](#). Give the new source file a base name of "DebugControl".

3. Using the **Solution Explorer**, delete the project's default control by deleting the code file with a base name of "`UserControl1`". For details, see [NIB:How to: Remove, Delete, and Exclude Items](#).
4. Build the solution.

Checkpoint

At this point, you will be able to see your custom control in the **Toolbox**.

To check your progress

- Find the new tab called **DebugControlLibrary Components** and click to select it. When it opens, you will see your control listed as **DebugControl** with the default icon beside it.

Adding a Property to Your Custom Control

To demonstrate that your custom control's code is running at design-time, you will add a property and set a breakpoint in the code that implements the property.

To add a property to your custom control

1. Open **DebugControl** in the **Code Editor**. Add the following code to the class definition:

```
Private demoStringValue As String = Nothing
<BrowsableAttribute(true)>
Public Property DemoString() As String

    Get
        Return Me.demoStringValue
    End Get

    Set(ByVal value As String)
        Me.demoStringValue = value
    End Set

End Property
```

```
private string demoStringValue = null;
[Browsable(true)]
public string DemoString
{
    get
    {
        return this.demoStringValue;
    }
    set
    {
        demoStringValue = value;
    }
}
```

2. Build the solution.

Adding Your Custom Control to the Host Form

To debug the design-time behavior of your custom control, you will place an instance of the custom control class on a host form.

To add your custom control to the host form

1. In the "DebuggingExample" project, open Form1 in the **Windows Forms Designer**.
2. In the **Toolbox**, open the **DebugControlLibrary Components** tab and drag a **DebugControl** instance

onto the form.

3. Find the `DemoString` custom property in the **Properties** window. Note that you can change its value as you would any other property. Also note that when the `DemoString` property is selected, the property's description string appears at the bottom of the **Properties** window.

Setting Up the Project for Design-Time Debugging

To debug your custom control's design-time behavior, you will debug a separate instance of Visual Studio that is running your custom control's code.

To set up the project for design-time debugging

1. Right-click on the **DebugControlLibrary** project in the **Solution Explorer** and select **Properties**.

2. In the **DebugControlLibrary** property sheet, select the **Debug** tab.

In the **Start Action** section, select **Start external program**. You will be debugging a separate instance of Visual Studio, so click the ellipsis (...) button to browse for the Visual Studio IDE. The name of the executable file is **devenv.exe**, and if you installed to the default location, its path is %programfiles%\Microsoft Visual Studio 9.0\Common7\IDE\devenv.exe.

3. Click **OK** to close the dialog box.

4. Right-click the **DebugControlLibrary** project and select **Set as StartUp Project** to enable this debugging configuration.

Debugging Your Custom Control at Design Time

Now you are ready to debug your custom control as it runs in design mode. When you start the debugging session, a new instance of Visual Studio will be created, and you will use it to load the "DebuggingExample" solution. When you open Form1 in the **Forms Designer**, an instance of your custom control will be created and will start running.

To debug your custom control at design time

1. Open the **DebugControl** source file in the **Code Editor** and place a breakpoint on the `set` accessor of the `DemoString` property.

2. Press F5 to start the debugging session. Note that a new instance of Visual Studio is created. You can distinguish between the instances in two ways:

- The debugging instance has the word **Running** in its title bar
- The debugging instance has the **Start** button on its **Debug** toolbar disabled

Your breakpoint is set in the debugging instance.

3. In the new instance of Visual Studio, open the "DebuggingExample" solution. You can easily find the solution by selecting **Recent Projects** from the **File** menu. The "DebuggingExample.sln" solution file will be listed as the most recently used file.

4. Open Form1 in the **Forms Designer** and select the **DebugControl** control.

5. Change the value of the `DemoString` property. Note that when you commit the change, the debugging instance of Visual Studio acquires focus and execution stops at your breakpoint. You can single-step through the property accessor just as you would any other code.

6. When you are finished with your debugging session, you can exit by dismissing the hosted instance of Visual Studio or by clicking the **Stop Debugging** button in the debugging instance.

Next Steps

Now that you can debug your custom controls at design time, there are many possibilities for expanding your control's interaction with the Visual Studio IDE.

- You can use the [DesignMode](#) property of the [Component](#) class to write code that will only execute at design time. For details, see [DesignMode](#).
- There are several attributes you can apply to your control's properties to manipulate your custom control's interaction with the designer. You can find these attributes in the [System.ComponentModel](#) namespace.
- You can write a custom designer for your custom control. This gives you complete control over the design experience using the extensible designer infrastructure exposed by Visual Studio. For details, see [Walkthrough: Creating a Windows Forms Control That Takes Advantage of Visual Studio Design-Time Features](#).

See Also

[Walkthrough: Creating a Windows Forms Control That Takes Advantage of Visual Studio Design-Time Features](#)

[How to: Access Design-Time Services](#)

[How to: Access Design-Time Support in Windows Forms](#)

Walkthrough: Creating a Windows Forms Control That Takes Advantage of Visual Studio Design-Time Features

5/4/2018 • 52 min to read • [Edit Online](#)

The design-time experience for a custom control can be enhanced by authoring an associated custom designer.

This walkthrough illustrates how to create a custom designer for a custom control. You will implement a `MarqueeControl` type and an associated designer class, called `MarqueeControlRootDesigner`.

The `MarqueeControl` type implements a display similar to a theater marquee, with animated lights and flashing text.

The designer for this control interacts with the design environment to provide a custom design-time experience. With the custom designer, you can assemble a custom `MarqueeControl` implementation with animated lights and flashing text in many combinations. You can use the assembled control on a form like any other Windows Forms control.

Tasks illustrated in this walkthrough include:

- Creating the Project
- Creating a Control Library Project
- Referencing the Custom Control Project
- Defining a Custom Control and Its Custom Designer
- Creating an Instance of Your Custom Control
- Setting Up the Project for Design-Time Debugging
- Implementing Your Custom Control
- Creating a Child Control for Your Custom Control
- Create the MarqueeBorder Child Control
- Creating a Custom Designer to Shadow and Filter Properties
- Handling Component Changes
- Adding Designer Verbs to your Custom Designer
- Creating a Custom `UITypeEditor`
- Testing your Custom Control in the Designer

When you are finished, your custom control will look something like the following:



For the complete code listing, see [How to: Create a Windows Forms Control That Takes Advantage of Design-Time Features](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

Prerequisites

In order to complete this walkthrough, you will need:

- Sufficient permissions to be able to create and run Windows Forms application projects on the computer where Visual Studio is installed.

Creating the Project

The first step is to create the application project. You will use this project to build the application that hosts the custom control.

To create the project

- Create a Windows Forms Application project called "MarqueeControlTest." For more information, see [How to: Create a Windows Application Project](#).

Creating a Control Library Project

The next step is to create the control library project. You will create a new custom control and its corresponding custom designer.

To create the control library project

1. Add a Windows Forms Control Library project to the solution. Name the project "MarqueeControlLibrary."
2. Using **Solution Explorer**, delete the project's default control by deleting the source file named "UserControl1.cs" or "UserControl1.vb", depending on your language of choice. For more information, see [NIB:How to: Remove, Delete, and Exclude Items](#).
3. Add a new **UserControl** item to the `MarqueeControlLibrary` project. Give the new source file a base name of "MarqueeControl."
4. Using **Solution Explorer**, create a new folder in the `MarqueeControlLibrary` project. For more information, see [NIB:How to: Add New Project Items](#). Name the new folder "Design."
5. Right-click the **Design** folder and add a new class. Give the source file a base name of "MarqueeControlRootDesigner."
6. You will need to use types from the System.Design assembly, so add this reference to the `MarqueeControlLibrary` project.

NOTE

To use the System.Design assembly, your project must target the full version of the .NET Framework, not the .NET Framework Client Profile. To change the target framework, see [How to: Target a Version of the .NET Framework](#).

Referencing the Custom Control Project

You will use the `MarqueeControlTest` project to test the custom control. The test project will become aware of the custom control when you add a project reference to the `MarqueeControlLibrary` assembly.

To reference the custom control project

- In the `MarqueeControlTest` project, add a project reference to the `MarqueeControlLibrary` assembly. Be sure to use the **Projects** tab in the **Add Reference** dialog box instead of referencing the `MarqueeControlLibrary` assembly directly.

Defining a Custom Control and Its Custom Designer

Your custom control will derive from the `UserControl` class. This allows your control to contain other controls, and it gives your control a great deal of default functionality.

Your custom control will have an associated custom designer. This allows you to create a unique design experience tailored specifically for your custom control.

You associate the control with its designer by using the `DesignerAttribute` class. Because you are developing the entire design-time behavior of your custom control, the custom designer will implement the `IRootDesigner` interface.

To define a custom control and its custom designer

- Open the `MarqueeControl` source file in the **Code Editor**. At the top of the file, import the following namespaces:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.ComponentModel.Design;
using System.Drawing;
using System.Windows.Forms;
using System.Windows.Forms.Design;
```

```
Imports System
Imports System.Collections
Imports System.ComponentModel
Imports System.ComponentModel.Design
Imports System.Drawing
Imports System.Windows.Forms
Imports System.Windows.Forms.Design
```

- Add the `DesignerAttribute` to the `MarqueeControl` class declaration. This associates the custom control with its designer.

```
[Designer( typeof( MarqueeControlLibrary.Design.MarqueeControlRootDesigner ), typeof( IRootDesigner ) )
]
public class MarqueeControl : UserControl
{
```

```
<Designer(GetType(MarqueeControlLibrary.Design.MarqueeControlRootDesigner), _
GetType(IRootDesigner))> _
Public Class MarqueeControl
    Inherits UserControl
```

- Open the `MarqueeControlRootDesigner` source file in the **Code Editor**. At the top of the file, import the

following namespaces:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.ComponentModel.Design;
using System.Diagnostics;
using System.Drawing.Design;
using System.Windows.Forms;
using System.Windows.Forms.Design;
```

```
Imports System
Imports System.Collections
Imports System.ComponentModel
Imports System.ComponentModel.Design
Imports System.Diagnostics
Imports System.Drawing.Design
Imports System.Windows.Forms
Imports System.Windows.Forms.Design
```

4. Change the declaration of `MarqueeControlRootDesigner` to inherit from the `DocumentDesigner` class. Apply the `ToolboxItemFilterAttribute` to specify the designer interaction with the **Toolbox**.

Note The definition for the `MarqueeControlRootDesigner` class has been enclosed in a namespace called "MarqueeControlLibrary.Design." This declaration places the designer in a special namespace reserved for design-related types.

```
namespace MarqueeControlLibrary.Design
{
    [ToolboxItemFilter("MarqueeControlLibrary.MarqueeBorder", ToolboxItemType.Require)]
    [ToolboxItemFilter("MarqueeControlLibrary.MarqueeText", ToolboxItemType.Require)]
    [System.Security.Permissions.PermissionSet(System.Security.Permissions.SecurityAction.Demand, Name =
    = "FullTrust")]
    public class MarqueeControlRootDesigner : DocumentDesigner
    {
```

```
Namespace MarqueeControlLibrary.Design

<ToolboxItemFilter("MarqueeControlLibrary.MarqueeBorder", _
    ToolboxItemType.Require), _
    ToolboxItemFilter("MarqueeControlLibrary.MarqueeText", _
    ToolboxItemType.Require)> _

<System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.Demand,
    Name:="FullTrust")> _
    Public Class MarqueeControlRootDesigner
        Inherits DocumentDesigner
```

5. Define the constructor for the `MarqueeControlRootDesigner` class. Insert a `.WriteLine` statement in the constructor body. This will be useful for debugging purposes.

```
public MarqueeControlRootDesigner()
{
    Trace.WriteLine("MarqueeControlRootDesigner ctor");
}
```

```
Public Sub New()
    Trace.WriteLine("MarqueeControlRootDesigner ctor")
End Sub
```

Creating an Instance of Your Custom Control

To observe the custom design-time behavior of your control, you will place an instance of your control on the form in `MarqueeControlTest` project.

To create an instance of your custom control

1. Add a new `UserControl` item to the `MarqueeControlTest` project. Give the new source file a base name of "DemoMarqueeControl."
2. Open the `DemoMarqueeControl` file in the **Code Editor**. At the top of the file, import the `MarqueeControlLibrary` namespace:

```
Imports MarqueeControlLibrary
```

```
using MarqueeControlLibrary;
```

1. Change the declaration of `DemoMarqueeControl` to inherit from the `MarqueeControl` class.
2. Build the project.
3. Open `Form1` in the Windows Forms Designer.
4. Find the **MarqueeControlTest Components** tab in the **Toolbox** and open it. Drag a `DemoMarqueeControl` from the **Toolbox** onto your form.
5. Build the project.

Setting Up the Project for Design-Time Debugging

When you are developing a custom design-time experience, it will be necessary to debug your controls and components. There is a simple way to set up your project to allow debugging at design time. For more information, see [Walkthrough: Debugging Custom Windows Forms Controls at Design Time](#).

To set up the project for design-time debugging

1. Right-click the `MarqueeControlLibrary` project and select **Properties**.
2. In the "MarqueeControlLibrary Property Pages" dialog box, select the **Debug** page.
3. In the **Start Action** section, select **Start External Program**. You will be debugging a separate instance of Visual Studio, so click the ellipsis (...) button to browse for the Visual Studio IDE. The name of the executable file is devenv.exe, and if you installed to the default location, its path is %programfiles%\Microsoft Visual Studio 9.0\Common7\IDE\devenv.exe.
4. Click OK to close the dialog box.
5. Right-click the `MarqueeControlLibrary` project and select "Set as StartUp Project" to enable this debugging configuration.

Checkpoint

You are now ready to debug the design-time behavior of your custom control. Once you have determined that the

debugging environment is set up correctly, you will test the association between the custom control and the custom designer.

To test the debugging environment and the designer association

1. Open the `MarqueeControlRootDesigner` source file in the **Code Editor** and place a breakpoint on the `WriteLine` statement.
2. Press F5 to start the debugging session. Note that a new instance of Visual Studio is created.
3. In the new instance of Visual Studio, open the "MarqueeControlTest" solution. You can easily find the solution by selecting **Recent Projects** from the **File** menu. The "MarqueeControlTest.sln" solution file will be listed as the most recently used file.
4. Open the `DemoMarqueeControl1` in the designer. Note that the debugging instance of Visual Studio acquires focus and execution stops at your breakpoint. Press F5 to continue the debugging session.

