

# Contents

## Controls

[Controls by Category](#)

[WPF Content Model](#)

[Control Library](#)

[Border](#)

[Animate a BorderThickness Value](#)

[BulletDecorator](#)

[Button](#)

[Create a Button That Has an Image](#)

[Calendar](#)

[Canvas](#)

[How-to Topics](#)

[Wrap a Border Around the Content of a Canvas](#)

[Get or Set Canvas Positioning Properties](#)

[Create and Use a Canvas](#)

[Use the Attached Properties of Canvas to Position Child Elements](#)

[Resize a Canvas by Using a Thumb](#)

[CheckBox](#)

[ComboBox](#)

[ContextMenu](#)

[ContextMenu Overview](#)

[DataGrid](#)

[Default Keyboard and Mouse Behavior in the DataGrid Control](#)

[How to: Add Row Details to a DataGrid Control](#)

[How to: Group, Sort, and Filter Data in the DataGrid Control](#)

[How to: Implement Validation with the DataGrid Control](#)

[Walkthrough: Display Data from a SQL Server Database in a DataGrid Control](#)

[Sizing Options in the DataGrid Control](#)

[DatePicker](#)

## DockPanel

### How-to Topics

[Get or Set a Dock Value](#)

[Create a DockPanel](#)

[Partition Space by Using the DockPanel Element](#)

## DocumentViewer

### Expander

#### Expander Overview

### How-to Topics

[Create an Expander with a ScrollViewer](#)

## FlowDocumentPageViewer

## FlowDocumentReader

## FlowDocumentScrollView

## Frame

## Grid

### How-to Topics

[Build a Standard UI Dialog Box by Using Grid](#)

[Create a Complex Grid](#)

[Create a Grid Element](#)

[Create and Use a GridLengthConverter Object](#)

[Manipulate Columns and Rows by Using ColumnDefinitionsCollections and RowDefinitionsCollections](#)

[Position the Child Elements of a Grid](#)

[Share Sizing Properties Between Grids](#)

## GridSplitter

### How-to Topics

[Resize Rows with a GridSplitter](#)

[Resize Columns with a GridSplitter](#)

[Make Sure That a GridSplitter Is Visible](#)

## GroupBox

### Define a GroupBox Template

## Image

### How-to Topics

[Use the Image Element](#)

[Convert an Image to Greyscale](#)

[Crop an Image](#)

[Rotate an Image](#)

## [Label](#)

[Create a Control That Has an Access Key and Text Wrapping](#)

## [ListBox](#)

[How-to Topics](#)

[Bind a ListBox to Data](#)

[Get a ListItem](#)

[Improve the Scrolling Performance of a ListBox](#)

## [ListView](#)

[Overviews](#)

[ListView Overview](#)

[GridView Overview](#)

[GridView Column Header Styles and Templates Overview](#)

[How-to Topics](#)

[Sort a GridView Column When a Header Is Clicked](#)

[Create a Custom View Mode for a ListView](#)

[Use Templates to Style a ListView That Uses GridView](#)

[Create a Style for a Dragged GridView Column Header](#)

[Display ListView Contents by Using a GridView](#)

[Use Triggers to Style Selected Items in a ListView](#)

[Create ListViewItems with a CheckBox](#)

[Display Data by Using GridViewRowPresenter](#)

[Group Items in a ListView That Implements a GridView](#)

[Style a Row in a ListView That Implements a GridView](#)

[Change the Horizontal Alignment of a Column in a ListView](#)

[Handle the MouseDoubleClick Event for Each Item in a ListView](#)

## [Menu](#)

[Menu Overview](#)

## [Panel](#)

## Panels Overview

### How-to Topics

[Create a Custom Panel Element](#)

[Override the Panel OnRender Method](#)

[Set the Height Properties of an Element](#)

[Set the Width Properties of an Element](#)

## [PasswordBox](#)

### [Popup](#)

[Popup Overview](#)

[Popup Placement Behavior](#)

### How-to Topics

[Animate a Popup](#)

[Specify a Custom Popup Position](#)

## [ProgressBar](#)

## [PrintDialog](#)

## [RadioButton](#)

## [RepeatButton](#)

## [RichTextBox](#)

[RichTextBox Overview](#)

### How-to Topics

[Extract the Text Content from a RichTextBox](#)

[Change Selection in a RichTextBox Programmatically](#)

[Save, Load, and Print RichTextBox Content](#)

[Position a Custom Context Menu in a RichTextBox](#)

## [ScrollBar](#)

[Customize the Thumb Size on a ScrollBar](#)

## [ScrollViewer](#)

[ScrollViewer Overview](#)

### How-to Topics

[Handle the ScrollChanged Event](#)

[Scroll Content by Using the IScrollInfo Interface](#)

[Use the Content-Scrolling Methods of ScrollViewer](#)

[Separator](#)

[Slider](#)

[Customize the Ticks on a Slider](#)

[StackPanel](#)

[How-to Topics](#)

[Choose Between StackPanel and DockPanel](#)

[Create a StackPanel](#)

[Horizontally or Vertically Align Content in a StackPanel](#)

[StatusBar](#)

[TabControl](#)

[TextBlock](#)

[TextBlock Overview](#)

[TextBox](#)

[TextBox Overview](#)

[How-to Topics](#)

[Create a Multiline TextBox Control](#)

[Detect When Text in a TextBox Has Changed](#)

[Enable Tab Characters in a TextBox Control](#)

[Get a Collection of Lines from a TextBox](#)

[Make a TextBox Control Read-Only](#)

[Position the Cursor at the Beginning or End of Text in a TextBox Control](#)

[Retrieve a Text Selection](#)

[Set Focus in a TextBox Control](#)

[Set the Text Content of a TextBox Control](#)

[Enable Spell Checking in a Text Editing Control](#)

[Use a Custom Context Menu with a TextBox](#)

[Use Spell Checking with a Context Menu](#)

[Add a Watermark to a TextBox](#)

[ToolBar](#)

[ToolBar Overview](#)

[Style Controls on a ToolBar](#)

[ToolTip](#)

[ToolTip Overview](#)

[How-to Topics](#)

[Position a ToolTip](#)

[Use the BetweenShowDelay Property](#)

[TreeView](#)

[TreeView Overview](#)

[How-to Topics](#)

[Create Simple or Complex TreeViews](#)

[Use SelectedValue, SelectedValuePath, and SelectedItem](#)

[Bind a TreeView to Data That Has an Indeterminable Depth](#)

[Improve the Performance of a TreeView](#)

[Find a TreeViewItem in a TreeView](#)

[WrapPanel](#)

[Viewbox](#)

[Apply Stretch Properties to the Contents of a Viewbox](#)

[Styles and Templates](#)

[Styling and Templating](#)

[Customizing an Existing Control with a ControlTemplate](#)

[How to: Find ControlTemplate-Generated Elements](#)

[Control Customization](#)

[Control Authoring Overview](#)

[Creating a Control That Has a Customizable Appearance](#)

[Guidelines for Designing Stylistic Controls](#)

[Adorners](#)

[Adorners Overview](#)

[How-to Topics](#)

[Implement an Adorner](#)

[Bind an Adorner to an Element](#)

[Adorn the Children of a Panel](#)

[Remove an Adorner from an Element](#)

[Remove all Adorners from an Element](#)

[Control Styles and Templates](#)

- [Button Styles and Templates](#)
- [Calendar Styles and Templates](#)
- [CheckBox Styles and Templates](#)
- [ComboBox Styles and Templates](#)
- [ContextMenu Styles and Templates](#)
- [DataGrid Styles and Templates](#)
- [DatePicker Styles and Templates](#)
- [DocumentViewer Styles and Templates](#)
- [Expander Styles and Templates](#)
- [Frame Styles and Templates](#)
- [GroupBox Styles and Templates](#)
- [Label Styles and Templates](#)
- [ListBox Styles and Templates](#)
- [ListView Styles and Templates](#)
- [Menu Styles and Templates](#)
- [NavigationWindow Styles and Templates](#)
- [PasswordBox Styles and Templates](#)
- [ProgressBar Styles and Templates](#)
- [RadioButton Styles and Templates](#)
- [RepeatButton Styles and Templates](#)
- [ScrollBar Styles and Templates](#)
- [ScrollViewer Styles and Templates](#)
- [Slider Styles and Templates](#)
- [StatusBar Styles and Templates](#)
- [TabControl Styles and Templates](#)
- [TextBox Styles and Templates](#)
- [Thumb Styles and Templates](#)
- [ToggleButton Styles and Templates](#)
- [ToolBar Styles and Templates](#)
- [ToolTip Styles and Templates](#)
- [TreeView Styles and Templates](#)
- [Window Styles and Templates](#)

[UI Automation of a WPF Custom Control](#)

[Walkthroughs: Create a Custom Animated Button](#)

[Create a Button by Using Blend](#)

[Create a Button by Using XAML](#)

# Controls

7 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) ships with many of the common UI components that are used in almost every Windows application, such as [Button](#), [Label](#), [TextBox](#), [Menu](#), and [ListBox](#). Historically, these objects have been referred to as controls. While the WPF SDK continues to use the term "control" to loosely mean any class that represents a visible object in an application, it is important to note that a class does not need to inherit from the [Control](#) class to have a visible presence. Classes that inherit from the [Control](#) class contain a [ControlTemplate](#), which allows the consumer of a control to radically change the control's appearance without having to create a new subclass. This topic discusses how controls (both those that do inherit from the [Control](#) class and those that do not) are commonly used in WPF.

## Creating an Instance of a Control

You can add a control to an application by using either Extensible Application Markup Language (XAML) or code. The following example shows how to create a simple application that asks a user for their first and last name. This example creates six controls: two labels, two text boxes, and two buttons, in XAML. All controls can be created similarly.

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="30"/>
        <RowDefinition Height="30"/>
        <RowDefinition Height="30"/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>

    <Label>
        Enter your first name:
    </Label>
    <TextBox Grid.Row="0" Grid.Column="1"
            Name="firstName" Margin="0,5,10,5"/>

    <Label Grid.Row="1" >
        Enter your last name:
    </Label>
    <TextBox Grid.Row="1" Grid.Column="1"
            Name="lastName" Margin="0,5,10,5"/>

    <Button Grid.Row="2" Grid.Column="0"
            Name="submit" Margin="2">
        View message
    </Button>

    <Button Grid.Row="2" Grid.Column="1"
            Name="Clear" Margin="2">
        Clear Name
    </Button>
</Grid>
```

The following example creates the same application in code. For brevity, the creation of the [Grid](#), `grid1`, has been excluded from the sample. `grid1` has the same column and row definitions as shown in the preceding XAML

example.