At this point, everything is in place for you to develop and debug your custom control and its associated custom designer. The remainder of this walkthrough will concentrate on the details of implementing features of the control and the designer.

Implementing Your Custom Control

The `MarqueeControl` is a `UserControl` with a little bit of customization. It exposes two methods: `start`, which starts the marquee animation, and `stop`, which stops the animation. Because the `MarqueeControl` contains child controls that implement the `IMarqueeWidget` interface, `start` and `stop` enumerate each child control and call the `StartMarquee` and `StopMarquee` methods, respectively, on each child control that implements `IMarqueeWidget`.

The appearance of the `MarqueeBorder` and `MarqueeText` controls is dependent on the layout, so `MarqueeControl` overrides the `OnLayout` method and calls `PerformLayout` on child controls of this type.

This is the extent of the `MarqueeControl` customizations. The run-time features are implemented by the `MarqueeBorder` and `MarqueeText` controls, and the design-time features are implemented by the `MarqueeBorderDesigner` and `MarqueeControlRootDesigner` classes.

To implement your custom control

1. Open the `MarqueeControl` source file in the **Code Editor**. Implement the `Start` and `Stop` methods.

```

public void Start()
{
    // The MarqueeControl may contain any number of
    // controls that implement IMarqueeWidget, so
    // find each IMarqueeWidget child and call its
    // StartMarquee method.
    foreach( Control cntrl in this.Controls )
    {
        if( cntrl is IMarqueeWidget )
        {
            IMarqueeWidget widget = cntrl as IMarqueeWidget;
            widget.StartMarquee();
        }
    }
}

public void Stop()
{
    // The MarqueeControl may contain any number of
    // controls that implement IMarqueeWidget, so find
    // each IMarqueeWidget child and call its StopMarquee
    // method.
    foreach( Control cntrl in this.Controls )
    {
        if( cntrl is IMarqueeWidget )
        {
            IMarqueeWidget widget = cntrl as IMarqueeWidget;
            widget.StopMarquee();
        }
    }
}

```

```

Public Sub Start()
    ' The MarqueeControl may contain any number of
    ' controls that implement IMarqueeWidget, so
    ' find each IMarqueeWidget child and call its
    ' StartMarquee method.
    Dim cntrl As Control
    For Each cntrl In Me.Controls
        If TypeOf cntrl Is IMarqueeWidget Then
            Dim widget As IMarqueeWidget = CType(cntrl, IMarqueeWidget)

                widget.StartMarquee()
        End If
    Next cntrl
End Sub

Public Sub [Stop]()
    ' The MarqueeControl may contain any number of
    ' controls that implement IMarqueeWidget, so find
    ' each IMarqueeWidget child and call its StopMarquee
    ' method.
    Dim cntrl As Control
    For Each cntrl In Me.Controls
        If TypeOf cntrl Is IMarqueeWidget Then
            Dim widget As IMarqueeWidget = CType(cntrl, IMarqueeWidget)

                widget.StopMarquee()
        End If
    Next cntrl
End Sub

```

2. Override the [OnLayout](#) method.

```

protected override void OnLayout(LayoutEventArgs levent)
{
    base.OnLayout (levent);

    // Repaint all IMarqueeWidget children if the layout
    // has changed.
    foreach( Control cntrl in this.Controls )
    {
        if( cntrl is IMarqueeWidget )
        {
            Control control = cntrl as Control;

            control.PerformLayout();
        }
    }
}

```

```

Protected Overrides Sub OnLayout(ByVal levent As LayoutEventArgs)
    MyBase.OnLayout(levent)

    ' Repaint all IMarqueeWidget children if the layout
    ' has changed.
    Dim cntrl As Control
    For Each cntrl In Me.Controls
        If TypeOf cntrl Is IMarqueeWidget Then
            Dim widget As IMarqueeWidget = CType(cntrl, IMarqueeWidget)

            cntrl.PerformLayout()
        End If
    Next cntrl
End Sub

```

Creating a Child Control for Your Custom Control

The `MarqueeControl` will host two kinds of child control: the `MarqueeBorder` control and the `MarqueeText` control.

- `MarqueeBorder` : This control paints a border of "lights" around its edges. The lights flash in sequence, so they appear to be moving around the border. The speed at which the lights flash is controlled by a property called `UpdatePeriod`. Several other custom properties determine other aspects of the control's appearance. Two methods, called `StartMarquee` and `StopMarquee`, control when the animation starts and stops.
- `MarqueeText` : This control paints a flashing string. Like the `MarqueeBorder` control, the speed at which the text flashes is controlled by the `updatePeriod` property. The `MarqueeText` control also has the `StartMarquee` and `StopMarquee` methods in common with the `MarqueeBorder` control.

At design time, the `MarqueeControlRootDesigner` allows these two control types to be added to a `MarqueeControl` in any combination.

Common features of the two controls are factored into an interface called `IMarqueeWidget`. This allows the `MarqueeControl` to discover any Marquee-related child controls and give them special treatment.

To implement the periodic animation feature, you will use `BackgroundWorker` objects from the `System.ComponentModel` namespace. You could use `Timer` objects, but when many `IMarqueeWidget` objects are present, the single UI thread may be unable to keep up with the animation.

To create a child control for your custom control

1. Add a new class item to the `MarqueeControlLibrary` project. Give the new source file a base name of "`IMarqueeWidget`".

2. Open the `IMarqueeWidget` source file in the **Code Editor** and change the declaration from `class` to

`interface :`

```
// This interface defines the contract for any class that is to
// be used in constructing a MarqueeControl.
public interface IMarqueeWidget
{
```

```
' This interface defines the contract for any class that is to
' be used in constructing a MarqueeControl.
Public Interface IMarqueeWidget
```

3. Add the following code to the `IMarqueeWidget` interface to expose two methods and a property that manipulate the marquee animation:

```
// This interface defines the contract for any class that is to
// be used in constructing a MarqueeControl.
public interface IMarqueeWidget
{
    // This method starts the animation. If the control can
    // contain other classes that implement IMarqueeWidget as
    // children, the control should call StartMarquee on all
    // its IMarqueeWidget child controls.
    void StartMarquee();

    // This method stops the animation. If the control can
    // contain other classes that implement IMarqueeWidget as
    // children, the control should call StopMarquee on all
    // its IMarqueeWidget child controls.
    void StopMarquee();

    // This method specifies the refresh rate for the animation,
    // in milliseconds.
    int UpdatePeriod
    {
        get;
        set;
    }
}
```

```

' This interface defines the contract for any class that is to
' be used in constructing a MarqueeControl.
Public Interface IMarqueeWidget

    ' This method starts the animation. If the control can
    ' contain other classes that implement IMarqueeWidget as
    ' children, the control should call StartMarquee on all
    ' its IMarqueeWidget child controls.
    Sub StartMarquee()

        ' This method stops the animation. If the control can
        ' contain other classes that implement IMarqueeWidget as
        ' children, the control should call StopMarquee on all
        ' its IMarqueeWidget child controls.
        Sub StopMarquee()

            ' This method specifies the refresh rate for the animation,
            ' in milliseconds.
            Property UpdatePeriod() As Integer

        End Interface

```

4. Add a new **Custom Control** item to the `MarqueeControlLibrary` project. Give the new source file a base name of "MarqueeText."
5. Drag a **BackgroundWorker** component from the **Toolbox** onto your `MarqueeText` control. This component will allow the `MarqueeText` control to update itself asynchronously.
6. In the Properties window, set the **BackgroundWorker** component's `WorkerReportsProgress` and `WorkerSupportsCancellation` properties to `true`. These settings allow the **BackgroundWorker** component to periodically raise the `ProgressChanged` event and to cancel asynchronous updates. For more information, see [BackgroundWorker Component](#).
7. Open the `MarqueeText` source file in the **Code Editor**. At the top of the file, import the following namespaces:

```

using System;
using System.ComponentModel;
using System.ComponentModel.Design;
using System.Diagnostics;
using System.Drawing;
using System.Threading;
using System.Windows.Forms;
using System.Windows.Forms.Design;

```

```

Imports System
Imports System.ComponentModel
Imports System.ComponentModel.Design
Imports System.Diagnostics
Imports System.Drawing
Imports System.Threading
Imports System.Windows.Forms
Imports System.Windows.Forms.Design

```

8. Change the declaration of `MarqueeText` to inherit from `Label` and to implement the `IMarqueeWidget` interface:

```
[ToolboxItemFilter("MarqueeControlLibrary.MarqueeText", ToolboxItemType.Require)]
public partial class MarqueeText : Label, IMarqueeWidget
{
```

```
<ToolboxItemFilter("MarqueeControlLibrary.MarqueeText", _
ToolboxItemType.Require)> _
Partial Public Class MarqueeText
    Inherits Label
    Implements IMarqueeWidget
```

9. Declare the instance variables that correspond to the exposed properties, and initialize them in the constructor. The `isLit` field determines if the text is to be painted in the color given by the `LightColor` property.

```
// When isLit is true, the text is painted in the light color;
// When isLit is false, the text is painted in the dark color.
// This value changes whenever the BackgroundWorker component
// raises the ProgressChanged event.
private bool isLit = true;

// These fields back the public properties.
private int updatePeriodValue = 50;
private Color lightColorValue;
private Color darkColorValue;

// These brushes are used to paint the light and dark
// colors of the text.
private Brush lightBrush;
private Brush darkBrush;

// This component updates the control asynchronously.
private BackgroundWorker backgroundWorker1;

public MarqueeText()
{
    // This call is required by the Windows.Forms Form Designer.
    InitializeComponent();

    // Initialize light and dark colors
    // to the control's default values.
    this.lightColorValue = this.ForeColor;
    this.darkColorValue = this.BackColor;
    this.lightBrush = new SolidBrush(this.lightColorValue);
    this.darkBrush = new SolidBrush(this.darkColorValue);
}
```

```

' When isLit is true, the text is painted in the light color;
' When isLit is false, the text is painted in the dark color.
' This value changes whenever the BackgroundWorker component
' raises the ProgressChanged event.
Private isLit As Boolean = True

' These fields back the public properties.
Private updatePeriodValue As Integer = 50
Private lightColorValue As Color
Private darkColorValue As Color

' These brushes are used to paint the light and dark
' colors of the text.
Private lightBrush As Brush
Private darkBrush As Brush

' This component updates the control asynchronously.
Private WithEvents backgroundWorker1 As BackgroundWorker

Public Sub New()
    ' This call is required by the Windows.Forms Form Designer.
    InitializeComponent()

    ' Initialize light and dark colors
    ' to the control's default values.
    Me.lightColorValue = Me.ForeColor
    Me.darkColorValue = Me.BackColor
    Me.lightBrush = New SolidBrush(Me.lightColorValue)
    Me.darkBrush = New SolidBrush(Me.darkColorValue)
End Sub 'New

```

10. Implement the `IMarqueeWidget` interface.

The `StartMarquee` and `StopMarquee` methods invoke the `BackgroundWorker` component's `RunWorkerAsync` and `CancelAsync` methods to start and stop the animation.

The `Category` and `Browsable` attributes are applied to the `UpdatePeriod` property so it appears in a custom section of the Properties window called "Marquee."

```

public virtual void StartMarquee()
{
    // Start the updating thread and pass it the UpdatePeriod.
    this.backgroundWorker1.RunWorkerAsync(this.UpdatePeriod);
}

public virtual void StopMarquee()
{
    // Stop the updating thread.
    this.backgroundWorker1.CancelAsync();
}

[Category("Marquee")]
[Browsable(true)]
public int UpdatePeriod
{
    get
    {
        return this.updatePeriodValue;
    }

    set
    {
        if (value > 0)
        {
            this.updatePeriodValue = value;
        }
        else
        {
            throw new ArgumentOutOfRangeException("UpdatePeriod", "must be > 0");
        }
    }
}

```

```

Public Overridable Sub StartMarquee() _
Implements IMarqueeWidget.StartMarquee
    ' Start the updating thread and pass it the UpdatePeriod.
    Me.backgroundWorker1.RunWorkerAsync(Me.UpdatePeriod)
End Sub

Public Overridable Sub StopMarquee() _
Implements IMarqueeWidget.StopMarquee
    ' Stop the updating thread.
    Me.backgroundWorker1.CancelAsync()
End Sub

<Category("Marquee"), Browsable(True)> _
Public Property UpdatePeriod() As Integer _
Implements IMarqueeWidget.UpdatePeriod

    Get
        Return Me.updatePeriodValue
    End Get

    Set(ByVal Value As Integer)
        If Value > 0 Then
            Me.updatePeriodValue = Value
        Else
            Throw New ArgumentOutOfRangeException("UpdatePeriod", "must be > 0")
        End If
    End Set

End Property

```

11. Implement the property accessors. You will expose two properties to clients: `LightColor` and `DarkColor`.

The `Category` and `Browsable` attributes are applied to these properties, so the properties appear in a custom section of the Properties window called "Marquee."

```
[Category("Marquee")]
[Browsable(true)]
public Color LightColor
{
    get
    {
        return this.lightColorValue;
    }
    set
    {
        // The LightColor property is only changed if the
        // client provides a different value. Comparing values
        // from the ToArgb method is the recommended test for
        // equality between Color structs.
        if (this.lightColorValue.ToArgb() != value.ToArgb())
        {
            this.lightColorValue = value;
            this.lightBrush = new SolidBrush(value);
        }
    }
}

[Category("Marquee")]
[Browsable(true)]
public Color DarkColor
{
    get
    {
        return this.darkColorValue;
    }
    set
    {
        // The DarkColor property is only changed if the
        // client provides a different value. Comparing values
        // from the ToArgb method is the recommended test for
        // equality between Color structs.
        if (this.darkColorValue.ToArgb() != value.ToArgb())
        {
            this.darkColorValue = value;
            this.darkBrush = new SolidBrush(value);
        }
    }
}
```

```

<Category("Marquee"), Browsable(True)> _
Public Property LightColor() As Color

    Get
        Return Me.lightColorValue
    End Get

    Set(ByVal Value As Color)
        ' The LightColor property is only changed if the
        ' client provides a different value. Comparing values
        ' from the ToArgb method is the recommended test for
        ' equality between Color structs.
        If Me.lightColorValue.ToArgb() <> Value.ToArgb() Then
            Me.lightColorValue = Value
            Me.lightBrush = New SolidBrush(Value)
        End If
    End Set

End Property

<Category("Marquee"), Browsable(True)> _
Public Property DarkColor() As Color

    Get
        Return Me.darkColorValue
    End Get

    Set(ByVal Value As Color)
        ' The DarkColor property is only changed if the
        ' client provides a different value. Comparing values
        ' from the ToArgb method is the recommended test for
        ' equality between Color structs.
        If Me.darkColorValue.ToArgb() <> Value.ToArgb() Then
            Me.darkColorValue = Value
            Me.darkBrush = New SolidBrush(Value)
        End If
    End Set

End Property

```

12. Implement the handlers for the [BackgroundWorker](#) component's [DoWork](#) and [ProgressChanged](#) events.

The [DoWork](#) event handler sleeps for the number of milliseconds specified by [UpdatePeriod](#) then raises the [ProgressChanged](#) event, until your code stops the animation by calling [CancelAsync](#).

The [ProgressChanged](#) event handler toggles the text between its light and dark state to give the appearance of flashing.

```
// This method is called in the worker thread's context,
// so it must not make any calls into the MarqueeText control.
// Instead, it communicates to the control using the
// ProgressChanged event.
//
// The only work done in this event handler is
// to sleep for the number of milliseconds specified
// by UpdatePeriod, then raise the ProgressChanged event.
private void backgroundWorker1_DoWork(
    object sender,
    System.ComponentModel.DoWorkEventArgs e)
{
    BackgroundWorker worker = sender as BackgroundWorker;

    // This event handler will run until the client cancels
    // the background task by calling CancelAsync.
    while (!worker.CancellationPending)
    {
        // The Argument property of the DoWorkEventArgs
        // object holds the value of UpdatePeriod, which
        // was passed as the argument to the RunWorkerAsync
        // method.
        Thread.Sleep((int)e.Argument);

        // The DoWork eventhandler does not actually report
        // progress; the ReportProgress event is used to
        // periodically alert the control to update its state.
        worker.ReportProgress(0);
    }
}

// The ProgressChanged event is raised by the DoWork method.
// This event handler does work that is internal to the
// control. In this case, the text is toggled between its
// light and dark state, and the control is told to
// repaint itself.
private void backgroundWorker1_ProgressChanged(object sender,
    System.ComponentModel.ProgressChangedEventArgs e)
{
    this.isLit = !this.isLit;
    this.Refresh();
}
```

```

' This method is called in the worker thread's context,
' so it must not make any calls into the MarqueeText control.
' Instead, it communicates to the control using the
' ProgressChanged event.
'

' The only work done in this event handler is
' to sleep for the number of milliseconds specified
' by UpdatePeriod, then raise the ProgressChanged event.
Private Sub backgroundWorker1_DoWork( _
    ByVal sender As Object, _
    ByVal e As System.ComponentModel.DoWorkEventArgs) _
Handles backgroundWorker1.DoWork
    Dim worker As BackgroundWorker = CType(sender, BackgroundWorker)

    ' This event handler will run until the client cancels
    ' the background task by calling CancelAsync.
    While Not worker.CancellationPending
        ' The Argument property of the DoWorkEventArgs
        ' object holds the value of UpdatePeriod, which
        ' was passed as the argument to the RunWorkerAsync
        ' method.
        Thread.Sleep(Fix(e.Argument))

        ' The DoWork eventhandler does not actually report
        ' progress; the ReportProgress event is used to
        ' periodically alert the control to update its state.
        worker.ReportProgress(0)
    End While
End Sub

' The ProgressChanged event is raised by the DoWork method.
' This event handler does work that is internal to the
' control. In this case, the text is toggled between its
' light and dark state, and the control is told to
' repaint itself.
Private Sub backgroundWorker1_ProgressChanged( _
    ByVal sender As Object, _
    ByVal e As System.ComponentModel.ProgressChangedEventArgs) _
Handles backgroundWorker1.ProgressChanged
    Me.isLit = Not Me.isLit
    Me.Refresh()
End Sub

```

13. Override the [OnPaint](#) method to enable the animation.

```

protected override void OnPaint(PaintEventArgs e)
{
    // The text is painted in the light or dark color,
    // depending on the current value of isLit.
    this.ForeColor =
        this.isLit ? this.lightColorValue : this.darkColorValue;

    base.OnPaint(e);
}

```

```

Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
    ' The text is painted in the light or dark color,
    ' depending on the current value of isLit.
    Me.ForeColor = IIf(Me.isLit, Me.lightColorValue, Me.darkColorValue)

    MyBase.OnPaint(e)
End Sub

```

14. Press F6 to build the solution.

Create the MarqueeBorder Child Control

The `MarqueeBorder` control is slightly more sophisticated than the `MarqueeText` control. It has more properties and the animation in the `OnPaint` method is more involved. In principle, it is quite similar to the `MarqueeText` control.

Because the `MarqueeBorder` control can have child controls, it needs to be aware of [Layout](#) events.

To create the MarqueeBorder control

1. Add a new **Custom Control** item to the `MarqueeControlLibrary` project. Give the new source file a base name of "MarqueeBorder."
2. Drag a [BackgroundWorker](#) component from the **Toolbox** onto your `MarqueeBorder` control. This component will allow the `MarqueeBorder` control to update itself asynchronously.
3. In the Properties window, set the [BackgroundWorker](#) component's `WorkerReportsProgress` and `WorkerSupportsCancellation` properties to `true`. These settings allow the [BackgroundWorker](#) component to periodically raise the `ProgressChanged` event and to cancel asynchronous updates. For more information, see [BackgroundWorker Component](#).
4. In the Properties window, click the Events button. Attach handlers for the `DoWork` and `ProgressChanged` events.
5. Open the `MarqueeBorder` source file in the **Code Editor**. At the top of the file, import the following namespaces:

```
using System;
using System.ComponentModel;
using System.ComponentModel.Design;
using System.Diagnostics;
using System.Drawing;
using System.Drawing.Design;
using System.Threading;
using System.Windows.Forms;
using System.Windows.Forms.Design;
```

```
Imports System
Imports System.ComponentModel
Imports System.ComponentModel.Design
Imports System.Diagnostics
Imports System.Drawing
Imports System.Drawing.Design
Imports System.Threading
Imports System.Windows.Forms
Imports System.Windows.Forms.Design
```

6. Change the declaration of `MarqueeBorder` to inherit from `Panel` and to implement the `IMarqueeWidget` interface.

```
[Designer(typeof(MarqueeControlLibrary.Design.MarqueeBorderDesigner))]
[ToolboxItemFilter("MarqueeControlLibrary.MarqueeBorder", ToolboxItemFilterType.Require)]
public partial class MarqueeBorder : Panel, IMarqueeWidget
{
```

```

<Designer(GetType(MarqueeControlLibrary.Design.MarqueeBorderDesigner)), _
ToolboxItemFilter("MarqueeControlLibrary.MarqueeBorder", _
ToolboxItemType.Require)> _
Partial Public Class MarqueeBorder
    Inherits Panel
    Implements IMarqueeWidget

```

7. Declare two enumerations for managing the `MarqueeBorder` control's state: `MarqueeSpinDirection`, which determines the direction in which the lights "spin" around the border, and `MarqueeLightShape`, which determines the shape of the lights (square or circular). Place these declarations before the `MarqueeBorder` class declaration.

```

// This defines the possible values for the MarqueeBorder
// control's SpinDirection property.
public enum MarqueeSpinDirection
{
    CW,
    CCW
}

// This defines the possible values for the MarqueeBorder
// control's LightShape property.
public enum MarqueeLightShape
{
    Square,
    Circle
}

```

```

' This defines the possible values for the MarqueeBorder
' control's SpinDirection property.
Public Enum MarqueeSpinDirection
    CW
    CCW
End Enum

' This defines the possible values for the MarqueeBorder
' control's LightShape property.
Public Enum MarqueeLightShape
    Square
    Circle
End Enum

```

8. Declare the instance variables that correspond to the exposed properties, and initialize them in the constructor.

```
public static int MaxLightSize = 10;

// These fields back the public properties.
private int updatePeriodValue = 50;
private int lightSizeValue = 5;
private int lightPeriodValue = 3;
private int lightSpacingValue = 1;
private Color lightColorValue;
private Color darkColorValue;
private MarqueeSpinDirection spinDirectionValue = MarqueeSpinDirection.CW;
private MarqueeLightShape lightShapeValue = MarqueeLightShape.Square;

// These brushes are used to paint the light and dark
// colors of the marquee lights.
private Brush lightBrush;
private Brush darkBrush;

// This field tracks the progress of the "first" light as it
// "travels" around the marquee border.
private int currentOffset = 0;

// This component updates the control asynchronously.
private System.ComponentModel.BackgroundWorker backgroundWorker1;

public MarqueeBorder()
{
    // This call is required by the Windows.Forms Form Designer.
    InitializeComponent();

    // Initialize light and dark colors
    // to the control's default values.
    this.lightColorValue = this.ForeColor;
    this.darkColorValue = this.BackColor;
    this.lightBrush = new SolidBrush(this.lightColorValue);
    this.darkBrush = new SolidBrush(this.darkColorValue);

    // The MarqueeBorder control manages its own padding,
    // because it requires that any contained controls do
    // not overlap any of the marquee lights.
    int pad = 2 * (this.lightSizeValue + this.lightSpacingValue);
    this.Padding = new Padding(pad, pad, pad, pad);

    SetStyle(ControlStyles.OptimizedDoubleBuffer, true);
}
```

```

Public Shared MaxLightSize As Integer = 10

' These fields back the public properties.
Private updatePeriodValue As Integer = 50
Private lightSizeValue As Integer = 5
Private lightPeriodValue As Integer = 3
Private lightSpacingValue As Integer = 1
Private lightColorValue As Color
Private darkColorValue As Color
Private spinDirectionValue As MarqueeSpinDirection = MarqueeSpinDirection.CW
Private lightShapeValue As MarqueeLightShape = MarqueeLightShape.Square

' These brushes are used to paint the light and dark
' colors of the marquee lights.
Private lightBrush As Brush
Private darkBrush As Brush

' This field tracks the progress of the "first" light as it
' "travels" around the marquee border.
Private currentOffset As Integer = 0

' This component updates the control asynchronously.
Private WithEvents backgroundWorker1 As System.ComponentModel.BackgroundWorker

Public Sub New()
    ' This call is required by the Windows.Forms Form Designer.
    InitializeComponent()

    ' Initialize light and dark colors
    ' to the control's default values.
    Me.lightColorValue = Me.ForeColor
    Me.darkColorValue = Me.BackColor
    Me.lightBrush = New SolidBrush(Me.lightColorValue)
    Me.darkBrush = New SolidBrush(Me.darkColorValue)

    ' The MarqueeBorder control manages its own padding,
    ' because it requires that any contained controls do
    ' not overlap any of the marquee lights.
    Dim pad As Integer = 2 * (Me.lightSizeValue + Me.lightSpacingValue)
    Me.Padding = New Padding(pad, pad, pad, pad)

    SetStyle(ControlStyles.OptimizedDoubleBuffer, True)
End Sub

```

9. Implement the `IMarqueeWidget` interface.

The `StartMarquee` and `StopMarquee` methods invoke the `BackgroundWorker` component's `RunWorkerAsync` and `CancelAsync` methods to start and stop the animation.

Because the `MarqueeBorder` control can contain child controls, the `StartMarquee` method enumerates all child controls and calls `StartMarquee` on those that implement `IMarqueeWidget`. The `StopMarquee` method has a similar implementation.

```

public virtual void StartMarquee()
{
    // The MarqueeBorder control may contain any number of
    // controls that implement IMarqueeWidget, so find
    // each IMarqueeWidget child and call its StartMarquee
    // method.
    foreach (Control cntrl in this.Controls)
    {
        if (cntrl is IMarqueeWidget)
        {
            IMarqueeWidget widget = cntrl as IMarqueeWidget;
            widget.StartMarquee();
        }
    }

    // Start the updating thread and pass it the UpdatePeriod.
    this.backgroundWorker1.RunWorkerAsync(this.UpdatePeriod);
}

public virtual void StopMarquee()
{
    // The MarqueeBorder control may contain any number of
    // controls that implement IMarqueeWidget, so find
    // each IMarqueeWidget child and call its StopMarquee
    // method.
    foreach (Control cntrl in this.Controls)
    {
        if (cntrl is IMarqueeWidget)
        {
            IMarqueeWidget widget = cntrl as IMarqueeWidget;
            widget.StopMarquee();
        }
    }

    // Stop the updating thread.
    this.backgroundWorker1.CancelAsync();
}

[Category("Marquee")]
[Browsable(true)]
public virtual int UpdatePeriod
{
    get
    {
        return this.updatePeriodValue;
    }

    set
    {
        if (value > 0)
        {
            this.updatePeriodValue = value;
        }
        else
        {
            throw new ArgumentOutOfRangeException("UpdatePeriod", "must be > 0");
        }
    }
}

```

```

Public Overridable Sub StartMarquee() _
Implements IMarqueeWidget.StartMarquee
    ' The MarqueeBorder control may contain any number of
    ' controls that implement IMarqueeWidget, so find
    ' each IMarqueeWidget child and call its StartMarquee
    ' method.
    Dim cntrl As Control
    For Each cntrl In Me.Controls
        If TypeOf cntrl Is IMarqueeWidget Then
            Dim widget As IMarqueeWidget = CType(cntrl, IMarqueeWidget)

                widget.StartMarquee()
            End If
        Next cntrl

        ' Start the updating thread and pass it the UpdatePeriod.
        Me.backgroundWorker1.RunWorkerAsync(Me.UpdatePeriod)
    End Sub

    Public Overridable Sub StopMarquee() _
Implements IMarqueeWidget.StopMarquee
    ' The MarqueeBorder control may contain any number of
    ' controls that implement IMarqueeWidget, so find
    ' each IMarqueeWidget child and call its StopMarquee
    ' method.
    Dim cntrl As Control
    For Each cntrl In Me.Controls
        If TypeOf cntrl Is IMarqueeWidget Then
            Dim widget As IMarqueeWidget = CType(cntrl, IMarqueeWidget)

                widget.StopMarquee()
            End If
        Next cntrl

        ' Stop the updating thread.
        Me.backgroundWorker1.CancelAsync()
    End Sub

<Category("Marquee"), Browsable(True)> _
Public Overridable Property UpdatePeriod() As Integer _
Implements IMarqueeWidget.UpdatePeriod

    Get
        Return Me.updatePeriodValue
    End Get

    Set(ByVal Value As Integer)
        If Value > 0 Then
            Me.updatePeriodValue = Value
        Else
            Throw New ArgumentOutOfRangeException("UpdatePeriod", _
                "must be > 0")
        End If
    End Set

    End Property