```
Label firstNameLabel;
Label lastNameLabel;
TextBox firstName;
TextBox lastName;
Button submit;
Button clear;

void CreateControls()
{
    firstNameLabel = new Label();
    firstNameLabel.Content = "Enter your first name:";
    grid1.Children.Add(firstNameLabel);

    firstName = new TextBox();
    firstName.Margin = new Thickness(0, 5, 10, 5);
    Grid.SetColumn(firstName, 1);
    grid1.Children.Add(firstName);

    lastNameLabel = new Label();
    lastNameLabel.Content = "Enter your last name:";
    Grid.SetRow(lastNameLabel, 1);
    grid1.Children.Add(lastNameLabel);

    lastName = new TextBox();
    lastName.Margin = new Thickness(0, 5, 10, 5);
    Grid.SetColumn(lastName, 1);
    Grid.SetRow(lastName, 1);
    grid1.Children.Add(lastName);

    submit = new Button();
    submit.Content = "View message";
    Grid.SetRow(submit, 2);
    grid1.Children.Add(submit);

    clear = new Button();
    clear.Content = "Clear Name";
    Grid.SetRow(clear, 2);
    Grid.SetColumn(clear, 1);
    grid1.Children.Add(clear);
}
```

```

Private firstNameLabel As Label
Private lastNameLabel As Label
Private firstName As TextBox
Private lastName As TextBox
Private submit As Button
Private clear As Button

Sub CreateControls()
    firstNameLabel = New Label()
    firstNameLabel.Content = "Enter your first name:"
    grid1.Children.Add(firstNameLabel)

    firstName = New TextBox()
    firstName.Margin = New Thickness(0, 5, 10, 5)
    Grid.SetColumn(firstName, 1)
    grid1.Children.Add(firstName)

    lastNameLabel = New Label()
    lastNameLabel.Content = "Enter your last name:"
    Grid.SetRow(lastNameLabel, 1)
    grid1.Children.Add(lastNameLabel)

    lastName = New TextBox()
    lastName.Margin = New Thickness(0, 5, 10, 5)
    Grid.SetColumn(lastName, 1)
    Grid.SetRow(lastName, 1)
    grid1.Children.Add(lastName)

    submit = New Button()
    submit.Content = "View message"
    Grid.SetRow(submit, 2)
    grid1.Children.Add(submit)

    clear = New Button()
    clear.Content = "Clear Name"
    Grid.SetRow(clear, 2)
    Grid.SetColumn(clear, 1)
    grid1.Children.Add(clear)

```

End Sub

## Changing the Appearance of a Control

It is common to change the appearance of a control to fit the look and feel of your application. You can change the appearance of a control by doing one of the following, depending on what you want to accomplish:

- Change the value of a property of the control.
- Create a [Style](#) for the control.
- Create a new [ControlTemplate](#) for the control.

### Changing a Control's Property Value

Many controls have properties that allow you to change how the control appears, such as the [Background](#) of a [Button](#). You can set the value properties in both XAML and code. The following example sets the [Background](#), [FontSize](#), and [FontWeight](#) properties on a [Button](#) in XAML.

```

<Button FontSize="14" FontWeight="Bold">
    <!--Set the Background property of the Button to
        a LinearGradientBrush.-->
    <Button.Background>
        <LinearGradientBrush StartPoint="0,0.5"
            EndPoint="1,0.5">
            <GradientStop Color="Green" Offset="0.0" />
            <GradientStop Color="White" Offset="0.9" />
        </LinearGradientBrush>

    </Button.Background>
    View message
</Button>

```

The following example sets the same properties in code.

```

LinearGradientBrush buttonBrush = new LinearGradientBrush();
buttonBrush.StartPoint = new Point(0, 0.5);
buttonBrush.EndPoint = new Point(1, 0.5);
buttonBrush.GradientStops.Add(new GradientStop(Colors.Green, 0));
buttonBrush.GradientStops.Add(new GradientStop(Colors.White, 0.9));

submit.Background = buttonBrush;
submit.FontSize = 14;
submit.FontWeight = FontWeights.Bold;

```

```

Dim buttonBrush As New LinearGradientBrush()
buttonBrush.StartPoint = New Point(0, 0.5)
buttonBrush.EndPoint = New Point(1, 0.5)
buttonBrush.GradientStops.Add(New GradientStop(Colors.Green, 0))
buttonBrush.GradientStops.Add(New GradientStop(Colors.White, 0.9))

submit.Background = buttonBrush
submit.FontSize = 14
submit.FontWeight = FontWeights.Bold

```

## Creating a Style for a Control

WPF gives you the ability to specify the appearance of controls wholesale, instead of setting properties on each instance in the application, by creating a [Style](#). The following example creates a [Style](#) that is applied to each [Button](#) in the application. [Style](#) definitions are typically defined in XAML in a [ResourceDictionary](#), such as the [Resources](#) property of the [FrameworkElement](#).

```

<Style TargetType="Button">
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="FontWeight" Value="Bold"/>
    <Setter Property="Background">
        <Setter.Value>
            <LinearGradientBrush StartPoint="0,0.5"
                EndPoint="1,0.5">
                <GradientStop Color="Green" Offset="0.0" />
                <GradientStop Color="White" Offset="0.9" />
            </LinearGradientBrush>

        </Setter.Value>
    </Setter>
</Style>

```

You can also apply a style to only certain controls of a specific type by assigning a key to the style and specifying that key in the [Style](#) property of your control. For more information about styles, see [Styling and Templating](#).

## Creating a ControlTemplate

A [Style](#) allows you to set properties on multiple controls at a time, but sometimes you might want to customize the appearance of a [Control](#) beyond what you can do by creating a [Style](#). Classes that inherit from the [Control](#) class have a [ControlTemplate](#), which defines the structure and appearance of a [Control](#). The [Template](#) property of a [Control](#) is public, so you can give a [Control](#) a [ControlTemplate](#) that is different than its default. You can often specify a new [ControlTemplate](#) for a [Control](#) instead of inheriting from a control to customize the appearance of a [Control](#).

Consider the very common control, [Button](#). The primary behavior of a [Button](#) is to enable an application to take some action when the user clicks it. By default, the [Button](#) in WPF appears as a raised rectangle. While developing an application, you might want to take advantage of the behavior of a [Button](#)--that is, by handling the button's click event--but you might change the button's appearance beyond what you can do by changing the button's properties. In this case, you can create a new [ControlTemplate](#).

The following example creates a [ControlTemplate](#) for a [Button](#). The [ControlTemplate](#) creates a [Button](#) with rounded corners and a gradient background. The [ControlTemplate](#) contains a [Border](#) whose [Background](#) is a [LinearGradientBrush](#) with two [GradientStop](#) objects. The first [GradientStop](#) uses data binding to bind the [Color](#) property of the [GradientStop](#) to the color of the button's background. When you set the [Background](#) property of the [Button](#), the color of that value will be used as the first [GradientStop](#). For more information about data binding, see [Data Binding Overview](#). The example also creates a [Trigger](#) that changes the appearance of the [Button](#) when [IsPressed](#) is `true`.

```

<!--Define a template that creates a gradient-colored button.-->
<Style TargetType="Button">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="Button">
                <Border
                    x:Name="Border"
                    CornerRadius="20"
                    BorderThickness="1"
                    BorderBrush="Black">
                    <Border.Background>
                        <LinearGradientBrush StartPoint="0,0.5"
                            EndPoint="1,0.5">
                            <GradientStop Color="{Binding Background.Color,
                                RelativeSource={RelativeSource TemplatedParent}}"
                                Offset="0.0" />
                            <GradientStop Color="White" Offset="0.9" />
                        </LinearGradientBrush>
                    </Border.Background>
                    <ContentPresenter
                        Margin="2"
                        HorizontalAlignment="Center"
                        VerticalAlignment="Center"
                        RecognizesAccessKey="True"/>
                </Border>
                <ControlTemplate.Triggers>
                    <!--Change the appearance of
                    the button when the user clicks it.-->
                    <Trigger Property="IsPressed" Value="true">
                        <Setter TargetName="Border" Property="Background">
                            <Setter.Value>
                                <LinearGradientBrush StartPoint="0,0.5"
                                    EndPoint="1,0.5">
                                    <GradientStop Color="{Binding Background.Color,
                                        RelativeSource={RelativeSource TemplatedParent}}"
                                        Offset="0.0" />
                                    <GradientStop Color="DarkSlateGray" Offset="0.9" />
                                </LinearGradientBrush>
                            </Setter.Value>
                        </Setter>
                    </Trigger>
                </ControlTemplate.Triggers>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

```

<Button Grid.Row="2" Grid.ColumnSpan="2" Name="submitName"
    Background="Green">View message</Button>

```

#### NOTE

The [Background](#) property of the [Button](#) must be set to a [SolidColorBrush](#) for the example to work properly.

## Subscribing to Events

You can subscribe to a control's event by using either XAML or code, but you can only handle an event in code. The following example shows how to subscribe to the `click` event of a [Button](#).

```
<Button Grid.Row="2" Grid.ColumnSpan="2" Name="submitName" Click="submit_Click"
    Background="Green">View message</Button>
```

```
submit.Click += new RoutedEventHandler(submit_Click);
```

```
AddHandler submit.Click, AddressOf submit_Click
```

The following example handles the `click` event of a [Button](#).

```
void submit_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Hello, " + firstName.Text + " " + lastName.Text);
}
```

```
Private Sub submit_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    MessageBox.Show("Hello, " + firstName.Text + " " + lastName.Text)

End Sub
```

## Rich Content in Controls

Most classes that inherit from the [Control](#) class have the capacity to contain rich content. For example, a [Label](#) can contain any object, such as a string, an [Image](#), or a [Panel](#). The following classes provide support for rich content and act as base classes for most of the controls in WPF.

- [ContentControl](#)-- Some examples of classes that inherit from this class are [Label](#), [Button](#), and [ToolTip](#).
- [ItemsControl](#)-- Some examples of classes that inherit from this class are [ListBox](#), [Menu](#), and [StatusBar](#).
- [HeaderedContentControl](#)-- Some examples of classes that inherit from this class are [TabItem](#), [GroupBox](#), and [Expander](#).
- [HeaderedItemsControl](#)--Some examples of classes that inherit from this class are [MenuItem](#), [TreeViewItem](#), and [ToolBar](#).