```

10. Implement the property accessors. The `MarqueeBorder` control has several properties for controlling its appearance.

```

[Category("Marquee")]
[Browsable(true)]
public int LightSize
{

```

```

        get
    {
        return this.lightSizeValue;
    }

    set
    {
        if (value > 0 && value <= MaxLightSize)
        {
            this.lightSizeValue = value;
            this.DockPadding.All = 2 * value;
        }
        else
        {
            throw new ArgumentOutOfRangeException("LightSize", "must be > 0 and < MaxLightSize");
        }
    }
}

[Category("Marquee")]
[Browsable(true)]
public int LightPeriod
{
    get
    {
        return this.lightPeriodValue;
    }

    set
    {
        if (value > 0)
        {
            this.lightPeriodValue = value;
        }
        else
        {
            throw new ArgumentOutOfRangeException("LightPeriod", "must be > 0 ");
        }
    }
}

[Category("Marquee")]
[Browsable(true)]
public Color LightColor
{
    get
    {
        return this.lightColorValue;
    }

    set
    {
        // The LightColor property is only changed if the
        // client provides a different value. Comparing values
        // from the ToArgb method is the recommended test for
        // equality between Color structs.
        if (this.lightColorValue.ToArgb() != value.ToArgb())
        {
            this.lightColorValue = value;
            this.lightBrush = new SolidBrush(value);
        }
    }
}

[Category("Marquee")]
[Browsable(true)]
public Color DarkColor
{
}

```

```

        get
    {
        return this.darkColorValue;
    }

    set
    {
        // The DarkColor property is only changed if the
        // client provides a different value. Comparing values
        // from the ToArgb method is the recommended test for
        // equality between Color structs.
        if (this.darkColorValue.ToArgb() != value.ToArgb())
        {
            this.darkColorValue = value;
            this.darkBrush = new SolidBrush(value);
        }
    }
}

[Category("Marquee")]
[Browsable(true)]
public int LightSpacing
{
    get
    {
        return this.lightSpacingValue;
    }

    set
    {
        if (value >= 0)
        {
            this.lightSpacingValue = value;
        }
        else
        {
            throw new ArgumentOutOfRangeException("LightSpacing", "must be >= 0");
        }
    }
}

[Category("Marquee")]
[Browsable(true)]
[EditorAttribute(typeof(LightShapeEditor),
    typeof(System.Drawing.Design.UITypeEditor))]
public MarqueeLightShape LightShape
{
    get
    {
        return this.lightShapeValue;
    }

    set
    {
        this.lightShapeValue = value;
    }
}

[Category("Marquee")]
[Browsable(true)]
public MarqueeSpinDirection SpinDirection
{
    get
    {
        return this.spinDirectionValue;
    }

    set
    {

```

```
        this.spinDirectionValue = value;
    }
}
```

```
<Category("Marquee"), Browsable(True)> _
Public Property LightSize() As Integer
    Get
        Return Me.lightSizeValue
    End Get

    Set(ByVal Value As Integer)
        If Value > 0 AndAlso Value <= MaxLightSize Then
            Me.lightSizeValue = Value
            Me.DockPadding.All = 2 * Value
        Else
            Throw New ArgumentOutOfRangeException("LightSize", _
                "must be > 0 and < MaxLightSize")
        End If
    End Set
End Property
```

```
<Category("Marquee"), Browsable(True)> _
Public Property LightPeriod() As Integer
    Get
        Return Me.lightPeriodValue
    End Get

    Set(ByVal Value As Integer)
        If Value > 0 Then
            Me.lightPeriodValue = Value
        Else
            Throw New ArgumentOutOfRangeException("LightPeriod", _
                "must be > 0 ")
        End If
    End Set
End Property
```

```
<Category("Marquee"), Browsable(True)> _
Public Property LightColor() As Color
    Get
        Return Me.lightColorValue
    End Get

    Set(ByVal Value As Color)
        ' The LightColor property is only changed if the
        ' client provides a different value. Comparing values
        ' from the ToArgb method is the recommended test for
        ' equality between Color structs.
        If Me.lightColorValue.ToArgb() <> Value.ToArgb() Then
            Me.lightColorValue = Value
            Me.lightBrush = New SolidBrush(Value)
        End If
    End Set
End Property
```

```
<Category("Marquee"), Browsable(True)> _
Public Property DarkColor() As Color
    Get
        Return Me.darkColorValue
    End Get

    Set(ByVal Value As Color)
        ' The DarkColor property is only changed if the
        ' client provides a different value. Comparing values
```

```

    ' client provides a different value. Comparing values
    ' from the ToArgb method is the recommended test for
    ' equality between Color structs.
    If Me.darkColorValue.ToInt32() <> Value.ToInt32() Then
        Me.darkColorValue = Value
        Me.darkBrush = New SolidBrush(Value)
    End If
End Set
End Property

<Category("Marquee"), Browsable(True)> _
Public Property LightSpacing() As Integer
Get
    Return Me.lightSpacingValue
End Get

Set(ByVal Value As Integer)
    If Value >= 0 Then
        Me.lightSpacingValue = Value
    Else
        Throw New ArgumentOutOfRangeException("LightSpacing", _
            "must be >= 0")
    End If
End Set
End Property

<Category("Marquee"), Browsable(True), _
EditorAttribute(GetType(LightShapeEditor), _
GetType(System.Drawing.Design.UITypeEditor))> _
Public Property LightShape() As MarqueeLightShape

Get
    Return Me.lightShapeValue
End Get

Set(ByVal Value As MarqueeLightShape)
    Me.lightShapeValue = Value
End Set

End Property

<Category("Marquee"), Browsable(True)> _
Public Property SpinDirection() As MarqueeSpinDirection

Get
    Return Me.spinDirectionValue
End Get

Set(ByVal Value As MarqueeSpinDirection)
    Me.spinDirectionValue = Value
End Set

End Property

```

11. Implement the handlers for the `BackgroundWorker` component's `DoWork` and `ProgressChanged` events.

The `DoWork` event handler sleeps for the number of milliseconds specified by `UpdatePeriod` then raises the `ProgressChanged` event, until your code stops the animation by calling `CancelAsync`.

The `ProgressChanged` event handler increments the position of the "base" light, from which the light/dark state of the other lights is determined, and calls the `Refresh` method to cause the control to repaint itself.

```
// This method is called in the worker thread's context,
// so it must not make any calls into the MarqueeBorder
// control. Instead, it communicates to the control using
// the ProgressChanged event.
//
// The only work done in this event handler is
// to sleep for the number of milliseconds specified
// by UpdatePeriod, then raise the ProgressChanged event.
private void backgroundWorker1_DoWork(object sender, System.ComponentModel.DoWorkEventArgs e)
{
    BackgroundWorker worker = sender as BackgroundWorker;

    // This event handler will run until the client cancels
    // the background task by calling CancelAsync.
    while (!worker.CancellationPending)
    {
        // The Argument property of the DoWorkEventArgs
        // object holds the value of UpdatePeriod, which
        // was passed as the argument to the RunWorkerAsync
        // method.
        Thread.Sleep((int)e.Argument);

        // The DoWork eventhandler does not actually report
        // progress; the ReportProgress event is used to
        // periodically alert the control to update its state.
        worker.ReportProgress(0);
    }
}

// The ProgressChanged event is raised by the DoWork method.
// This event handler does work that is internal to the
// control. In this case, the currentOffset is incremented,
// and the control is told to repaint itself.
private void backgroundWorker1_ProgressChanged(
    object sender,
    System.ComponentModel.ProgressChangedEventArgs e)
{
    this.currentOffset++;
    this.Refresh();
}
```

```

' This method is called in the worker thread's context,
' so it must not make any calls into the MarqueeBorder
' control. Instead, it communicates to the control using
' the ProgressChanged event.
'

' The only work done in this event handler is
' to sleep for the number of milliseconds specified
' by UpdatePeriod, then raise the ProgressChanged event.
Private Sub backgroundWorker1_DoWork( _
    ByVal sender As Object, _
    ByVal e As System.ComponentModel.DoWorkEventArgs) _
Handles backgroundWorker1.DoWork
    Dim worker As BackgroundWorker = CType(sender, BackgroundWorker)

    ' This event handler will run until the client cancels
    ' the background task by calling CancelAsync.
    While Not worker.CancellationPending
        ' The Argument property of the DoWorkEventArgs
        ' object holds the value of UpdatePeriod, which
        ' was passed as the argument to the RunWorkerAsync
        ' method.
        Thread.Sleep(Fix(e.Argument))

        ' The DoWork eventhandler does not actually report
        ' progress; the ReportProgress event is used to
        ' periodically alert the control to update its state.
        worker.ReportProgress(0)
    End While
End Sub

'

' The ProgressChanged event is raised by the DoWork method.
' This event handler does work that is internal to the
' control. In this case, the currentOffset is incremented,
' and the control is told to repaint itself.
Private Sub backgroundWorker1_ProgressChanged( _
    ByVal sender As Object, _
    ByVal e As System.ComponentModel.ProgressChangedEventArgs) _
Handles backgroundWorker1.ProgressChanged
    Me.currentOffset += 1
    Me.Refresh()
End Sub

```

12. Implement the helper methods, `IsLit` and `DrawLight`.

The `IsLit` method determines the color of a light at a given position. Lights that are "lit" are drawn in the color given by the `LightColor` property, and those that are "dark" are drawn in the color given by the `DarkColor` property.

The `DrawLight` method draws a light using the appropriate color, shape, and position.

```
// This method determines if the marquee light at lightIndex
// should be lit. The currentOffset field specifies where
// the "first" light is located, and the "position" of the
// light given by lightIndex is computed relative to this
// offset. If this position modulo lightPeriodValue is zero,
// the light is considered to be on, and it will be painted
// with the control's lightBrush.
protected virtual bool IsLit(int lightIndex)
{
    int directionFactor =
        (this.spinDirectionValue == MarqueeSpinDirection.CW ? -1 : 1);

    return (
        (lightIndex + directionFactor * this.currentOffset) % this.lightPeriodValue == 0
    );
}

protected virtual void DrawLight(
    Graphics g,
    Brush brush,
    int xPos,
    int yPos)
{
    switch (this.lightShapeValue)
    {
        case MarqueeLightShape.Square:
        {
            g.FillRectangle(brush, xPos, yPos, this.lightSizeValue, this.lightSizeValue);
            break;
        }
        case MarqueeLightShape.Circle:
        {
            g.FillEllipse(brush, xPos, yPos, this.lightSizeValue, this.lightSizeValue);
            break;
        }
        default:
        {
            Trace.Assert(false, "Unknown value for light shape.");
            break;
        }
    }
}
```

```

' This method determines if the marquee light at lightIndex
' should be lit. The currentOffset field specifies where
' the "first" light is located, and the "position" of the
' light given by lightIndex is computed relative to this
' offset. If this position modulo lightPeriodValue is zero,
' the light is considered to be on, and it will be painted
' with the control's lightBrush.

Protected Overridable Function IsLit(ByVal lightIndex As Integer) As Boolean
    Dim directionFactor As Integer = _
        IIf(Me.spinDirectionValue = MarqueeSpinDirection.CW, -1, 1)

    Return (lightIndex + directionFactor * Me.currentOffset) Mod Me.lightPeriodValue = 0
End Function

Protected Overridable Sub DrawLight( _
    ByVal g As Graphics, _
    ByVal brush As Brush, _
    ByVal xPos As Integer, _
    ByVal yPos As Integer)

    Select Case Me.lightShapeValue
        Case MarqueeLightShape.Square
            g.FillRectangle( _
                brush, _
                xPos, _
                yPos, _
                Me.lightSizeValue, _
                Me.lightSizeValue)
        Exit Select
        Case MarqueeLightShape.Circle
            g.FillEllipse( _
                brush, _
                xPos, _
                yPos, _
                Me.lightSizeValue, _
                Me.lightSizeValue)
        Exit Select
        Case Else
            Trace.Assert(False, "Unknown value for light shape.")
        Exit Select
    End Select

    End Sub

```

13. Override the [OnLayout](#) and [OnPaint](#) methods.

The [OnPaint](#) method draws the lights along the edges of the [MarqueeBorder](#) control.

Because the [OnPaint](#) method depends on the dimensions of the [MarqueeBorder](#) control, you need to call it whenever the layout changes. To achieve this, override [OnLayout](#) and call [Refresh](#).

```

protected override void OnLayout(LayoutEventArgs levent)
{
    base.OnLayout(levent);

    // Repaint when the layout has changed.
    this.Refresh();
}

// This method paints the lights around the border of the
// control. It paints the top row first, followed by the
// right side, the bottom row, and the left side. The color
// of each light is determined by the IsLit method and
// depends on the light's position relative to the value
// of currentOffset

```

```
// OF CURRENTLIGHTSET.  
protected override void OnPaint(PaintEventArgs e)  
{  
    Graphics g = e.Graphics;  
    g.Clear(this.BackColor);  
  
    base.OnPaint(e);  
  
    // If the control is large enough, draw some lights.  
    if (this.Width > MaxLightSize &&  
        this.Height > MaxLightSize)  
    {  
        // The position of the next light will be incremented  
        // by this value, which is equal to the sum of the  
        // light size and the space between two lights.  
        int increment =  
            this.lightSizeValue + this.lightSpacingValue;  
  
        // Compute the number of lights to be drawn along the  
        // horizontal edges of the control.  
        int horizontalLights =  
            (this.Width - increment) / increment;  
  
        // Compute the number of lights to be drawn along the  
        // vertical edges of the control.  
        int verticalLights =  
            (this.Height - increment) / increment;  
  
        // These local variables will be used to position and  
        // paint each light.  
        int xPos = 0;  
        int yPos = 0;  
        int lightCounter = 0;  
        Brush brush;  
  
        // Draw the top row of lights.  
        for (int i = 0; i < horizontalLights; i++)  
        {  
            brush = IsLit(lightCounter) ? this.lightBrush : this.darkBrush;  
  
            DrawLight(g, brush, xPos, yPos);  
  
            xPos += increment;  
            lightCounter++;  
        }  
  
        // Draw the lights flush with the right edge of the control.  
        xPos = this.Width - this.lightSizeValue;  
  
        // Draw the right column of lights.  
        for (int i = 0; i < verticalLights; i++)  
        {  
            brush = IsLit(lightCounter) ? this.lightBrush : this.darkBrush;  
  
            DrawLight(g, brush, xPos, yPos);  
  
            yPos += increment;  
            lightCounter++;  
        }  
  
        // Draw the lights flush with the bottom edge of the control.  
        yPos = this.Height - this.lightSizeValue;  
  
        // Draw the bottom row of lights.  
        for (int i = 0; i < horizontalLights; i++)  
        {  
            brush = IsLit(lightCounter) ? this.lightBrush : this.darkBrush;  
  
            DrawLight(g, brush, xPos, yPos);  
        }  
    }  
}
```

```

        xPos -= increment;
        lightCounter++;
    }

    // Draw the lights flush with the left edge of the control.
    xPos = 0;

    // Draw the left column of lights.
    for (int i = 0; i < verticalLights; i++)
    {
        brush = IsLit(lightCounter) ? this.lightBrush : this.darkBrush;

        DrawLight(g, brush, xPos, yPos);

        yPos -= increment;
        lightCounter++;
    }
}
}
}

```

```

Protected Overrides Sub OnLayout(ByVal levent As LayoutEventArgs)
    MyBase.OnLayout(levent)

    ' Repaint when the layout has changed.
    Me.Refresh()
End Sub

' This method paints the lights around the border of the
' control. It paints the top row first, followed by the
' right side, the bottom row, and the left side. The color
' of each light is determined by the IsLit method and
' depends on the light's position relative to the value
' of currentOffset.
Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
    Dim g As Graphics = e.Graphics
    g.Clear(Me.BackColor)

    MyBase.OnPaint(e)

    ' If the control is large enough, draw some lights.
    If Me.Width > MaxLightSize AndAlso Me.Height > MaxLightSize Then
        ' The position of the next light will be incremented
        ' by this value, which is equal to the sum of the
        ' light size and the space between two lights.
        Dim increment As Integer = _
            Me.lightSizeValue + Me.lightSpacingValue

        ' Compute the number of lights to be drawn along the
        ' horizontal edges of the control.
        Dim horizontalLights As Integer = _
            (Me.Width - increment) / increment

        ' Compute the number of lights to be drawn along the
        ' vertical edges of the control.
        Dim verticalLights As Integer = _
            (Me.Height - increment) / increment

        ' These local variables will be used to position and
        ' paint each light.
        Dim xPos As Integer = 0
        Dim yPos As Integer = 0
        Dim lightCounter As Integer = 0
        Dim brush As Brush

        ' Draw the top row of lights.
        xPos = increment;
        lightCounter++;
    }

    // Draw the lights flush with the left edge of the control.
    xPos = 0;

    // Draw the left column of lights.
    for (int i = 0; i < verticalLights; i++)
    {
        brush = IsLit(lightCounter) ? this.lightBrush : this.darkBrush;

        DrawLight(g, brush, xPos, yPos);

        yPos -= increment;
        lightCounter++;
    }
}
}
}

```

```

    Dim i As Integer
    For i = 0 To horizontalLights - 1
        brush = IIf(IsLit(lightCounter), Me.lightBrush, Me.darkBrush)

        DrawLight(g, brush, xPos, yPos)

        xPos += increment
        lightCounter += 1
    Next i

    ' Draw the lights flush with the right edge of the control.
    xPos = Me.Width - Me.lightSizeValue

    ' Draw the right column of lights.
    'Dim i As Integer
    For i = 0 To verticalLights - 1
        brush = IIf(IsLit(lightCounter), Me.lightBrush, Me.darkBrush)

        DrawLight(g, brush, xPos, yPos)

        yPos += increment
        lightCounter += 1
    Next i

    ' Draw the lights flush with the bottom edge of the control.
    yPos = Me.Height - Me.lightSizeValue

    ' Draw the bottom row of lights.
    'Dim i As Integer
    For i = 0 To horizontalLights - 1
        brush = IIf(IsLit(lightCounter), Me.lightBrush, Me.darkBrush)

        DrawLight(g, brush, xPos, yPos)

        xPos -= increment
        lightCounter += 1
    Next i

    ' Draw the lights flush with the left edge of the control.
    xPos = 0

    ' Draw the left column of lights.
    'Dim i As Integer
    For i = 0 To verticalLights - 1
        brush = IIf(IsLit(lightCounter), Me.lightBrush, Me.darkBrush)

        DrawLight(g, brush, xPos, yPos)

        yPos -= increment
        lightCounter += 1
    Next i
End If
End Sub

```

Creating a Custom Designer to Shadow and Filter Properties

The `MarqueeControlRootDesigner` class provides the implementation for the root designer. In addition to this designer, which operates on the `MarqueeControl`, you will need a custom designer that is specifically associated with the `MarqueeBorder` control. This designer provides custom behavior that is appropriate in the context of the custom root designer.

Specifically, the `MarqueeBorderDesigner` will "shadow" and filter certain properties on the `MarqueeBorder` control, changing their interaction with the design environment.

Intercepting calls to a component's property accessor is known as "shadowing." It allows a designer to track the

value set by the user and optionally pass that value to the component being designed.

For this example, the [Visible](#) and [Enabled](#) properties will be shadowed by the [MarqueeBorderDesigner](#), which prevents the user from making the [MarqueeBorder](#) control invisible or disabled during design time.

Designers can also add and remove properties. For this example, the [Padding](#) property will be removed at design time, because the [MarqueeBorder](#) control programmatically sets the padding based on the size of the lights specified by the [LightSize](#) property.

The base class for [MarqueeBorderDesigner](#) is [ComponentDesigner](#), which has methods that can change the attributes, properties, and events exposed by a control at design time:

- [PreFilterProperties](#)
- [PostFilterProperties](#)
- [PreFilterAttributes](#)
- [PostFilterAttributes](#)
- [PreFilterEvents](#)
- [PostFilterEvents](#)

When changing the public interface of a component using these methods, you must follow these rules:

- Add or remove items in the [PreFilter](#) methods only
- Modify existing items in the [PostFilter](#) methods only
- Always call the base implementation first in the [PreFilter](#) methods
- Always call the base implementation last in the [PostFilter](#) methods

Adhering to these rules ensures that all designers in the design-time environment have a consistent view of all components being designed.

The [ComponentDesigner](#) class provides a dictionary for managing the values of shadowed properties, which relieves you of the need to create specific instance variables.

To create a custom designer to shadow and filter properties

1. Right-click the **Design** folder and add a new class. Give the source file a base name of "MarqueeBorderDesigner."
2. Open the [MarqueeBorderDesigner](#) source file in the **Code Editor**. At the top of the file, import the following namespaces:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.ComponentModel.Design;
using System.Diagnostics;
using System.Windows.Forms;
using System.Windows.Forms.Design;
```

```
Imports System
Imports System.Collections
Imports System.ComponentModel
Imports System.ComponentModel.Design
Imports System.Diagnostics
Imports System.Windows.Forms
Imports System.Windows.Forms.Design
```

3. Change the declaration of `MarqueeBorderDesigner` to inherit from `ParentControlDesigner`.

Because the `MarqueeBorder` control can contain child controls, `MarqueeBorderDesigner` inherits from `ParentControlDesigner`, which handles the parent-child interaction.

```
namespace MarqueeControlLibrary.Design
{
    [System.Security.Permissions.PermissionSet(System.Security.Permissions.SecurityAction.Demand, Name = "FullTrust")]
    public class MarqueeBorderDesigner : ParentControlDesigner
    {
```

```
Namespace MarqueeControlLibrary.Design

<System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.Demand, Name:="FullTrust")> _
Public Class MarqueeBorderDesigner
    Inherits ParentControlDesigner
```

4. Override the base implementation of `PreFilterProperties`.

```
protected override void PreFilterProperties(IDictionary properties)
{
    base.PreFilterProperties(properties);

    if (properties.Contains("Padding"))
    {
        properties.Remove("Padding");
    }

    properties["Visible"] = TypeDescriptor.CreateProperty(
        typeof(MarqueeBorderDesigner),
        (PropertyDescriptor)properties["Visible"],
        new Attribute[0]);

    properties["Enabled"] = TypeDescriptor.CreateProperty(
        typeof(MarqueeBorderDesigner),
        (PropertyDescriptor)properties["Enabled"],
        new Attribute[0]);
}
```

```

Protected Overrides Sub PreFilterProperties( _
 ByVal properties As IDictionary)

    MyBase.PreFilterProperties(properties)

    If properties.Contains("Padding") Then
        properties.Remove("Padding")
    End If

    properties("Visible") = _
    TypeDescriptor.CreateProperty(GetType(MarqueeBorderDesigner), _
    CType(properties("Visible"), PropertyDescriptor), _
    New Attribute(-1) {})

    properties("Enabled") = _
    TypeDescriptor.CreateProperty(GetType(MarqueeBorderDesigner), _
    CType(properties("Enabled"), _
    PropertyDescriptor), _
    New Attribute(-1) {})

End Sub

```

5. Implement the [Enabled](#) and [Visible](#) properties. These implementations shadow the control's properties.

```

public bool Visible
{
    get
    {
        return (bool)ShadowProperties["Visible"];
    }
    set
    {
        this.ShadowProperties["Visible"] = value;
    }
}

public bool Enabled
{
    get
    {
        return (bool)ShadowProperties["Enabled"];
    }
    set
    {
        this.ShadowProperties["Enabled"] = value;
    }
}

```

```

Public Property Visible() As Boolean
    Get
        Return CBool(ShadowProperties("Visible"))
    End Get
    Set(ByVal Value As Boolean)
        Me.ShadowProperties("Visible") = Value
    End Set
End Property

Public Property Enabled() As Boolean
    Get
        Return CBool(ShadowProperties("Enabled"))
    End Get
    Set(ByVal Value As Boolean)
        Me.ShadowProperties("Enabled") = Value
    End Set
End Property

```

Handling Component Changes

The `MarqueeControlRootDesigner` class provides the custom design-time experience for your `MarqueeControl` instances. Most of the design-time functionality is inherited from the `DocumentDesigner` class; your code will implement two specific customizations: handling component changes, and adding designer verbs.

As users design their `MarqueeControl` instances, your root designer will track changes to the `MarqueeControl` and its child controls. The design-time environment offers a convenient service, `IComponentChangeService`, for tracking changes to component state.

You acquire a reference to this service by querying the environment with the `GetService` method. If the query is successful, your designer can attach a handler for the `ComponentChanged` event and perform whatever tasks are required to maintain a consistent state at design time.

In the case of the `MarqueeControlRootDesigner` class, you will call the `Refresh` method on each `IMarqueeWidget` object contained by the `MarqueeControl`. This will cause the `IMarqueeWidget` object to repaint itself appropriately when properties like its parent's `Size` are changed.

To handle component changes

1. Open the `MarqueeControlRootDesigner` source file in the **Code Editor** and override the `Initialize` method. Call the base implementation of `Initialize` and query for the `IComponentChangeService`.

```

base.Initialize(component);

IComponentChangeService cs =
    GetService(typeof(IComponentChangeService))
    as IComponentChangeService;

if (cs != null)
{
    cs.ComponentChanged +=
        new ComponentChangedEventHandler(OnComponentChanged);
}

```

```

 MyBase.Initialize(component)

 Dim cs As IComponentChangeService = _
 CType(GetService(GetType(IComponentChangeService)), _
 IComponentChangeService)

 If (cs IsNot Nothing) Then
     AddHandler cs.ComponentChanged, AddressOf OnComponentChanged
 End If

```

2. Implement the [OnComponentChanged](#) event handler. Test the sending component's type, and if it is an [IMarqueeWidget](#), call its [Refresh](#) method.

```

private void OnComponentChanged(
    object sender,
    ComponentChangedEventArgs e)
{
    if (e.Component is IMarqueeWidget)
    {
        this.Control.Refresh();
    }
}

```

```

Private Sub OnComponentChanged( _
    ByVal sender As Object, _
    ByVal e As ComponentChangedEventArgs)
    If TypeOf e.Component Is IMarqueeWidget Then
        Me.Control.Refresh()
    End If
End Sub

```

Adding Designer Verbs to your Custom Designer

A designer verb is a menu command linked to an event handler. Designer verbs are added to a component's shortcut menu at design time. For more information, see [DesignerVerb](#).

You will add two designer verbs to your designers: **Run Test** and **Stop Test**. These verbs will allow you to view the run-time behavior of the [MarqueeControl](#) at design time. These verbs will be added to the [MarqueeControlRootDesigner](#).

When **Run Test** is invoked, the verb event handler will call the [StartMarquee](#) method on the [MarqueeControl](#).

When **Stop Test** is invoked, the verb event handler will call the [StopMarquee](#) method on the [MarqueeControl](#). The implementation of the [StartMarquee](#) and [StopMarquee](#) methods call these methods on contained controls that implement [IMarqueeWidget](#), so any contained [IMarqueeWidget](#) controls will also participate in the test.

To add designer verbs to your custom designers

1. In the [MarqueeControlRootDesigner](#) class, add event handlers named [OnVerbRunTest](#) and [OnVerbStopTest](#).

```

private void OnVerbRunTest(object sender, EventArgs e)
{
    MarqueeControl c = this.Control as MarqueeControl;

    c.Start();
}

private void OnVerbStopTest(object sender, EventArgs e)
{
    MarqueeControl c = this.Control as MarqueeControl;

    c.Stop();
}

```

```

Private Sub OnVerbRunTest( _
    ByVal sender As Object, _
    ByVal e As EventArgs)

    Dim c As MarqueeControl = CType(Me.Control, MarqueeControl)
    c.Start()

End Sub

Private Sub OnVerbStopTest( _
    ByVal sender As Object, _
    ByVal e As EventArgs)

    Dim c As MarqueeControl = CType(Me.Control, MarqueeControl)
    c.Stop()

End Sub

```

2. Connect these event handlers to their corresponding designer verbs. `MarqueeControlRootDesigner` inherits a `DesignerVerbCollection` from its base class. You will create two new `DesignerVerb` objects and add them to this collection in the `Initialize` method.