For more information about these base classes, see [WPF Content Model](#).

## See also

- [Styling and Templating](#)
- [Controls by Category](#)
- [Control Library](#)
- [Data Templating Overview](#)
- [Data Binding Overview](#)
- [Input](#)
- [Enable a Command](#)
- [Walkthroughs: Create a Custom Animated Button](#)
- [Control Customization](#)

# Controls by Category

2 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) controls can be logically grouped into several categories. These categories can be used to select the appropriate control for your scenario by helping you see which controls have similar usage patterns or functionality.

## Layout

Layout controls are used to manage the size, dimensions, position, and arrangement of child elements.

- [Border](#)
- [BulletDecorator](#)
- [Canvas](#)
- [DockPanel](#)
- [Expander](#)
- [Grid](#)
- [GridSplitter](#)
- [GroupBox](#)
- [Panel](#)
- [ResizeGrip](#)
- [Separator](#)
- [ScrollBar](#)
- [ScrollViewer](#)
- [StackPanel](#)
- [Thumb](#)
- [Viewbox](#)
- [VirtualizingStackPanel](#)
- [Window](#)
- [WrapPanel](#)

## Buttons

Buttons are one of the most basic user interface controls. Applications typically perform some task in the [Click](#) event when a user clicks on them.

- [Button](#)
- [RepeatButton](#)

## Data Display

Data display controls are used to show information from a data source.

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

## Date Display and Selection

Date controls are used to display and select calendar information.

- [Calendar](#)
- [DatePicker](#)

## Menus

Menus are used to group related actions or to provide contextual assistance.

- [ContextMenu](#)
- [Menu](#)
- [ToolBar](#)

## Selection

Selection controls are used to enable a user to select one or more options.

- [CheckBox](#)
- [ComboBox](#)
- [ListBox](#)
- [RadioButton](#)
- [Slider](#)

## Navigation

Navigation controls enhance or extend the application navigation experience by creating targeting frames or tabbed application appearance.

- [Frame](#)
- [Hyperlink](#)
- [Page](#)
- [NavigationWindow](#)
- [TabControl](#)

## Dialog Boxes

Dialog boxes provide targeted support for common user-interaction scenarios such as printing.

- [OpenFileDialog](#)
- [PrintDialog](#)
- [SaveFileDialog](#)

## User Information

User information controls provide contextual feedback or clarify an application's user interface. The user typically cannot interact with these controls.

- [AccessText](#)
- [Label](#)
- [Popup](#)
- [ProgressBar](#)
- [StatusBar](#)
- [TextBlock](#)
- [ToolTip](#)

## Documents

WPF includes several specialized controls for viewing documents. These controls optimize the reading experience, based on the targeted user scenario.

- [DocumentViewer](#)
- [FlowDocumentPageViewer](#)
- [FlowDocumentReader](#)
- [FlowDocumentScrollView](#)
- [StickyNoteControl](#)

## Input

Input controls enable the user to input text and other content.

- [TextBox](#)
- [RichTextBox](#)
- [PasswordBox](#)

## Media

WPF includes integrated support for hosting both audio and video content, as well as [codecs] for most popular image formats.

- [Image](#)
- [MediaElement](#)
- [SoundPlayerAction](#)

## Digital Ink

Digital ink controls provide integrated support for Tablet PC features, such as ink viewing and ink input.

- [InkCanvas](#)
- [InkPresenter](#)

## See also

- [Control Library](#)

# WPF Content Model

4 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) is a presentation platform that provides many controls and control-like types whose primary purpose is to display different types of content. To determine which control to use or which control to derive from, you should understand the kinds of objects a particular control can best display.

This topic summarizes the content model for WPF control and control-like types. The content model describes what content can be used in a control. This topic also lists the content properties for each content model. A content property is a property that is used to store the content of the object.

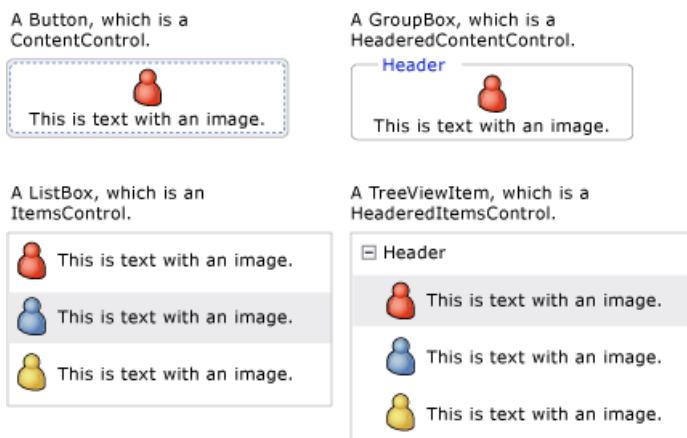
## Classes That Contain Arbitrary Content

Some controls can contain an object of any type, such as a string, a `DateTime` object, or a `UIElement` that is a container for additional items. For example, a `Button` can contain an image and some text; or a `CheckBox` can contain the value of `DateTime.Now`.

WPF has four classes that can contain arbitrary content. The following table lists the classes, which inherit from `Control`.

CLASS THAT CONTAINS ARBITRARY CONTENT	CONTENT
<code>ContentControl</code>	A single arbitrary object.
<code>HeaderedContentControl</code>	A header and a single item, both of which are arbitrary objects.
<code>ItemsControl</code>	A collection of arbitrary objects.
<code>HeaderedItemsControl</code>	A header and a collection of items, all of which are arbitrary objects.

Controls that inherit from these classes can contain the same type of content and treat the content in the same way. The following illustration shows one control from each content model that contains an image and some text:

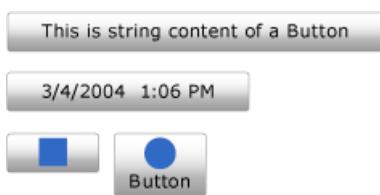


## Controls That Contain a Single Arbitrary Object

The `ContentControl` class contains a single piece of arbitrary content. Its content property is `Content`. The following controls inherit from `ContentControl` and use its content model:

- [Button](#)
- [ButtonBase](#)
- [CheckBox](#)
- [ComboBoxItem](#)
- [ContentControl](#)
- [Frame](#)
- [GridViewColumnHeader](#)
- [GroupItem](#)
- [Label](#)
- [ListBoxItem](#)
- [ListViewItem](#)
- [NavigationWindow](#)
- [RadioButton](#)
- [RepeatButton](#)
- [ScrollViewer](#)
- [StatusBarItem](#)
- [ToggleButton](#)
- [ToolTip](#)
- [UserControl](#)
- [Window](#)

The following illustration shows four buttons whose [Content](#) is set to a string, a [DateTime](#) object, a [Rectangle](#), and a [Panel](#) that contains an [Ellipse](#) and a [TextBlock](#):



For an example of how to set the [Content](#) property, see [ContentControl](#).

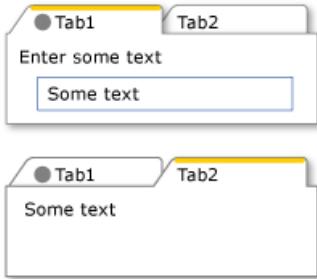
### Controls That Contain a Header and a Single Arbitrary Object

The [HeaderedContentControl](#) class inherits from [ContentControl](#) and displays content with a header. It inherits the content property, [Content](#), from [ContentControl](#) and defines the [Header](#) property that is of type [Object](#); therefore, both can be an arbitrary object.

The following controls inherit from [HeaderedContentControl](#) and use its content model:

- [Expander](#)
- [GroupBox](#)
- [TabItem](#)

The following illustration shows two [TabItem](#) objects. The first [TabItem](#) has [UIElement](#) objects as the [Header](#) and the [Content](#). The [Header](#) is set to a [StackPanel](#) that contains an [Ellipse](#) and a [TextBlock](#). The [Content](#) is set to a [StackPanel](#) that contains a [TextBlock](#) and a [Label](#). The second [TabItem](#) has a string in the [Header](#) and a [TextBlock](#) in the [Content](#).



For an example of how to create [TabItem](#) objects, see [HeaderedContentControl](#).

### Controls That Contain a Collection of Arbitrary Objects

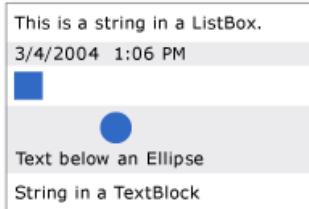
The [ItemsControl](#) class inherits from [Control](#) and can contain multiple items, such as strings, objects, or other elements. Its content properties are [ItemsSource](#) and [Items](#). [ItemsSource](#) is typically used to populate the [ItemsControl](#) with a data collection. If you do not want to use a collection to populate the [ItemsControl](#), you can add items by using the [Items](#) property.

The following controls inherit from [ItemsControl](#) and use its content model:

- [Menu](#)
- [MenuBase](#)
- [ContextMenu](#)
- [ComboBox](#)
- [ItemsControl](#)
- [ListBox](#)
- [ListView](#)
- [TabControl](#)
- [TreeView](#)
- [Selector](#)
- [StatusBar](#)

The following illustration shows a [ListBox](#) that contains these types of items:

- A string.
- A [DateTime](#) object.
- A [UIElement](#).
- A [Panel](#) that contains an [Ellipse](#) and a [TextBlock](#).



## Controls That Contain a Header and a Collection of Arbitrary Objects

The [HeaderedItemsControl](#) class inherits from [ItemsControl](#) and can contain multiple items, such as strings, objects, or other elements, and a header. It inherits the [ItemsControl](#) content properties, [ItemsSource](#), and [Items](#), and it defines the [Header](#) property that can be an arbitrary object.

The following controls inherit from [HeaderedItemsControl](#) and use its content model:

- [MenuItem](#)
- [ToolBar](#)
- [TreeViewItem](#)

## Classes That Contain a Collection of UIElement Objects

The [Panel](#) class positions and arranges child [UIElement](#) objects. Its content property is [Children](#).