```

this.Verbs.Add(
    new DesignerVerb("Run Test",
        new EventHandler(OnVerbRunTest))
);

this.Verbs.Add(
    new DesignerVerb("Stop Test",
        new EventHandler(OnVerbStopTest))
);

```

```

Me.Verbs.Add(New DesignerVerb("Run Test", _
    New EventHandler(AddressOf OnVerbRunTest)))

Me.Verbs.Add(New DesignerVerb("Stop Test", _
    New EventHandler(AddressOf OnVerbStopTest)))

```

Creating a Custom UITypeEditor

When you create a custom design-time experience for users, it is often desirable to create a custom interaction with the Properties window. You can accomplish this by creating a `UITypeEditor`. For more information, see [How to: Create a UI Type Editor](#).

The `MarqueeBorder` control exposes several properties in the Properties window. Two of these properties,

`MarqueeSpinDirection` and `MarqueeLightShape` are represented by enumerations. To illustrate the use of a UI type editor, the `MarqueeLightShape` property will have an associated `UITypeEditor` class.

To create a custom UI type editor

1. Open the `MarqueeBorder` source file in the **Code Editor**.
2. In the definition of the `MarqueeBorder` class, declare a class called `LightShapeEditor` that derives from `UITypeEditor`.

```
// This class demonstrates the use of a custom UITypeEditor.  
// It allows the MarqueeBorder control's LightShape property  
// to be changed at design time using a customized UI element  
// that is invoked by the Properties window. The UI is provided  
// by the LightShapeSelectionControl class.  
internal class LightShapeEditor : UITypeEditor  
{
```

```
' This class demonstrates the use of a custom UITypeEditor.  
' It allows the MarqueeBorder control's LightShape property  
' to be changed at design time using a customized UI element  
' that is invoked by the Properties window. The UI is provided  
' by the LightShapeSelectionControl class.  
Friend Class LightShapeEditor  
    Inherits UITypeEditor
```

3. Declare an `IWindowsFormsEditorService` instance variable called `editorService`.

```
private IWindowsFormsEditorService editorService = null;
```

```
Private editorService As IWindowsFormsEditorService = Nothing
```

4. Override the `GetEditText` method. This implementation returns `DropDown`, which tells the design environment how to display the `LightShapeEditor`.

```
public override UITypeEditorEditStyle GetEditText(  
System.ComponentModel.ITypeDescriptorContext context)  
{  
    return UITypeEditorEditStyle.DropDown;  
}
```

```
Public Overrides Function GetEditText( _  
ByVal context As System.ComponentModel.ITypeDescriptorContext) _  
As UITypeEditorEditStyle  
    Return UITypeEditorEditStyle.DropDown  
End Function
```

5. Override the `EditValue` method. This implementation queries the design environment for an `IWindowsFormsEditorService` object. If successful, it creates a `LightShapeSelectionControl`. The `DropDownControl` method is invoked to start the `LightShapeEditor`. The return value from this invocation is returned to the design environment.

```

public override object EditValue(
    ITypeDescriptorContext context,
    IServiceProvider provider,
    object value)
{
    if (provider != null)
    {
        editorService =
            provider.GetService(
                typeof(IWindowsFormsEditorService))
            as IWindowsFormsEditorService;
    }

    if (editorService != null)
    {
        LightShapeSelectionControl selectionControl =
            new LightShapeSelectionControl(
                (MarqueeLightShape)value,
                editorService);

        editorService.DropDownControl(selectionControl);

        value = selectionControl.LightShape;
    }

    return value;
}

```

```

Public Overrides Function EditValue( _
    ByVal context As ITypeDescriptorContext, _
    ByVal provider As IServiceProvider, _
    ByVal value As Object) As Object
    If (provider IsNot Nothing) Then
        editorService = _
            CType(provider.GetService(GetType(IWindowsFormsEditorService)), _
            IWindowsFormsEditorService)
    End If

    If (editorService IsNot Nothing) Then
        Dim selectionControl As _
            New LightShapeSelectionControl( _
            CType(value, MarqueeLightShape), _
            editorService)

        editorService.DropDownControl(selectionControl)

        value = selectionControl.LightShape
    End If

    Return value
End Function

```

Creating a View Control for your Custom UITypeEditor

1. The `MarqueeLightShape` property supports two types of light shapes: `Square` and `Circle`. You will create a custom control used solely for the purpose of graphically displaying these values in the Properties window. This custom control will be used by your `UITypeEditor` to interact with the Properties window.

To create a view control for your custom UI type editor

1. Add a new `UserControl` item to the `MarqueeControlLibrary` project. Give the new source file a base name of "LightShapeSelectionControl."

2. Drag two **Panel** controls from the **Toolbox** onto the **LightShapeSelectionControl**. Name them **squarePanel** and **circlePanel**. Arrange them side by side. Set the **Size** property of both **Panel** controls to (60, 60). Set the **Location** property of the **squarePanel** control to (8, 10). Set the **Location** property of the **circlePanel** control to (80, 10). Finally, set the **Size** property of the **LightShapeSelectionControl** to (150, 80).
3. Open the **LightShapeSelectionControl** source file in the **Code Editor**. At the top of the file, import the **System.Windows.Forms.Design** namespace:

```
Imports System.Windows.Forms.Design
```

```
using System.Windows.Forms.Design;
```

1. Implement **Click** event handlers for the **squarePanel** and **circlePanel** controls. These methods invoke **CloseDropDown** to end the custom **UITypeEditor** editing session.

```
private void squarePanel_Click(object sender, EventArgs e)
{
    this.lightShapeValue = MarqueeLightShape.Square;

    this.Invalidate( false );

    this.editorService.CloseDropDown();
}

private void circlePanel_Click(object sender, EventArgs e)
{
    this.lightShapeValue = MarqueeLightShape.Circle;

    this.Invalidate( false );

    this.editorService.CloseDropDown();
}
```

```
Private Sub squarePanel_Click( _
    ByVal sender As Object, _
    ByVal e As EventArgs)

    Me.lightShapeValue = MarqueeLightShape.Square
    Me.Invalidate(False)
    Me.editorService.CloseDropDown()

End Sub

Private Sub circlePanel_Click( _
    ByVal sender As Object, _
    ByVal e As EventArgs)

    Me.lightShapeValue = MarqueeLightShape.Circle
    Me.Invalidate(False)
    Me.editorService.CloseDropDown()

End Sub
```

2. Declare an **IWindowsFormsEditorService** instance variable called **editorService**.

```
Private editorService As IWindowsFormsEditorService
```

```
private IWindowsFormsEditorService editorService;
```

1. Declare a `MarqueeLightShape` instance variable called `lightShapeValue`.

```
private MarqueeLightShape lightShapeValue = MarqueeLightShape.Square;
```

```
Private lightShapeValue As MarqueeLightShape = MarqueeLightShape.Square
```

2. In the `LightShapeSelectionControl` constructor, attach the `Click` event handlers to the `squarePanel` and `circlePanel` controls' `Click` events. Also, define a constructor overload that assigns the `MarqueeLightShape` value from the design environment to the `lightShapeValue` field.

```
// This constructor takes a MarqueeLightShape value from the
// design-time environment, which will be used to display
// the initial state.
public LightShapeSelectionControl(
    MarqueeLightShape lightShape,
    IWindowsFormsEditorService editorService )
{
    // This call is required by the designer.
    InitializeComponent();

    // Cache the light shape value provided by the
    // design-time environment.
    this.lightShapeValue = lightShape;

    // Cache the reference to the editor service.
    this.editorService = editorService;

    // Handle the Click event for the two panels.
    this.squarePanel.Click += new EventHandler(squarePanel_Click);
    this.circlePanel.Click += new EventHandler(circlePanel_Click);
}
```

```
' This constructor takes a MarqueeLightShape value from the
' design-time environment, which will be used to display
' the initial state.
Public Sub New( _
    ByVal lightShape As MarqueeLightShape, _
    ByVal editorService As IWindowsFormsEditorService)
    ' This call is required by the Windows.Forms Form Designer.
    InitializeComponent()

    ' Cache the light shape value provided by the
    ' design-time environment.
    Me.lightShapeValue = lightShape

    ' Cache the reference to the editor service.
    Me.editorService = editorService

    ' Handle the Click event for the two panels.
    AddHandler Me.squarePanel.Click, AddressOf squarePanel_Click
    AddHandler Me.circlePanel.Click, AddressOf circlePanel_Click
End Sub
```

3. In the `Dispose` method, detach the `Click` event handlers.

```

protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        // Be sure to unhook event handlers
        // to prevent "lapsed listener" leaks.
        this.squarePanel.Click -=
            new EventHandler(squarePanel_Click);
        this.circlePanel.Click -=
            new EventHandler(circlePanel_Click);

        if(components != null)
        {
            components.Dispose();
        }
    }
    base.Dispose( disposing );
}

```

```

Protected Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing Then
        ' Be sure to unhook event handlers
        ' to prevent "lapsed listener" leaks.
        RemoveHandler Me.squarePanel.Click, AddressOf squarePanel_Click
        RemoveHandler Me.circlePanel.Click, AddressOf circlePanel_Click

        If (components IsNot Nothing) Then
            components.Dispose()
        End If

    End If
    MyBase.Dispose(disposing)
End Sub

```

4. In **Solution Explorer**, click the **Show All Files** button. Open the LightShapeSelectionControl.Designer.cs or LightShapeSelectionControl.Designer.vb file, and remove the default definition of the **Dispose** method.
5. Implement the **LightShape** property.

```

// LightShape is the property for which this control provides
// a custom user interface in the Properties window.
public MarqueeLightShape LightShape
{
    get
    {
        return this.lightShapeValue;
    }

    set
    {
        if( this.lightShapeValue != value )
        {
            this.lightShapeValue = value;
        }
    }
}

```

```
' LightShape is the property for which this control provides
' a custom user interface in the Properties window.
Public Property LightShape() As MarqueeLightShape

    Get
        Return Me.lightShapeValue
    End Get

    Set(ByVal Value As MarqueeLightShape)
        If Me.lightShapeValue <> Value Then
            Me.lightShapeValue = Value
        End If
    End Set

End Property
```

6. Override the [OnPaint](#) method. This implementation will draw a filled square and circle. It will also highlight the selected value by drawing a border around one shape or the other.

```
protected override void OnPaint(PaintEventArgs e)
{
    base.OnPaint (e);

    using(
        Graphics gSquare = this.squarePanel.CreateGraphics(),
        gCircle = this.circlePanel.CreateGraphics() )
    {
        // Draw a filled square in the client area of
        // the squarePanel control.
        gSquare.FillRectangle(
            Brushes.Red,
            0,
            0,
            this.squarePanel.Width,
            this.squarePanel.Height
        );

        // If the Square option has been selected, draw a
        // border inside the squarePanel.
        if( this.lightShapeValue == MarqueeLightShape.Square )
        {
            gSquare.DrawRectangle(
                Pens.Black,
                0,
                0,
                this.squarePanel.Width-1,
                this.squarePanel.Height-1);
        }

        // Draw a filled circle in the client area of
        // the circlePanel control.
        gCircle.Clear( this.circlePanel.BackColor );
        gCircle.FillEllipse(
            Brushes.Blue,
            0,
            0,
            this.circlePanel.Width,
            this.circlePanel.Height
        );

        // If the Circle option has been selected, draw a
        // border inside the circlePanel.
        if( this.lightShapeValue == MarqueeLightShape.Circle )
        {
            gCircle.DrawRectangle(
                Pens.Black,
                0,
                0,
                this.circlePanel.Width-1,
                this.circlePanel.Height-1);
        }
    }
}
```

```

Protected Overrides Sub OnPaint(ByVal e As PaintEventArgs)
    MyBase.OnPaint(e)

    Dim gCircle As Graphics = Me.circlePanel.CreateGraphics()
    Try
        Dim gSquare As Graphics = Me.squarePanel.CreateGraphics()
        Try
            ' Draw a filled square in the client area of
            ' the squarePanel control.
            gSquare.FillRectangle( _
                Brushes.Red, _
                0, _
                0, _
                Me.squarePanel.Width, _
                Me.squarePanel.Height)

            ' If the Square option has been selected, draw a
            ' border inside the squarePanel.
            If Me.lightShapeValue = MarqueeLightShape.Square Then
                gSquare.DrawRectangle( _
                    Pens.Black, _
                    0, _
                    0, _
                    Me.squarePanel.Width - 1, _
                    Me.squarePanel.Height - 1)
            End If

            ' Draw a filled circle in the client area of
            ' the circlePanel control.
            gCircle.Clear(Me.circlePanel.BackColor)
            gCircle.FillEllipse( _
                Brushes.Blue, _
                0, _
                0, _
                Me.circlePanel.Width, _
                Me.circlePanel.Height)

            ' If the Circle option has been selected, draw a
            ' border inside the circlePanel.
            If Me.lightShapeValue = MarqueeLightShape.Circle Then
                gCircle.DrawRectangle( _
                    Pens.Black, _
                    0, _
                    0, _
                    Me.circlePanel.Width - 1, _
                    Me.circlePanel.Height - 1)
            End If
        Finally
            gSquare.Dispose()
        End Try
    Finally
        gCircle.Dispose()
    End Try
End Sub

```

Testing your Custom Control in the Designer

At this point, you can build the `MarqueeControlLibrary` project. Test your implementation by creating a control that inherits from the `MarqueeControl` class and using it on a form.

To create a custom MarqueeControl implementation

1. Open `DemoMarqueeControl1` in the Windows Forms Designer. This creates an instance of the `DemoMarqueeControl1` type and displays it in an instance of the `MarqueeControlRootDesigner` type.

2. In the **Toolbox**, open the **MarqueeControlLibrary Components** tab. You will see the `MarqueeBorder` and `MarqueeText` controls available for selection.
3. Drag an instance of the `MarqueeBorder` control onto the `DemoMarqueeControl1` design surface. Dock this `MarqueeBorder` control to the parent control.
4. Drag an instance of the `MarqueeText` control onto the `DemoMarqueeControl1` design surface.
5. Build the solution.
6. Right-click the `DemoMarqueeControl1` and from the shortcut menu select the **Run Test** option to start the animation. Click **Stop Test** to stop the animation.
7. Open **Form1** in Design view.
8. Place two `Button` controls on the form. Name them `startButton` and `stopButton`, and change the `Text` property values to **Start** and **Stop**, respectively.
9. Implement `Click` event handlers for both `Button` controls.
10. In the **Toolbox**, open the **MarqueeControlTest Components** tab. You will see the `DemoMarqueeControl1` available for selection.
11. Drag an instance of `DemoMarqueeControl1` onto the **Form1** design surface.
12. In the `Click` event handlers, invoke the `Start` and `Stop` methods on the `DemoMarqueeControl1`.

```

Private Sub startButton_Click(sender As Object, e As System.EventArgs)
    Me.demoMarqueeControl1.Start()
End Sub 'startButton_Click

Private Sub stopButton_Click(sender As Object, e As System.EventArgs)
Me.demoMarqueeControl1.Stop()
End Sub 'stopButton_Click

```

```

private void startButton_Click(object sender, System.EventArgs e)
{
    this.demoMarqueeControl1.Start();
}

private void stopButton_Click(object sender, System.EventArgs e)
{
    this.demoMarqueeControl1.Stop();
}

```

1. Set the `MarqueeControlTest` project as the startup project and run it. You will see the form displaying your `DemoMarqueeControl1`. Click the **Start** button to start the animation. You should see the text flashing and the lights moving around the border.

Next Steps

The `MarqueeControlLibrary` demonstrates a simple implementation of custom controls and associated designers. You can make this sample more sophisticated in several ways:

- Change the property values for the `DemoMarqueeControl1` in the designer. Add more `MarqueBorder` controls and dock them within their parent instances to create a nested effect. Experiment with different settings for the `UpdatePeriod` and the light-related properties.
- Author your own implementations of `IMarqueeWidget`. You could, for example, create a flashing "neon sign"

or an animated sign with multiple images.

- Further customize the design-time experience. You could try shadowing more properties than [Enabled](#) and [Visible](#), and you could add new properties. Add new designer verbs to simplify common tasks like docking child controls.
- License the `MarqueeControl1`. For more information, see [How to: License Components and Controls](#).
- Control how your controls are serialized and how code is generated for them. For more information, see [Dynamic Source Code Generation and Compilation](#).

See Also

[UserControl](#)

[ParentControlDesigner](#)

[DocumentDesigner](#)

[IRootDesigner](#)

[DesignerVerb](#)

[UITypeEditor](#)

[BackgroundWorker](#)

[How to: Create a Windows Forms Control That Takes Advantage of Design-Time Features](#)

[Extending Design-Time Support](#)

[Custom Designers](#)

[.NET Shape Library: A Sample Designer](#)

How to: Author Controls for Windows Forms

5/4/2018 • 2 min to read • [Edit Online](#)

A control represents a graphical link between the user and the program. A control can provide or process data, accept user input, respond to events, or perform any number of other functions that connect the user and the application. Because a control is essentially a component with a graphical interface, it can serve any function that a component does, as well as provide user interaction. Controls are created to serve specific purposes, and authoring controls is just another programming task. With that in mind, the following steps represent an overview of the control authoring process. Links provide additional information on the individual steps.

NOTE

If you want to author a custom control to use on Web Forms, see [Developing Custom ASP.NET Server Controls](#).

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To author a control

1. Determine what you want your control to accomplish, or what part it will play in your application. Factors to consider are:
 - What kind of graphical interface do you need?
 - What specific user interactions will this control handle?
 - Is the functionality you need provided by any existing controls?
 - Can you get the functionality you need by combining several Windows Forms controls?
2. If you need an object model for your control, determine how functionality will be distributed throughout the object model, and divide up functionality between the control and any subobjects. An object model may be useful if you are planning a complex control, or want to incorporate several functionalities.
3. Determine the type of control (for example, user control, custom control, inherited Windows Forms control) you need. For details, see [Control Type Recommendations](#) and [Varieties of Custom Controls](#).
4. Express functionality as properties, methods, and events of the control and its subobjects or subsidiary structures, and assign appropriate access levels (for example, public, protected, and so on).
5. If you need custom painting for your control, add code for it. For details, see [Custom Control Painting and Rendering](#).
6. If your control inherits from [UserControl](#), you can test its runtime behavior by building the control project and running it in the **UserControl Test Container**. For more information, see [How to: Test the Run-Time Behavior of a UserControl](#).
7. You can also test and debug your control by creating a new project, such as a Windows Application, and placing it into a container. This process is demonstrated as part of [Walkthrough: Authoring a Composite Control with Visual Basic](#).
8. As you add each feature, add features to your test project to exercise the new functionality.
9. Repeat, refining the design.

10. Package and deploy your control. For details, see [Deploying Applications, Services, and Components](#).

See Also

[Walkthrough: Authoring a Composite Control with Visual Basic](#)

[Walkthrough: Inheriting from a Windows Forms Control with Visual Basic](#)

[How to: Inherit from the UserControl Class](#)

[How to: Inherit from the Control Class](#)

[How to: Inherit from Existing Windows Forms Controls](#)

[How to: Test the Run-Time Behavior of a UserControl](#)

[Varieties of Custom Controls](#)

How to: Author Composite Controls

5/4/2018 • 2 min to read • [Edit Online](#)

Composite controls can be employed in many ways. You can author them as part of a Windows desktop application project, and use them only on forms in the project. Or you can author them in a Windows Control Library project, compile the project into an assembly, and use the controls in other projects. You can even inherit from them, and use visual inheritance to customize them quickly for special purposes.

NOTE

If you want to author a composite control to use on Web Forms, see [Developing Custom ASP.NET Server Controls](#).

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To author a composite control

1. Open a new **Windows Application** project called `DemoControlHost`.
2. On the **Project** menu, click **Add User Control**.
3. In the **Add New Item** dialog box, give the class file (.vb or .cs file) the name you want the composite control to have.
4. Click the **Add** button to create the class file for the composite control.
5. Add controls from the **Toolbox** to the composite control surface.
6. Place code in event procedures, to handle events raised by the composite control or by its constituent controls.
7. Close the designer for the composite control, and save the file when you are prompted.
8. On the **Build** menu, click **Build Solution**.

The project must be built in order for custom controls to appear in the **Toolbox**.

9. Use the **DemoControlHost** tab of the **Toolbox** to add instances of your control to `Form1`.

To author a control class library

1. Open a new **Windows Control Library** project.

By default, the project contains a composite control.

2. Add controls and code as described in the procedure above.
3. Choose a control you do not want inheriting classes to change, and set the **Modifiers** property of this control to **Private**.
4. Build the DLL.

To inherit from a composite control in a control class library

1. On the **File** menu, point to **Add** and select **New Project** to add a new **Windows Application** project to the solution.

2. In **Solution Explorer**, right-click the **References** folder for the new project and choose **Add Reference** to open the **Add Reference** dialog box.
3. Select the **Projects** tab and double-click your control library project.
4. On the **Build** menu, click **Build Solution**.
5. In **Solution Explorer**, right-click your control library project and select **Add New Item** from the shortcut menu.
6. Select the **Inherited User Control** template from the **Add New Item** dialog box.
7. In the **Inheritance Picker** dialog box, double-click the control you want to inherit from.

A new control is added to your project.

8. Open the visual designer for the new control and add additional constituent controls.

You can see the constituent controls that were inherited from the composite control in your DLL, and you can alter the properties of controls whose **Modifiers** property is **Public**. You cannot change the properties of the control whose **Modifiers** property is **Private**.

See Also

[Walkthrough: Authoring a Composite Control with Visual Basic](#)

[Walkthrough: Authoring a Composite Control with Visual C#](#)

[Walkthrough: Inheriting from a Windows Forms Control with Visual Basic](#)

[Walkthrough: Inheriting from a Windows Forms Control with Visual C#](#)

[Control Type Recommendations](#)

[How to: Author Controls for Windows Forms](#)

[Varieties of Custom Controls](#)

How to: Inherit from the UserControl Class

5/4/2018 • 1 min to read • [Edit Online](#)

To combine the functionality of one or more Windows Forms controls with custom code, you can create a *user control*. User controls combine rapid control development, standard Windows Forms control functionality, and the versatility of custom properties and methods. When you begin creating a user control, you are presented with a visible designer, upon which you can place standard Windows Forms controls. These controls retain all of their inherent functionality, as well as the appearance and behavior (look and feel) of standard controls. Once these controls are built into the user control, however, they are no longer available to you through code. The user control does its own painting and also handles all of the basic functionality associated with controls.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To create a user control

1. Create a new **Windows Control Library** project.

A new project is created with a blank user control.

2. Drag controls from the **Windows Forms** tab of the **Toolbox** onto your designer.
3. These controls should be positioned and designed as you want them to appear in the final user control. If you want to allow developers to access the constituent controls, you must declare them as public, or selectively expose properties of the constituent control. For details, see [How to: Expose Properties of Constituent Controls](#).
4. Implement any custom methods or properties that your control will incorporate.
5. Press F5 to build the project and run your control in the **UserControl Test Container**. For more information, see [How to: Test the Run-Time Behavior of a UserControl](#).

See Also

[Varieties of Custom Controls](#)

[How to: Inherit from the Control Class](#)

[How to: Inherit from Existing Windows Forms Controls](#)

[How to: Author Controls for Windows Forms](#)

[Troubleshooting Inherited Event Handlers in Visual Basic](#)

[How to: Test the Run-Time Behavior of a UserControl](#)

How to: Inherit from Existing Windows Forms Controls

5/4/2018 • 2 min to read • [Edit Online](#)

If you want to extend the functionality of an existing control, you can create a control derived from an existing control through inheritance. When inheriting from an existing control, you inherit all of the functionality and visual properties of that control. For example, if you were creating a control that inherited from [Button](#), your new control would look and act exactly like a standard [Button](#) control. You could then extend or modify the functionality of your new control through the implementation of custom methods and properties. In some controls, you can also change the visual appearance of your inherited control by overriding its [OnPaint](#) method.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To create an inherited control

1. Create a new **Windows Forms Application** project.
2. From the **Project** menu, choose **Add New Item**.

The **Add New Item** dialog box appears.

3. In the **Add New Item** dialog box, double-click **Custom Control**.

A new custom control is added to your project.

4. If you are using Visual Basic, at the top of **Solution Explorer**, click **Show All Files**. Expand `CustomControl1.vb` and then open `CustomControl1.Designer.vb` in the Code Editor.
5. If you are using C#, open `CustomControl1.cs` in the Code Editor.
6. Locate the class declaration, which inherits from [Control](#).
7. Change the base class to the control that you want to inherit from.

For example, if you want to inherit from [Button](#), change the class declaration to the following:

```
Partial Class CustomControl1  
    Inherits System.Windows.Forms.Button
```

```
public partial class CustomControl1 : System.Windows.Forms.Button
```

8. If you are using Visual Basic, save and close `CustomControl1.Designer.vb`. Open `CustomControl1.vb` in the Code Editor.
9. Implement any custom methods or properties that your control will incorporate.
10. If you want to modify the graphical appearance of your control, override the [OnPaint](#) method.

NOTE

Overriding [OnPaint](#) will not allow you to modify the appearance of all controls. Those controls that have all of their painting done by Windows (for example, [TextBox](#)) never call their [OnPaint](#) method, and thus will never use the custom code. Refer to the Help documentation for the particular control you want to modify to see if the [OnPaint](#) method is available. For a list of all the Windows Form Controls, see [Controls to Use on Windows Forms](#). If a control does not have [OnPaint](#) listed as a member method, you cannot alter its appearance by overriding this method. For more information about custom painting, see [Custom Control Painting and Rendering](#).

```
Protected Overrides Sub OnPaint(ByVal e As _
    System.Windows.Forms.PaintEventArgs)
    MyBase.OnPaint(e)
    ' Insert code to do custom painting.
    ' If you want to completely change the appearance of your control,
    ' do not call MyBase.OnPaint(e).
End Sub
```