The following classes inherit from the [Panel](#) class and use its content model:

- [Canvas](#)
- [DockPanel](#)
- [Grid](#)
- [TabPanel](#)
- [ToolBarOverflowPanel](#)
- [ToolBarPanel](#)
- [UniformGrid](#)
- [StackPanel](#)
- [VirtualizingPanel](#)
- [VirtualizingStackPanel](#)
- [WrapPanel](#)

For more information, see [Panels Overview](#).

## Classes That Affect the Appearance of a UIElement

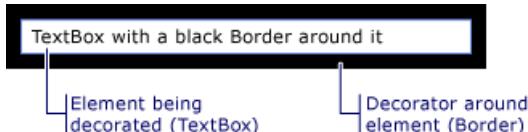
The [Decorator](#) class applies visual effects onto or around a single child [UIElement](#). Its content property is [Child](#).

The following classes inherit from [Decorator](#) and use its content model:

- [AdornerDecorator](#)
- [Border](#)
- [BulletDecorator](#)
- [ButtonChrome](#)

- [ClassicBorderDecorator](#)
- [InkPresenter](#)
- [ListBoxChrome](#)
- [SystemDropShadowChrome](#)
- [Viewbox](#)

The following illustration shows a [TextBox](#) that has (is decorated with) a [Border](#) around it.



TextBlock that has a Border

## Classes That Provide Visual Feedback About a UIElement

The [Adorner](#) class provides visual cues to a user. For example, use an [Adorner](#) to add functional handles to elements or provide state information about a control. The [Adorner](#) class provides a framework so that you can create your own adorners. WPF does not provide any implemented adorners. For more information, see [Adorners Overview](#).

## Classes That Enable Users to Enter Text

WPF provides three primary controls that enable users to enter text. Each control displays the text differently. The following table lists these three text-related controls, their capabilities when they display text, and their properties that contain the control's text.

CONTROL	TEXT IS DISPLAYED AS	CONTENT PROPERTY
TextBox	Plain text	Text
RichTextBox	Formatted text	Document
PasswordBox	Hidden text (characters are masked)	Password

## Classes That Display Your Text

Several classes can be used to display plain or formatted text. You can use [TextBlock](#) to display small amounts of text. If you want to display large amounts of text, use the [FlowDocumentReader](#), [FlowDocumentPageViewer](#), or [FlowDocumentScrollViewer](#) controls.

The [TextBlock](#) has two content properties: [Text](#) and [Inlines](#). When you want to display text that uses consistent formatting, the [Text](#) property is often your best choice. If you plan to use different formatting throughout the text, use the [Inlines](#) property. The [Inlines](#) property is a collection of [Inline](#) objects, which specify how to format text.

The following table lists the content property for [FlowDocumentReader](#), [FlowDocumentPageViewer](#), and [FlowDocumentScrollViewer](#) classes.

CONTROL	CONTENT PROPERTY	CONTENT PROPERTY TYPE
FlowDocumentPageViewer	Document	IDocumentPaginatorSource

CONTROL	CONTENT PROPERTY	CONTENT PROPERTY TYPE
FlowDocumentReader	Document	FlowDocument
FlowDocumentScrollView	Document	FlowDocument

The [FlowDocument](#) implements the [IDocumentPaginatorSource](#) interface; therefore, all three classes can take a [FlowDocument](#) as content.

## Classes That Format Your Text

[TextElement](#) and its related classes allow you to format text. [TextElement](#) objects contain and format text in [TextBlock](#) and [FlowDocument](#) objects. The two primary types of [TextElement](#) objects are [Block](#) elements and [Inline](#) elements. A [Block](#) element represents a block of text, such as a paragraph or list. An [Inline](#) element represents a portion of text in a block. Many [Inline](#) classes specify formatting for the text to which they are applied. Each [TextElement](#) has its own content model. For more information, see the [TextElement Content Model Overview](#).

## See also

- [Advanced](#)

# Control Library

2 minutes to read • [Edit Online](#)

The Windows Presentation Foundation (WPF) control library contains information on the controls provided by Windows Presentation Foundation (WPF), listed alphabetically.

## In This Section

[Border](#)  
[BulletDecorator](#)  
[Button](#)  
[Calendar](#)  
[Canvas](#)  
[CheckBox](#)  
[ComboBox](#)  
[ContextMenu](#)  
[DataGrid](#)  
[DatePicker](#)  
[DockPanel](#)  
[DocumentViewer](#)  
[Expander](#)  
[FlowDocumentPageViewer](#)  
[FlowDocumentReader](#)  
[FlowDocumentScrollView](#)  
[Frame](#)  
[Grid](#)  
[GridSplitter](#)  
[GroupBox](#)  
[Image](#)  
[Label](#)  
[ListBox](#)  
[ListView](#)  
[Menu](#)  
[Panel](#)  
[PasswordBox](#)  
[Popup](#)  
[ProgressBar](#)  
[PrintDialog](#)  
[RadioButton](#)  
[RepeatButton](#)  
[RichTextBox](#)  
[ScrollBar](#)  
[ScrollViewer](#)  
[Separator](#)  
[Slider](#)  
[StackPanel](#)  
[StatusBar](#)  
[TabControl](#)  
[TextBlock](#)

[TextBox](#)  
[ToolBar](#)  
[ToolTip](#)  
[TreeView](#)  
[WrapPanel](#)  
[Viewbox](#)

## Reference

[System.Windows.Controls](#)

[System.Windows.Controls.Primitives](#)

## Related Sections

[Control Customization](#)

[Controls by Category](#)

[WPF Content Model](#)

# Border

2 minutes to read • [Edit Online](#)

The following sample demonstrates how to dynamically change properties of the `Border` element.

## In This Section

[Animate a BorderThickness Value](#)

## Reference

[Decorator](#)

[Border](#)

## Related Sections

[Panels Overview](#)

[Alignment, Margins, and Padding Overview](#)

# How to: Animate a BorderThickness Value

2 minutes to read • [Edit Online](#)

This example shows how to animate changes to the thickness of a border by using the [ThicknessAnimation](#) class.

## Example

The following example animates the thickness of a border by using [ThicknessAnimation](#). The example uses the [BorderThickness](#) property of [Border](#).

```

using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Shapes;
using System.Windows.Media.Animation;
using System.Windows.Media;

namespace SDKSamples
{
    public class ThicknessAnimationExample : Page
    {
        public ThicknessAnimationExample()
        {

            // Create a NameScope for this page so that
            // Storyboards can be used.
            NameScope.SetNameScope(this, new NameScope());

            // Create a Border which will be the target of the animation.
            Border myBorder = new Border();
            myBorder.Background = Brushes.Gray;
            myBorder.BorderBrush = Brushes.Black;
            myBorder.BorderThickness = new Thickness(1);
            myBorder.Margin = new Thickness(0, 60, 0, 20);
            myBorder.Padding = new Thickness(20);

            // Assign the border a name so that
            // it can be targeted by a Storyboard.
            this.RegisterName(
                "myAnimatedBorder", myBorder);

            ThicknessAnimation myThicknessAnimation = new ThicknessAnimation();
            myThicknessAnimation.Duration = TimeSpan.FromSeconds(1.5);
            myThicknessAnimation.FillBehavior = FillBehavior.HoldEnd;

            // Set the From and To properties of the animation.
            // BorderThickness animates from left=1, right=1, top=1, and bottom=1
            // to left=28, right=28, top=14, and bottom=14 over one and a half seconds.
            myThicknessAnimation.From = new Thickness(1, 1, 1, 1);
            myThicknessAnimation.To = new Thickness(28, 14, 28, 14);

            // Set the animation to target the Size property
            // of the object named "myArcSegment."
            Storyboard.SetTargetName(myThicknessAnimation, "myAnimatedBorder");
            Storyboard.SetTargetProperty(
                myThicknessAnimation, new PropertyPath(Border.BorderThicknessProperty));

            // Create a storyboard to apply the animation.
            Storyboard ellipseStoryboard = new Storyboard();
            ellipseStoryboard.Children.Add(myThicknessAnimation);

            // Start the storyboard when the Path loads.
            myBorder.Loaded += delegate(object sender, RoutedEventArgs e)
            {
                ellipseStoryboard.Begin(this);
            };

            StackPanel myStackPanel = new StackPanel();
            myStackPanel.HorizontalAlignment = HorizontalAlignment.Center;
            myStackPanel.Children.Add(myBorder);

            Content = myStackPanel;
        }
    }
}

```

```

Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Shapes
Imports System.Windows.Media.Animation
Imports System.Windows.Media

Namespace SDKSamples
    Public Class ThicknessAnimationExample
        Inherits Page
        Public Sub New()

            ' Create a NameScope for this page so that
            ' Storyboards can be used.
            NameScope.SetNameScope(Me, New NameScope())

            ' Create a Border which will be the target of the animation.
            Dim myBorder As New Border()
            With myBorder
                .Background = Brushes.Gray
                .BorderBrush = Brushes.Black
                .BorderThickness = New Thickness(1)
                .Margin = New Thickness(0, 60, 0, 20)
                .Padding = New Thickness(20)
            End With

            ' Assign the border a name so that
            ' it can be targeted by a Storyboard.
            Me.RegisterName("myAnimatedBorder", myBorder)

            Dim myThicknessAnimation As New ThicknessAnimation()
            With myThicknessAnimation
                .Duration = TimeSpan.FromSeconds(1.5)
                .FillBehavior = FillBehavior.HoldEnd

                ' Set the From and To properties of the animation.
                ' BorderThickness animates from left=1, right=1, top=1, and bottom=1
                ' to left=28, right=28, top=14, and bottom=14 over one and a half seconds.
                .From = New Thickness(1, 1, 1, 1)
                .To = New Thickness(28, 14, 28, 14)
            End With

            ' Set the animation to target the Size property
            ' of the object named "myArcSegment."
            Storyboard.SetTargetName(myThicknessAnimation, "myAnimatedBorder")
            Storyboard.SetTargetProperty(myThicknessAnimation, New
PropertyPath(Border.BorderThicknessProperty))

            ' Create a storyboard to apply the animation.
            Dim ellipseStoryboard As New Storyboard()
            ellipseStoryboard.Children.Add(myThicknessAnimation)

            ' Start the storyboard when the Path loads.
            AddHandler myBorder.Loaded, Sub(sender As Object, e As RoutedEventArgs)
ellipseStoryboard.Begin(Me)

            Dim myStackPanel As New StackPanel()
            myStackPanel.HorizontalAlignment = HorizontalAlignment.Center
            myStackPanel.Children.Add(myBorder)

            Content = myStackPanel
        End Sub
    End Class
End Namespace

```

For the complete sample, see [Animation Example Gallery](#).

## See also

- [ThicknessAnimation](#)
- [BorderThickness](#)
- [Border](#)
- [Animation Overview](#)
- [Animation and Timing How-to Topics](#)
- [Animate the Thickness of a Border by Using Key Frames](#)

# BulletDecorator

2 minutes to read • [Edit Online](#)

**BulletDecorator** has two content properties: **Bullet** and **Child**. The **Bullet** property defines the **UIElement** to use as a bullet. The **Child** property defines a **UIElement** that visually aligns with the bullet.

The following illustration shows examples of controls that use a **BulletDecorator**.

- A BulletDecorator with a CheckBox Bullet.
- A BulletDecorator with a RadioButton Bullet.
- A BulletDecorator with a TextBox Bullet.

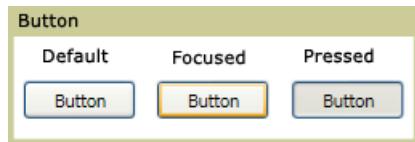
## Reference

[BulletDecorator](#)

# Button

2 minutes to read • [Edit Online](#)

A [Button](#) control reacts to user input from a mouse, keyboard, stylus, or other input device and raises a [Click](#) event. A [Button](#) is a basic user interface (UI) component that can contain simple content, such as text, and can also contain complex content, such as images and [Panel](#) controls.



## In This Section

[Create a Button That Has an Image](#)

## Reference

[Button](#)

[ButtonBase](#)

[RadioButton](#)

[RepeatButton](#)

# How to: Create a Button That Has an Image

2 minutes to read • [Edit Online](#)

This example shows how to include an image on a [Button](#).

## Example

The following example creates two [Button](#) controls. One [Button](#) contains text and the other contains an image. The image is in a folder called data, which is a subfolder of the example's project folder. When a user clicks the [Button](#) that has the image, the background and the text of the other [Button](#) change.

This example creates [Button](#) controls by using markup but uses code to write the [Click](#) event handlers.

```
<Button Name="btn5" Width="50" Height="30" Click="OnClick5">
    <Image Source="data\flower.jpg"></Image>
</Button>
<Button Name="btn6" BorderBrush="Black">Click the picture.</Button>
```

```
void OnClick5(object sender, RoutedEventArgs e)
{
    btn6.FontSize = 16;
    btn6.Content = "This is my favorite photo.";
    btn6.Background = Brushes.Red;
}
```

```
Private Sub OnClick5(ByVal sender As Object, ByVal e As RoutedEventArgs)
    btn6.FontSize = 16
    btn6.Content = "This is my favorite photo."
    btn6.Background = Brushes.Red
End Sub
```

## See also

- [Controls](#)
- [Control Library](#)

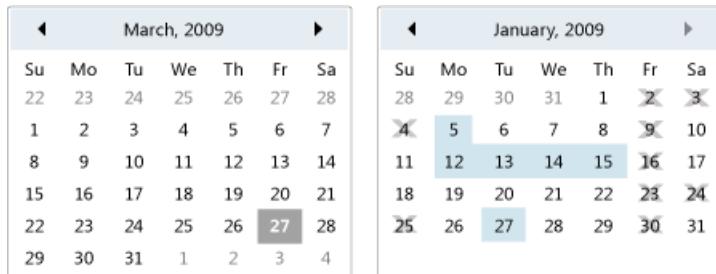
# Calendar

2 minutes to read • [Edit Online](#)

A calendar enables a user to select a date by using a visual calendar display.

A [Calendar](#) control can be used on its own, or as a drop-down part of a [DatePicker](#) control. For more information, see [DatePicker](#).

The following illustration shows two [Calendar](#) controls, one with selections and blackout dates and one without.



## Calendar controls

The following table provides information about tasks that are typically associated with the [Calendar](#).

TASK	IMPLEMENTATION
Specify dates that cannot be selected.	Use the <a href="#">BlackoutDates</a> property.
Have the <a href="#">Calendar</a> display a month, an entire year, or a decade.	Set the <a href="#">DisplayMode</a> property to Month, Year, or Decade.
Specify whether the user can select a date, a range of dates, or multiple ranges of dates.	Use the <a href="#">SelectionMode</a> .
Specify the range of dates that the <a href="#">Calendar</a> displays.	Use the <a href="#">DisplayDateStart</a> and <a href="#">DisplayDateEnd</a> properties.
Specify whether the current date is highlighted.	Use the <a href="#">IsTodayHighlighted</a> property. By default, <a href="#">IsTodayHighlighted</a> is <code>true</code> .
Change the size of the <a href="#">Calendar</a> .	Use a <a href="#">Viewbox</a> or set the <a href="#">LayoutTransform</a> property to a <a href="#">ScaleTransform</a> . Note that if you set the <a href="#">Width</a> and <a href="#">Height</a> properties of a <a href="#">Calendar</a> , the actual calendar does not change its size.

The [Calendar](#) control provides basic navigation using either the mouse or keyboard. The following table summarizes keyboard navigation.

KEY COMBINATION	DISPLAYMODE	ACTION
ARROW	Month	Changes the <a href="#">SelectedDate</a> property if the <a href="#">SelectionMode</a> property is not set to <a href="#">None</a> .

KEY COMBINATION	DISPLAYMODE	ACTION
ARROW	Year	Changes the month of the <a href="#">DisplayDate</a> property. Note that the <a href="#">SelectedDate</a> does not change.
ARROW	Decade	Changes the year of the <a href="#">DisplayDate</a> . Note that the <a href="#">SelectedDate</a> does not change.
SHIFT+ARROW	Month	If <a href="#">SelectionMode</a> is not set to <a href="#">SingleDate</a> or <a href="#">None</a> , extends the range of selected dates.
HOME	Month	Changes the <a href="#">SelectedDate</a> to the first day of the current month.
HOME	Year	Changes the month of the <a href="#">DisplayDate</a> to the first month of the year. The <a href="#">SelectedDate</a> does not change.
HOME	Decade	Changes the year of the <a href="#">DisplayDate</a> to the first year of the decade. The <a href="#">SelectedDate</a> does not change.
END	Month	Changes the <a href="#">SelectedDate</a> to the last day of the current month.
END	Year	Changes the month of the <a href="#">DisplayDate</a> to the last month of the year. The <a href="#">SelectedDate</a> does not change.
END	Decade	Changes the year of the <a href="#">DisplayDate</a> to the last year of the decade. The <a href="#">SelectedDate</a> does not change.
CTRL+UP ARROW	Any	Switches to the next larger <a href="#">DisplayMode</a> . If <a href="#">DisplayMode</a> is already <a href="#">Decade</a> , no action.
CTRL+DOWN ARROW	Any	Switches to the next smaller <a href="#">DisplayMode</a> . If <a href="#">DisplayMode</a> is already <a href="#">Month</a> , no action.
SPACEBAR or ENTER	Year or Decade	Switches <a href="#">DisplayMode</a> to the <a href="#">Month</a> or <a href="#">Year</a> represented by focused item.

## See also

- [Controls](#)
- [Styling and Templating](#)

# Canvas

2 minutes to read • [Edit Online](#)

[Canvas](#) is a layout control that enables absolute positioning of child elements.

## In This Section

### [How-to Topics](#)

## Reference

[Panel](#)

[Canvas](#)

[DockPanel](#)

[Grid](#)

[StackPanel](#)

[VirtualizingStackPanel](#)

[WrapPanel](#)

## Related Sections

[Layout](#)

[Walkthrough: My first WPF desktop application](#)

[ScrollViewer Overview](#)

# Canvas How-to Topics

2 minutes to read • [Edit Online](#)

The topics in this section describe how to use the [Canvas](#) element to absolutely position child elements.

## In This Section

- [Wrap a Border Around the Content of a Canvas](#)
- [Get or Set Canvas Positioning Properties](#)
- [Create and Use a Canvas](#)
- [Use the Attached Properties of Canvas to Position Child Elements](#)
- [Resize a Canvas by Using a Thumb](#)

## Reference

- [Panel](#)
- [Canvas](#)
- [DockPanel](#)
- [Grid](#)
- [StackPanel](#)
- [VirtualizingStackPanel](#)
- [WrapPanel](#)

## Related Sections

- [Layout](#)
- [Walkthrough: My first WPF desktop application](#)
- [ScrollViewer Overview](#)

# How to: Wrap a Border Around the Content of a Canvas

2 minutes to read • [Edit Online](#)

This example shows how to wrap a [Canvas](#) element with a [Border](#).

## Example

The following example shows how to display `Hello World!` inside a [Canvas](#) element. The [Canvas](#) element is wrapped by a [Border](#) element so that a border outlines the element.

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
    <Border
        BorderThickness="2"
        BorderBrush="Black"
        Background="LightGray"
        HorizontalAlignment="Left"
        VerticalAlignment="Top"
        Width="100"
        Height="100">
        <Canvas>
            <TextBlock Canvas.Top="10" Canvas.Left="20">Hello World!</TextBlock>
        </Canvas>
    </Border>
</Page>
```

## See also

- [Canvas](#)
- [Border](#)
- [Panels Overview](#)

# How to: Get or Set Canvas Positioning Properties

2 minutes to read • [Edit Online](#)

This example shows how to use the positioning methods of the [Canvas](#) element to position child content. This example uses content in a [ListBoxItem](#) to represent positioning values and converts the values into instances of [Double](#), which is a required argument for positioning. The values are then converted back into strings and displayed as text in a [TextBlock](#) element by using the [GetLeft](#) method.

## Example

The following example creates a [ListBox](#) element that has eleven selectable [ListBoxItem](#) elements. The [SelectionChanged](#) event triggers the `ChangeLeft` custom method, which the subsequent code block defines.

Each [ListBoxItem](#) represents a [Double](#) value, which is one of the arguments that the [SetLeft](#) method of [Canvas](#) accepts. In order to use a [ListBoxItem](#) to represent an instance of [Double](#), you must first convert the [ListBoxItem](#) to the correct data type.