```
protected override void OnPaint(PaintEventArgs pe)
{
    base.OnPaint(pe);
    // Insert code to do custom painting.
    // If you want to completely change the appearance of your control,
    // do not call base.OnPaint(pe).
}
```

11. Save and test your control.

See Also

[Varieties of Custom Controls](#)

[How to: Inherit from the Control Class](#)

[How to: Inherit from the UserControl Class](#)

[How to: Author Controls for Windows Forms](#)

[Troubleshooting Inherited Event Handlers in Visual Basic](#)

[Walkthrough: Inheriting from a Windows Forms Control with Visual Basic](#)

[Walkthrough: Inheriting from a Windows Forms Control with Visual C#](#)

How to: Inherit from the Control Class

5/4/2018 • 1 min to read • [Edit Online](#)

If you want to create a completely custom control to use on a Windows Form, you should inherit from the [Control](#) class. While inheriting from the [Control](#) class requires that you perform more planning and implementation, it also provides you with the largest range of options. When inheriting from [Control](#), you inherit the very basic functionality that makes controls work. The functionality inherent in the [Control](#) class handles user input through the keyboard and mouse, defines the bounds and size of the control, provides a windows handle, and provides message handling and security. It does not incorporate any painting, which in this case is the actual rendering of the graphical interface of the control, nor does it incorporate any specific user interaction functionality. You must provide all of these aspects through custom code.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To create a custom control

1. Create a new **Windows Application** or **Windows Control Library** project.
2. From the **Project** menu, choose **Add Class**.
3. In the **Add New Item** dialog box, click **Custom Control**.

A new custom control is added to your project.

4. Press F7 to open the **Code Editor** for your custom control.
5. Locate the [OnPaint](#) method, which will be empty except for a call to the [OnPaint](#) method of the base class.
6. Modify the code to incorporate any custom painting you want for your control.

For information about writing code to render graphics for controls, see [Custom Control Painting and Rendering](#).

7. Implement any custom methods, properties, or events that your control will incorporate.
8. Save and test your control.

See Also

[Varieties of Custom Controls](#)

[How to: Inherit from the UserControl Class](#)

[How to: Inherit from Existing Windows Forms Controls](#)

[How to: Author Controls for Windows Forms](#)

[Troubleshooting Inherited Event Handlers in Visual Basic](#)

[Developing Windows Forms Controls at Design Time](#)

How to: Align a Control to the Edges of Forms at Design Time

5/4/2018 • 1 min to read • [Edit Online](#)

You can make your control align to the edge of your forms by setting the [Dock](#). This property designates where your control resides in the form. The [Dock](#) property can be set to the following values:

SETTING	EFFECT ON YOUR CONTROL
Bottom	Docks to the bottom of the form.
Fill	Fills all remaining space in the form.
Left	Docks to the left side of the form.
None	Does not dock anywhere, and it appears at the location specified by its Location .
Right	Docks to the right side of the form.
Top	Docks to the top of the form.

These values can also be set in code. For more information, see [How to: Align a Control to the Edges of Forms](#).

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

To set the Dock property for your control at design time

1. In the Windows Forms Designer, select your control.
2. In the **Properties** window, click the drop-down box next to the [Dock](#) property.
A graphical interface representing the six possible [Dock](#) settings is displayed.
3. Choose the appropriate setting.
4. Your control will now dock in the manner specified by the setting.

See Also

[Control.Dock](#)

[Control.Anchor](#)

[How to: Align a Control to the Edges of Forms](#)

[Walkthrough: Arranging Controls on Windows Forms Using Snaplines](#)

[How to: Anchor Controls on Windows Forms](#)

[How to: Anchor and Dock Child Controls in a TableLayoutPanel Control](#)

[How to: Anchor and Dock Child Controls in a FlowLayoutPanel Control](#)

[Walkthrough: Arranging Controls on Windows Forms Using a TableLayoutPanel](#)

[Walkthrough: Arranging Controls on Windows Forms Using a FlowLayoutPanel](#)

[Developing Windows Forms Controls at Design Time](#)

How to: Display a Control in the Choose Toolbox Items Dialog Box

5/4/2018 • 1 min to read • [Edit Online](#)

As you develop and distribute controls, you may want those controls to appear in the **Choose Toolbox Items** dialog box, which is displayed when you right-click the **Toolbox** and select **Choose Items**. You can enable your control to appear in this dialog box by using the AssemblyFoldersEx registration procedure.

To display your control in the Choose Toolbox Items dialog box

- Install your control assembly to the global assembly cache. For more information, see [How to: Install an Assembly into the Global Assembly Cache](#)

-or-
- Register your control and its associated design-time assemblies by using the AssemblyFoldersEx registration procedure. AssemblyFoldersEx is a registry location where third-party vendors store paths for each version of the framework that they support. Design-time resolution can look in this registry location to find reference assemblies. The registry script can specify the controls you want to appear in the Toolbox. For more information, see [Deploying a Custom Control and Design-time Assemblies \(Visual Studio 2013\)](#).

See Also

[Choose Toolbox Items Dialog Box \(Visual Studio\)](#)

[Deploying a Custom Control and Design-time Assemblies \(Visual Studio 2013\)](#)

[Developing Windows Forms Controls at Design Time](#)

[How to: Install an Assembly into the Global Assembly Cache](#)

[Walkthrough: Automatically Populating the Toolbox with Custom Components](#)

How to: Provide a Toolbox Bitmap for a Control

5/4/2018 • 1 min to read • [Edit Online](#)

If you want to have a special icon for your control appear in the **Toolbox**, you can specify a particular image by using the [ToolboxBitmapAttribute](#). This class is an *attribute*, a special kind of class you can attach to other classes. For more information about attributes, see [NOT IN BUILD: Attributes Overview in Visual Basic](#) for Visual Basic and [Attributes](#) for Visual C#.

Using the [ToolboxBitmapAttribute](#), you can specify a string that indicates the path and file name for a 16 by 16 pixel bitmap. This bitmap then appears next to your control when added to the **Toolbox**. You can also specify a [Type](#), in which case the bitmap associated with that type is loaded. If you specify both a [Type](#) and a string, the control searches for an image resource with the name specified by the string parameter in the assembly containing the type specified by the [Type](#) parameter.

To specify a Toolbox bitmap for your control

1. Add the [ToolboxBitmapAttribute](#) to the class declaration of your control before the `class` keyword for visual Basic, and above the class declaration for Visual C#.

```
' Specifies the bitmap associated with the Button type.  
<ToolboxBitmap(GetType(Button))> Class MyControl1  
' Specifies a bitmap file.  
End Class  
<ToolboxBitmap("C:\Documents and Settings\Joe\MyPics\myImage.bmp")> _  
    Class MyControl2  
End Class  
' Specifies a type that indicates the assembly to search, and the name  
' of an image resource to look for.  
<ToolboxBitmap(GetType(MyControl), "MyControlBitmap")> Class MyControl  
End Class
```

```
// Specifies the bitmap associated with the Button type.  
[ToolboxBitmap(typeof(Button))]  
class MyControl1 : UserControl  
{  
}  
// Specifies a bitmap file.  
[ToolboxBitmap(@"C:\Documents and Settings\Joe\MyPics\myImage.bmp")]  
class MyControl2 : UserControl  
{  
}  
// Specifies a type that indicates the assembly to search, and the name  
// of an image resource to look for.  
[ToolboxBitmap(typeof(MyControl), "MyControlBitmap")]  
class MyControl : UserControl  
{  
}
```

2. Rebuild the project.

NOTE

The bitmap does not appear in the Toolbox for autogenerated controls and components. To see the bitmap, reload the control by using the **Choose Toolbox Items** dialog box. For more information, see [Walkthrough: Automatically Populating the Toolbox with Custom Components](#).

See Also

[ToolboxBitmapAttribute](#)

[Walkthrough: Automatically Populating the Toolbox with Custom Components](#)

[Developing Windows Forms Controls at Design Time](#)

[Attributes \(Visual Basic\)](#)

[Attributes](#)

How to: Test the Run-Time Behavior of a UserControl

5/4/2018 • 2 min to read • [Edit Online](#)

When you develop a [UserControl](#), you need to test its run-time behavior. You can create a separate Windows-based application project and place your control on a test form, but this procedure is inconvenient. A faster and easier way is to use the **UserControl Test Container** provided by Visual Studio. This test container starts directly from your Windows control library project.

IMPORTANT

For the test container to load your [UserControl](#), the control must have at least one public constructor.

NOTE

The dialog boxes and menu commands you see might differ from those described in Help depending on your active settings or edition. To change your settings, choose **Import and Export Settings** on the **Tools** menu. For more information, see [Customizing Development Settings in Visual Studio](#).

NOTE

A Visual C++ control cannot be tested using the **UserControl Test Container**.

To test the run-time behavior of a UserControl

1. Create a Windows control library project called **TestContainerExample**. For details, see [Windows Control Library Template](#).
2. In the **Windows Forms Designer**, drag a [Label](#) control from the **Toolbox** onto the control's design surface.
3. Press F5 to build the project and run the **UserControl Test Container**. The test container appears with your [UserControl](#) in the **Preview** pane.
4. Select the [BackColor](#) property displayed in the [PropertyGrid](#) control to the right of the **Preview** pane. Change its value to `controlDark`. Observe that the control changes to a darker color. Try changing other property values and observe the effect on your control.
5. Click the **Dock Fill User Control** check box below the **Preview** pane. Observe that the control is resized to fill the pane. Resize the test container and observe that the control is resized with the pane.
6. Close the test container.
7. Add another user control to the **TestContainerExample** project. For details, see [NIB:How to: Add Existing Items to a Project](#).
8. In the **Windows Forms Designer**, drag a [Button](#) control from the **Toolbox** onto the control's design surface.
9. Press F5 to build the project and run the test container.

10. Click the **Select User Control** ComboBox to switch between the two user controls.

Testing User Controls from Another Project

You can test user controls from other projects in your current project's test container.

To test user controls from another project

1. Create a Windows control library project called **TestContainerExample2**. For details, see [Windows Control Library Template](#).
2. In the **Windows Forms Designer**, drag a [RadioButton](#) control from the **Toolbox** onto the control's design surface.
3. Press F5 to build the project and run the test container. The test container appears with your [UserControl](#) in the **Preview** pane.
4. Click the **Load** button.
5. In the **Open** dialog box, navigate to **TestContainerExample.dll**, which you built in the previous procedure. Select **TestContainerExample.dll** and click the **Open** button to load the user controls
6. Use the **Select User Control**ComboBox to switch between the two user controls from the **TestContainerExample** project.

See Also

[UserControl](#)

[How to: Author Composite Controls](#)

[Walkthrough: Authoring a Composite Control with Visual Basic](#)

[Walkthrough: Authoring a Composite Control with Visual C#](#)

[User Control Designer](#)

Design-Time Errors in the Windows Forms Designer

5/4/2018 • 2 min to read • [Edit Online](#)

This topic explains the meaning and use of the design-time error list that appears in Microsoft Visual Studio when the Windows Forms Designer fails to load. If this error list appears, you should not interpret it as a bug in the designer, but as an aid to correcting errors in your code.

A basic understanding of this error list will help you debug your applications by providing detailed information about the errors and suggesting possible solutions.

The Design-Time Error List Interface

If the Windows Forms Designer fails to load, an error list will appear in the designer. The errors are grouped into categories. For example, if you have four instances of undeclared variables, these will be grouped into the same error category. Each error category includes a brief description that summarizes the error.

You can expand or collapse an error category by either clicking on the error category heading or by clicking the expand/collapse chevron. When you expand an error category, the following additional help is displayed:

- Instances of this error.
- Help with this error.
- Forum posts about this error.

Instances of This Error

The additional help list all instances of the error in your current project. Many errors include an exact location in the following format: *[Project Name] [Form Name] Line:[Line Number] Column:[Column Number]*. The **Go to code** link takes you to the location in your code where the error occurs.

If a call stack is associated with the error, you can click the **Show Call Stack** link, which further expands the error to show the call stack. Examining the stack can provide valuable debugging information. For example, you can track the functions that were called before the error occurred. The call stack is selectable so that you can copy and save it.

NOTE

In Visual Basic, the design-time error list does not display more than one error, but it may display multiple instances of the same error. In Visual C++, the errors do not have goto code links/line number links.

Help with This Error

If the error contains a link to an associated MSDN help topic, the additional help will include a link to the help topic. When you click the link, the associated help topic appears in Visual Studio.

Forum posts about this error

The additional help will include a link to MSDN forum posts related to the error. The forums are searched based on the string of the error message. You can also try searching the following forums:

- [Windows Forms Designer Forum](#)
- [Windows Forms Forums](#)

Ignore and Continue

You can choose to ignore the error condition and continue loading the designer. Choosing this action may result in unexpected behavior. For example, controls may not appear on the design surface.

See Also

- [Troubleshooting Design-Time Development](#)
- [Troubleshooting Control and Component Authoring](#)
- [Developing Windows Forms Controls at Design Time](#)
- [Windows Forms Designer Error Messages](#)

Troubleshooting Control and Component Authoring

5/4/2018 • 3 min to read • [Edit Online](#)

This topic lists the following common problems that arise when developing components and controls. For more information, see [Programming with Components](#).

- Cannot Add Control to Toolbox
- Cannot Debug the Windows Forms User Control or Component
- Event Is Raised Twice in Inherited Control or Component
- Design-Time Error: "Failed to Create Component '*Component Name*'"
- STAThreadAttribute
- Component Icon Does Not Appear in Toolbox

Cannot Add Control to Toolbox

If you want to add a custom control that you created in another project or a third-party control to the **Toolbox**, you must do so manually. If the current project contains your control or component, it should appear in the **Toolbox** automatically. For more information, see [Walkthrough: Automatically Populating the Toolbox with Custom Components](#).

To add a control to the Toolbox

1. Right-click the **Toolbox** and from the shortcut menu, select **Choose Items**.

2. In the **Choose Toolbox Items** dialog box, add the component:

- If you want to add a .NET Framework component or control, click the **.NET Framework Components** tab.
– or –
- If you want to add a COM component or ActiveX control, click the **COM Components** tab.

3. If your control is listed in the dialog box, confirm it is selected, and then click **OK**.

The control is added to the **Toolbox**.

4. If your control is not listed in the dialog box, do the following:

- a. Click the **Browse** button.
- b. Browse to the folder that contains the .dll file that contains your control.
- c. Select the .dll file and click **Open**.

Your control appears in the dialog box.

d. Confirm that your control is selected, and then click **OK**.

Your control is added to the **Toolbox**.

Cannot Debug the Windows Forms User Control or Component

If your control derives from the [UserControl](#) class, you can debug its run-time behavior with the test container. For

more information, see [How to: Test the Run-Time Behavior of a UserControl](#).

Other custom controls and components are not stand-alone projects. They must be hosted by an application such as a Windows Forms project. To debug a control or component, you must add it to a Windows Forms project.

To debug a control or component

1. From the **Build** menu, click **Build Solution** to build your solution.
2. From the **File** menu, choose **Add**, and then **New Project** to add a test project to your application.
3. In the **Add New Project** dialog box choose **Windows Application** for the type of project.
4. In **Solution Explorer**, right-click the **References** node for the new project. On the shortcut menu, click **Add Reference** to add a reference to the project containing the control or component.
5. Create an instance of your control or component in the test project. If your component is in the **Toolbox**, you can drag it to your designer surface, or you can create the instance programmatically, as shown in the following code example.

```
Dim Component1 As New MyNeatComponent()
```

```
MyNeatComponent Component1 = new MyNeatComponent();
```

You can now debug your control or component as usual.

For more information about debugging, see [Debugging in Visual Studio](#) and [Walkthrough: Debugging Custom Windows Forms Controls at Design Time](#).

Event Is Raised Twice in Inherited Control or Component

This is likely due to a duplicated `Handles` clause. For more information, see [Troubleshooting Inherited Event Handlers in Visual Basic](#).

Design-Time Error: "Failed to Create Component 'Component Name'"

Your component or control must provide a default constructor with no parameters. When the design environment creates an instance of your component or control, it does not attempt to provide any parameters to constructor overloads that take parameters.

STAThreadAttribute

The `STAThreadAttribute` informs the common language runtime (CLR) that Windows Forms uses the single-threaded apartment model. You may notice unintended behavior if you do not apply this attribute to your Windows Forms application's `Main` method. For example, background images may not appear for controls like `ListView`. Some controls may also require this attribute for correct AutoComplete and drag-and-drop behavior.

Component Icon Does Not Appear in Toolbox

When you use `ToolboxBitmapAttribute` to associate an icon with your custom component, the bitmap does not appear in the Toolbox for autogenerated components. To see the bitmap, reload the control by using the **Choose Toolbox Items** dialog box. For more information, see [How to: Provide a Toolbox Bitmap for a Control](#).

See Also

[Developing Windows Forms Controls at Design Time](#)

[Walkthrough: Automatically Populating the Toolbox with Custom Components](#)

[How to: Test the Run-Time Behavior of a UserControl](#)

[Walkthrough: Debugging Custom Windows Forms Controls at Design Time](#)

[Component Authoring](#)

[Troubleshooting Design-Time Development](#)

[Programming with Components](#)