```
<ListBox Grid.Column="1" Grid.Row="1" VerticalAlignment="Top" Width="60" Margin="10,0,0,0"
SelectionChanged="ChangeLeft">
    <ListBoxItem>Auto</ListBoxItem>
    <ListBoxItem>10</ListBoxItem>
    <ListBoxItem>20</ListBoxItem>
    <ListBoxItem>30</ListBoxItem>
    <ListBoxItem>40</ListBoxItem>
    <ListBoxItem>50</ListBoxItem>
    <ListBoxItem>60</ListBoxItem>
    <ListBoxItem>70</ListBoxItem>
    <ListBoxItem>80</ListBoxItem>
    <ListBoxItem>90</ListBoxItem>
    <ListBoxItem>100</ListBoxItem>
</ListBox>
```

When a user changes the [ListBox](#) selection, it invokes the `ChangeLeft` custom method. This method passes the [ListBoxItem](#) to a [LengthConverter](#) object, which converts the [Content](#) of a [ListBoxItem](#) to an instance of [Double](#) (notice that this value has already been converted to a [String](#) by using the [ToString](#) method). This value is then passed back to the [SetLeft](#) and [GetLeft](#) methods of [Canvas](#) in order to change the position of the `text1` object.

```
private void ChangeLeft(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    LengthConverter myLengthConverter = new LengthConverter();
    Double db1 = (Double)myLengthConverter.ConvertFromString(li.Content.ToString());
    Canvas.SetLeft(text1, db1);
    String st1 = (String)myLengthConverter.ConvertToString(Canvas.GetLeft(text1));
    canvasLeft.Text = "Canvas.Left = " + st1;
}
```

```
Private Sub ChangeLeft(ByVal sender As Object, ByVal e As SelectionChangedEventArgs)
    Dim li As ListBoxItem = CType(CType(sender, ListBox).SelectedItem, ListBoxItem)
    Dim myLengthConverter As New LengthConverter
    Dim db1 As Double = CType(myLengthConverter.ConvertFromString(li.Content.ToString()), Double)
    Canvas.SetLeft(text1, db1)
    Dim st1 As String = CType(myLengthConverter.ConvertToString(Canvas.GetLeft(text1)), String)
    canvasLeft.Text = "Canvas.Left = " + st1
End Sub
```

## See also

- [Canvas](#)
- [ListBoxItem](#)
- [LengthConverter](#)
- [Panels Overview](#)

# How to: Create and Use a Canvas

2 minutes to read • [Edit Online](#)

This example shows how to create and use an instance of [Canvas](#).

## Example

The following example explicitly positions two [TextBlock](#) elements by using the [SetTop](#) and [SetLeft](#) methods of [Canvas](#). The example also assigns a [Background](#) color of [LightSteelBlue](#) to the [Canvas](#).

### NOTE

When you use Extensible Application Markup Language (XAML) to position [TextBlock](#) elements, use the [Top](#) and [Left](#) properties.

```
private void CreateAndShowMainWindow()
{
    // Create the application's main window
    mainWindow = new Window();

    // Create a canvas sized to fill the window
    Canvas myCanvas = new Canvas();
    myCanvas.Background = Brushes.LightSteelBlue;

    // Add a "Hello World!" text element to the Canvas
    TextBlock txt1 = new TextBlock();
    txt1.FontSize = 14;
    txt1.Text = "Hello World!";
    Canvas.SetTop(txt1, 100);
    Canvas.SetLeft(txt1, 10);
    myCanvas.Children.Add(txt1);

    // Add a second text element to show how absolute positioning works in a Canvas
    TextBlock txt2 = new TextBlock();
    txt2.FontSize = 22;
    txt2.Text = "Isn't absolute positioning handy?";
    Canvas.SetTop(txt2, 200);
    Canvas.SetLeft(txt2, 75);
    myCanvas.Children.Add(txt2);
    mainWindow.Content = myCanvas;
    mainWindow.Title = "Canvas Sample";
    mainWindow.Show();
}
```

```
WindowTitle = "Canvas Sample"
'Create a Canvas as the root Panel
Dim myCanvas As New Canvas()
myCanvas.Background = Brushes.LightSteelBlue

Dim txt1 As New TextBlock
txt1.FontSize = 14
txt1.Text = "Hello World!"
Canvas.SetLeft(txt1, 10)
Canvas.SetTop(txt1, 100)
myCanvas.Children.Add(txt1)

'Add a second text element to show how absolute positioning works in a Canvas
Dim txt2 As New TextBlock
txt2.FontSize = 22
txt2.Text = "Isn't absolute positioning handy?"
Canvas.SetLeft(txt2, 75)
Canvas.SetTop(txt2, 200)
myCanvas.Children.Add(txt2)
Me.Content = myCanvas
```

## See also

- [Canvas](#)
- [TextBlock](#)
- [SetTop](#)
- [SetLeft](#)
- [Top](#)
- [Left](#)
- [Panels Overview](#)
- [How-to Topics](#)

# How to: Use the Attached Properties of Canvas to Position Child Elements

2 minutes to read • [Edit Online](#)

This example shows how to use the attached properties of `Canvas` to position child elements.

## Example

The following example adds four `Button` elements as child elements of a parent `Canvas`. Each element is represented by a `Bottom`, `Left`, `Right`, and `Top`. Each `Button` is positioned relative to the parent `Canvas` and according to its assigned property value.

```

// Create the application's main window
mainWindow = gcnew System::Windows::Window();
mainWindow->Title = "Canvas Attached Properties Sample";

// Add a Border
Border^ myBorder = gcnew Border();
myBorder->HorizontalAlignment = HorizontalAlignment::Left;
myBorder->VerticalAlignment = VerticalAlignment::Top;
myBorder->BorderBrush = Brushes::Black;
myBorder->BorderThickness = System::Windows::Thickness(2);

// Create the Canvas
Canvas^ myCanvas = gcnew Canvas();
myCanvas->Background = Brushes::LightBlue;
myCanvas->Width = 400;
myCanvas->Height = 400;

// Create the child Button elements
Button^ myButton1 = gcnew Button();
Button^ myButton2 = gcnew Button();
Button^ myButton3 = gcnew Button();
Button^ myButton4 = gcnew Button();

// Set Positioning attached properties on Button elements
Canvas::SetTop(myButton1, 50);
myButton1->Content = "Canvas.Top=50";
Canvas::SetBottom(myButton2, 50);
myButton2->Content = "Canvas.Bottom=50";
Canvas::SetLeft(myButton3, 50);
myButton3->Content = "Canvas.Left=50";
Canvas::SetRight(myButton4, 50);
myButton4->Content = "Canvas.Right=50";

// Add Buttons to the Canvas' Children collection
myCanvas->Children->Add(myButton1);
myCanvas->Children->Add(myButton2);
myCanvas->Children->Add(myButton3);
myCanvas->Children->Add(myButton4);

// Add the Canvas as the lone Child of the Border
myBorder->Child = myCanvas;

// Add the Border as the Content of the Parent Window Object
mainWindow->Content = myBorder;
mainWindow->Show();

```

```
// Create the application's main window
mainWindow = new Window ();
mainWindow.Title = "Canvas Attached Properties Sample";

// Add a Border
Border myBorder = new Border();
myBorder.HorizontalAlignment = HorizontalAlignment.Left;
myBorder.VerticalAlignment = VerticalAlignment.Top;
myBorder.BorderBrush = Brushes.Black;
myBorder.BorderThickness = new Thickness(2);

// Create the Canvas
Canvas myCanvas = new Canvas();
myCanvas.Background = Brushes.LightBlue;
myCanvas.Width = 400;
myCanvas.Height = 400;

// Create the child Button elements
Button myButton1 = new Button();
Button myButton2 = new Button();
Button myButton3 = new Button();
Button myButton4 = new Button();

// Set Positioning attached properties on Button elements
Canvas.SetTop(myButton1, 50);
myButton1.Content = "Canvas.Top=50";
Canvas.SetBottom(myButton2, 50);
myButton2.Content = "Canvas.Bottom=50";
Canvas.SetLeft(myButton3, 50);
myButton3.Content = "Canvas.Left=50";
Canvas.SetRight(myButton4, 50);
myButton4.Content = "Canvas.Right=50";

// Add Buttons to the Canvas' Children collection
myCanvas.Children.Add(myButton1);
myCanvas.Children.Add(myButton2);
myCanvas.Children.Add(myButton3);
myCanvas.Children.Add(myButton4);

// Add the Canvas as the lone Child of the Border
myBorder.Child = myCanvas;

// Add the Border as the Content of the Parent Window Object
mainWindow.Content = myBorder;
mainWindow.Show ();
```

```

WindowTitle = "Attached Properties Sample"
' Add a Border
Dim myBorder As New Border()
myBorder.HorizontalAlignment = Windows.HorizontalAlignment.Left
myBorder.VerticalAlignment = Windows.VerticalAlignment.Top
myBorder.BorderBrush = Brushes.Black
myBorder.BorderThickness = New Thickness(2)

' Create a Canvas as the root Panel
Dim myCanvas As New Canvas()
myCanvas.Background = Brushes.LightBlue
myCanvas.Width = 400
myCanvas.Height = 400

' Create the child Button elements
Dim myButton1 As New Button()
Dim myButton2 As New Button()
Dim myButton3 As New Button()
Dim myButton4 As New Button()

' Set Positioning attached properties on Button elements
Canvas.SetTop(myButton1, 50)
myButton1.Content = "Canvas.Top=50"
Canvas.SetBottom(myButton2, 50)
myButton2.Content = "Canvas.Bottom=50"
Canvas.SetLeft(myButton3, 50)
myButton3.Content = "Canvas.Left=50"
Canvas.SetRight(myButton4, 50)
myButton4.Content = "Canvas.Right=50"

' Add Buttons to the Canvas' Children collection
myCanvas.Children.Add(myButton1)
myCanvas.Children.Add(myButton2)
myCanvas.Children.Add(myButton3)
myCanvas.Children.Add(myButton4)

' Add the Canvas as the lone Child of the Border
myBorder.Child = myCanvas
Me.Content = myBorder

```

## See also

- [Canvas](#)
- [Bottom](#)
- [Left](#)
- [Right](#)
- [Top](#)
- [Button](#)
- [Panels Overview](#)
- [How-to Topics](#)
- [Attached Properties Overview](#)

# How to: Resize a Canvas by Using a Thumb

2 minutes to read • [Edit Online](#)

This example shows how to use a [Thumb](#) control to resize a [Canvas](#) control.

## Example

The [Thumb](#) control provides drag functionality that can be used to move or resize controls by monitoring the [DragStarted](#), [DragDelta](#) and [DragCompleted](#) events of the [Thumb](#).

The user begins a drag operation by pressing the left mouse button when the mouse pointer is paused on the [Thumb](#) control. The drag operation continues as long as the left mouse button remains pressed. During the drag operation, the [DragDelta](#) can occur more than once. Each time it occurs, the [DragDeltaEventArgs](#) class provides the change in position that corresponds to the change in mouse position. When the user releases the left mouse button, the drag operation is finished. The drag operation only provides new coordinates; it does not automatically reposition the [Thumb](#).

The following example shows a [Thumb](#) control that is the child element of a [Canvas](#) control. The event handler for its [DragDelta](#) event provides the logic to move the [Thumb](#) and resize the [Canvas](#). The event handlers for the [DragStarted](#) and [DragCompleted](#) event change the color of the [Thumb](#) during a drag operation. The following example defines the [Thumb](#).

```
<Thumb Name="myThumb" Canvas.Left="80" Canvas.Top="80" Background="Blue"
       Width="20" Height="20" DragDelta="onDragDelta"
       DragStarted="onDragStarted" DragCompleted="onDragCompleted"
    />
```

The following example shows the [DragDelta](#) event handler that moves the [Thumb](#) and resizes the [Canvas](#) in response to a mouse movement.

```
void onDragDelta(object sender, DragDeltaEventArgs e)
{
    //Move the Thumb to the mouse position during the drag operation
    double yadjust = myCanvasStretch.Height + e.VerticalChange;
    double xadjust = myCanvasStretch.Width + e.HorizontalChange;
    if ((xadjust >= 0) && (yadjust >= 0))
    {
        myCanvasStretch.Width = xadjust;
        myCanvasStretch.Height = yadjust;
        Canvas.SetLeft(myThumb, Canvas.GetLeft(myThumb) +
                       e.HorizontalChange);
        Canvas.SetTop(myThumb, Canvas.GetTop(myThumb) +
                     e.VerticalChange);
        changes.Text = "Size: " +
                      myCanvasStretch.Width.ToString() +
                      ", " +
                      myCanvasStretch.Height.ToString();
    }
}
```

The following example shows the [DragStarted](#) event handler.

```
void onDragStarted(object sender, DragStartedEventArgs e)
{
    myThumb.Background = Brushes.Orange;
}
```

```
Private Sub onDragStarted(ByVal sender As Object, ByVal e As DragStartedEventArgs)
    myThumb.Background = Brushes.Orange
End Sub
```

The following example shows the [DragCompleted](#) event handler.

```
void onDragCompleted(object sender, DragCompletedEventArgs e)
{
    myThumb.Background = Brushes.Blue;
}
```

```
Private Sub onDragCompleted(ByVal sender As Object, _
                           ByVal e As DragCompletedEventArgs)
    myThumb.Background = Brushes.Blue
End Sub
```

For the complete sample, see [Thumb Drag Functionality Sample](#).

## See also

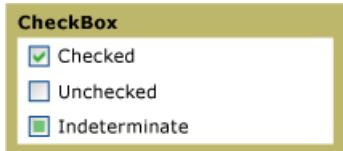
- [Thumb](#)
- [DragStarted](#)
- [DragDelta](#)
- [DragCompleted](#)

# CheckBox

2 minutes to read • [Edit Online](#)

You can use a [CheckBox](#) in the user interface (UI) of your application to represent options that a user can select or clear. You can use a single check box or you can group two or more check boxes.

The following graphic shows the different states of a [CheckBox](#).



CheckBox controls in different states

## Reference

[CheckBox](#)

[RadioButton](#)

[ButtonBase](#)

[RepeatButton](#)

## Related Sections

# ComboBox

2 minutes to read • [Edit Online](#)

The [ComboBox](#) control presents users with a list of options. The list is shown and hidden as the control expands and collapses. In its default state, the list is collapsed, displaying only one choice. The user clicks a button to see the complete list of options.

The following illustration shows a [ComboBox](#) in different states.



Collapsed and expanded

## Reference

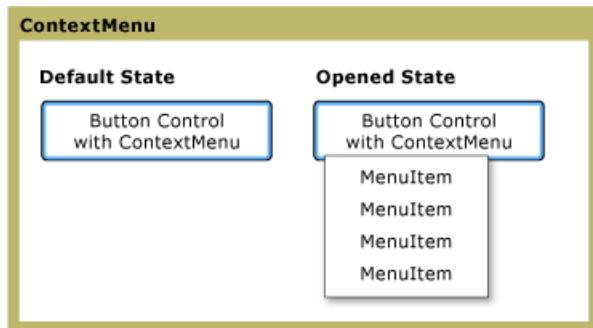
[ComboBox](#)

# ContextMenu

2 minutes to read • [Edit Online](#)

The [ContextMenu](#) allows a control to display a [Menu](#) that is specific to the context of the control. Typically, the [ContextMenu](#) is exposed in the user interface (UI) through the right mouse button or through the keyboard's menu button.

The following figure illustrates a [ContextMenu](#) in two different states: the default state and the open state. In the default state, the control is collapsed. When the right mouse button is pressed over the parent of the menu, the control expands and displays the menu items.



ContextMenu in different states

## In This Section

[ContextMenu Overview](#)

## Reference

[ContextMenu](#)

## Related Sections

# ContextMenu Overview

2 minutes to read • [Edit Online](#)

The [ContextMenu](#) class represents the element that exposes functionality by using a context-specific [Menu](#).

Typically, a user exposes the [ContextMenu](#) in the user interface (UI) by right-clicking the mouse button. This topic introduces the [ContextMenu](#) element and provides examples of how to use it in Extensible Application Markup Language (XAML) and code.

## ContextMenu Control

A [ContextMenu](#) is attached to a specific control. The [ContextMenu](#) element enables you to present users with a list of items that specify commands or options that are associated with a particular control, for example, a [Button](#). Users right-click the control to make the menu appear. Typically, clicking a [MenuItem](#) opens a submenu or causes an application to carry out a command.

## Creating ContextMenus

The following examples show how to create a [ContextMenu](#) with submenus. The [ContextMenu](#) controls are attached to button controls.

```
<Button Name="cmButton" Height="30">
    Button with Context Menu
    <Button.ContextMenu>
        <ContextMenu Name="cm" Opened="OnOpened" Closed="OnClosed" StaysOpen="true">
            <MenuItem Header="File"/>
            <MenuItem Header="Save"/>
            <MenuItem Header="SaveAs"/>
            <MenuItem Header="Recent Files">
                <MenuItem Header="ReadMe.txt"/>
                <MenuItem Header="Schedule.xls"/>
            </MenuItem>
        </ContextMenu>
    </Button.ContextMenu>
</Button>
```

```
btn = new Button();
btn.Content = "Created with C#";
contextmenu = new ContextMenu();
btn.ContextMenu = contextmenu;
mi = new MenuItem();
mi.Header = "File";
mia = new MenuItem();
mia.Header = "New";
mi.Items.Add(mia);
mib = new MenuItem();
mib.Header = "Open";
mi.Items.Add(mib);
mib1 = new MenuItem();
mib1.Header = "Recently Opened";
mib.Items.Add(mib1);
mib1a = new MenuItem();
mib1a.Header = "Text.xaml";
mib1.Items.Add(mib1a);
contextmenu.Items.Add(mi);
cv2.Children.Add(btn);
```

```

Dim btn As New Button()
Dim contextmenu As New ContextMenu()
Dim mi As New MenuItem()
Dim mia As New MenuItem()

btn.Background = Brushes.Red
btn.Height = 30
btn.Content = "Created with Visual Basic."

mi.Header = ("Item 1")
contextmenu.Items.Add(mi)
mia.Header = ("Item 2")
contextmenu.Items.Add(mia)

btn.ContextMenu = (contextmenu)
cv2.Children.Add(btn)

```

## Applying Styles to a ContextMenu

By using a control [Style](#), you can dramatically change the appearance and behavior of a [ContextMenu](#) without writing a custom control. In addition to setting visual properties, you can also apply styles to parts of a control. For example, you can change the behavior of parts of the control by using properties, or you can add parts to, or change the layout of, a [ContextMenu](#). The following examples show several ways to add styles to [ContextMenu](#) controls.

The first example defines a style called `SimpleSysResources`, which shows how to use the current system settings in your style. The example assigns [MenuHighlightBrushKey](#) as the [Background](#) color and [MenuTextBrushKey](#) as the [Foreground](#) color of the [ContextMenu](#).

```

<Style x:Key="SimpleSysResources" TargetType="{x:Type MenuItem}">
    <Setter Property = "Background" Value=
        "{DynamicResource {x:Static SystemColors.MenuHighlightBrushKey}}"/>
    <Setter Property = "Foreground" Value=
        "{DynamicResource {x:Static SystemColors.MenuTextBrushKey}}"/>
</Style>

```

The following example uses the [Trigger](#) element to change the appearance of a [Menu](#) in response to events that are raised on the [ContextMenu](#). When a user moves the mouse over the menu, the appearance of the [ContextMenu](#) items changes.

```

<Style x:Key="Triggers" TargetType="{x:Type MenuItem}">
    <Style.Triggers>
        <Trigger Property="MenuItem.IsMouseOver" Value="true">
            <Setter Property = "FontSize" Value="16"/>
            <Setter Property = "FontStyle" Value="Italic"/>
            <Setter Property = "Foreground" Value="Red"/>
        </Trigger>
    </Style.Triggers>
</Style>

```

## See also

- [ContextMenu](#)
- [Style](#)
- [Menu](#)
- [MenuItem](#)

- [ContextMenu](#)
- [ContextMenu Styles and Templates](#)
- [WPF Controls Gallery Sample](#)

# DataGrid

2 minutes to read • [Edit Online](#)

The [DataGrid](#) control enables you to display and edit data from many different sources, such as from a SQL database, LINQ query, or any other bindable data source. For more information, see [Binding Sources Overview](#).

Columns can display text, controls, such as a [ComboBox](#), or any other WPF content, such as images, buttons, or any content contained in a template. You can use a [DataGridTemplateColumn](#) to display data defined in a template. The following table lists the column types that are provided by default.

GENERATED COLUMN TYPE	DATA TYPE
<a href="#">DataGridTextColumn</a>	<a href="#">String</a>
<a href="#">DataGridCheckBoxColumn</a>	<a href="#">Boolean</a>
<a href="#">DataGridComboBoxColumn</a>	<a href="#">Enum</a>
<a href="#">DataGridHyperlinkColumn</a>	<a href="#">Uri</a>

[DataGrid](#) can be customized in appearance, such as cell font, color, and size. [DataGrid](#) supports all styling and templating functionality of other WPF controls. [DataGrid](#) also includes default and customizable behaviors for editing, sorting, and validation.

The following table lists some of the common tasks for [DataGrid](#) and how to accomplish them. By viewing the related API, you can find more information and sample code.

SCENARIO	APPROACH
Alternating background colors	Set the <a href="#">AlternationIndex</a> property to 2 or more, and then assign a <a href="#">Brush</a> to the <a href="#">RowBackground</a> and <a href="#">AlternatingRowBackground</a> properties.
Define cell and row selection behavior	Set the <a href="#">SelectionMode</a> and <a href="#">SelectionUnit</a> properties.
Customize the visual appearance of headers, cells, and rows	Apply a new <a href="#">Style</a> to the <a href="#">ColumnHeaderStyle</a> , <a href="#">RowHeaderStyle</a> , <a href="#">CellStyle</a> , or <a href="#">RowStyle</a> properties.
Set sizing options	Set the <a href="#">Height</a> , <a href="#">MaxHeight</a> , <a href="#">MinHeight</a> , <a href="#">Width</a> , <a href="#">MaxWidth</a> , or <a href="#">MinWidth</a> properties. For more information, see <a href="#">Sizing Options in the DataGrid Control</a> .
Access selected items	Check the <a href="#">SelectedCells</a> property to get the selected cells and the <a href="#">SelectedItems</a> property to get the selected rows. For more information, see <a href="#">SelectedCells</a> .
Customize end-user interactions	Set the <a href="#">CanUserAddRows</a> , <a href="#">CanUserDeleteRows</a> , <a href="#">CanUserReorderColumns</a> , <a href="#">CanUserResizeColumns</a> , <a href="#">CanUserResizeRows</a> , and <a href="#">CanUserSortColumns</a> properties.
Cancel or change auto-generated columns	Handle the <a href="#">AutoGeneratingColumn</a> event.

SCENARIO	APPROACH
Freeze a column	Set the <a href="#">FrozenColumnCount</a> property to 1 and move the column to the left-most position by setting the <a href="#">DisplayIndex</a> property to 0.
Use XML data as the data source	Bind the <a href="#">ItemsSource</a> on the <a href="#">DataGrid</a> to the XPath query that represents the collection of items. Create each column in the <a href="#">DataGrid</a> . Bind each column by setting the XPath on the binding to the query that gets the property on the item source. For an example, see <a href="#">DataGridTextColumn</a> .

## Related Topics

TITLE	DESCRIPTION
<a href="#">Walkthrough: Display Data from a SQL Server Database in a DataGrid Control</a>	Describes how to set up a new WPF project, add an Entity Framework Element, set the source, and display the data in a <a href="#">DataGrid</a> .
<a href="#">How to: Add Row Details to a DataGrid Control</a>	Describes how to create row details for a <a href="#">DataGrid</a> .
<a href="#">How to: Implement Validation with the DataGrid Control</a>	Describes how to validate values in <a href="#">DataGrid</a> cells and rows, and display validation feedback.
<a href="#">Default Keyboard and Mouse Behavior in the DataGrid Control</a>	Describes how to interact with the <a href="#">DataGrid</a> control by using the keyboard and mouse.
<a href="#">How to: Group, Sort, and Filter Data in the DataGrid Control</a>	Describes how to view data in a <a href="#">DataGrid</a> in different ways by grouping, sorting, and filtering the data.
<a href="#">Sizing Options in the DataGrid Control</a>	Describes how to control absolute and automatic sizing in the <a href="#">DataGrid</a> .

## See also

- [DataGrid](#)
- [Styling and Templating](#)
- [Data Binding Overview](#)
- [Data Templating Overview](#)
- [Controls](#)
- [WPF Content Model](#)

# Default Keyboard and Mouse Behavior in the DataGrid Control

6 minutes to read • [Edit Online](#)

This topic describes how users can interact with the [DataGrid](#) control by using the keyboard and mouse.

Typical interactions with the [DataGrid](#) include navigation, selection, and editing. Selection behavior is affected by the [SelectionMode](#) and [SelectionUnit](#) properties. The default values that cause the behavior described in this topic are [DataGridSelectionMode.Extended](#) and [DataGridSelectionUnit.FullRow](#). Changing these values might cause behavior that is different from that described. When a cell is in edit mode, the editing control might override the standard keyboard behavior of the [DataGrid](#).

## Default Keyboard Behavior

The following table lists the default keyboard behavior for the [DataGrid](#).

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, pressing the DOWN ARROW does nothing.
UP ARROW	Moves the focus to the cell directly above the current cell. If the focus is in the first row, pressing the UP ARROW 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, pressing the LEFT ARROW 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, pressing the RIGHT ARROW 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	If rows are not grouped, 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.  If rows are grouped, moves the focus to the last row in the <a href="#">DataGrid</a> without changing columns.
PAGE UP	If rows are not grouped, scrolls the control upward by the number of rows that are fully displayed. Moves focus to the first displayed row without changing columns.  If rows are grouped, moves the focus to the first row in the <a href="#">DataGrid</a> without changing columns.

KEY OR KEY COMBINATION	DESCRIPTION
TAB	<p>Moves the focus to the next cell in the current row. If the focus is 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 current cell is in edit mode and pressing TAB causes focus to move away from the current row, any changes that were made to the row are committed before focus is changed.</p>
SHIFT+TAB	<p>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 current cell is in edit mode and pressing TAB causes focus to move away from the current row, any changes that were made to the row are committed before focus is changed.</p>
CTRL+ DOWN ARROW	Moves the focus to the last cell in the current column.
CTRL+ UP ARROW	Moves the focus to the first cell in the current column.
CTRL+ RIGHT ARROW	Moves the focus to the last cell in the current row.
CTRL+ LEFT ARROW	Moves the focus to the first cell in the current row.
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	Same as PAGE DOWN.
CTRL+ PAGE UP	Same as PAGE UP.
F2	If the <code>DataGridView.IsReadOnly</code> property is <code>false</code> and the <code>DataGridViewColumn.IsReadOnly</code> property is <code>false</code> for the current column, puts the current cell into cell edit mode.
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 and reverts any changes that were made in the control. If the underlying data source implements <code>IEditableObject</code> , pressing ESC a second time cancels edit mode for the entire row.
BACKSPACE	Deletes the character before the cursor when editing a cell.
DELETE	Deletes the character after the cursor when editing a cell.

KEY OR KEY COMBINATION	DESCRIPTION
CTRL+ENTER	Commits any changes to the current cell without moving the focus.
CTRL+A	If <a href="#">SelectionMode</a> is set to <a href="#">Extended</a> , selects all rows in the <a href="#">DataGrid</a> .

## Selection Keys

If the [SelectionMode](#) property is set to [Extended](#), the navigation behavior does not change, but navigating with the keyboard while pressing SHIFT (including CTRL+SHIFT) will modify a multi-row selection. Before navigation starts, the control marks the current row as an anchor row. When you navigate while pressing SHIFT, the selection includes all rows between the anchor row and the current row.

The following selection keys modify multi-row selection.

- SHIFT+DOWN ARROW
- SHIFT+UP ARROW
- SHIFT+PAGE DOWN
- SHIFT+PAGE UP
- CTRL+SHIFT+DOWN ARROW
- CTRL+SHIFT+UP ARROW
- CTRL+SHIFT+HOME
- CTRL+SHIFT+END

## Default Mouse Behavior

The following table lists the default mouse behavior for the [DataGrid](#).

MOUSE ACTION	DESCRIPTION
Click an unselected row	Makes the clicked row the current row, and the clicked cell the current cell.
Click the current cell	Puts the current cell into edit mode.
Drag a column header cell	If the <a href="#">DataGrid.CanUserReorderColumns</a> property is <code>true</code> and the <a href="#">DataGridColumn.CanUserReorder</a> property is <code>true</code> for the current column, moves the column so that it can be dropped into a new position.
Drag a column header separator	If the <a href="#">DataGrid.CanUserResizeColumns</a> property is <code>true</code> and the <a href="#">DataGridColumn.CanUserResize</a> property is <code>true</code> for the current column, resizes the column.
Double-click a column header separator	If the <a href="#">DataGrid.CanUserResizeColumns</a> property is <code>true</code> and the <a href="#">DataGridColumn.CanUserResize</a> property is <code>true</code> for the current column, auto-sizes the column using the <a href="#">Auto</a> sizing mode.

MOUSE ACTION	DESCRIPTION
Click a column header cell	If the <a href="#">DataGrid.CanUserSortColumns</a> property is <code>true</code> and the <a href="#">DataGridColumn.CanUserSort</a> property is <code>true</code> for the current column, sorts the column.  Clicking the header of a column that is already sorted will reverse the sort direction of that column.  Pressing the SHIFT key while clicking multiple column headers will sort by multiple columns in the order clicked.
CTRL+click a row	If <a href="#">SelectionMode</a> is set to <a href="#">Extended</a> , modifies a non-contiguous multi-row selection.  If the row is already selected, deselects the row.
SHIFT+click a row	If <a href="#">SelectionMode</a> is set to <a href="#">Extended</a> , modifies a contiguous multi-row selection.
Click a row group header	Expands or collapses the group.
Click the Select All button at the top left corner of the <a href="#">DataGrid</a>	If <a href="#">SelectionMode</a> is set to <a href="#">Extended</a> , selects all rows in the <a href="#">DataGrid</a> .

## Mouse Selection

If the [SelectionMode](#) property is set to [Extended](#), clicking a row while pressing CTRL or SHIFT will modify a multi-row selection.

When you click a row while pressing CTRL, the row will change its selection state while all other rows retain their current selection state. Do this to select non-adjacent rows.

When you click a row while pressing SHIFT, the selection includes all rows between the current row and an anchor row located at the position of the current row prior to the click. Subsequent clicks while pressing SHIFT change the current row, but not the anchor row. Do this to select a range of adjacent rows.

CTRL+SHIFT can be combined to select non-adjacent ranges of adjacent rows. To do this, select the first range by using SHIFT+click as described earlier. After the first range of rows is selected, use CTRL+click to select the first row in the next range, and then click the last row in the next range while pressing CTRL+SHIFT.

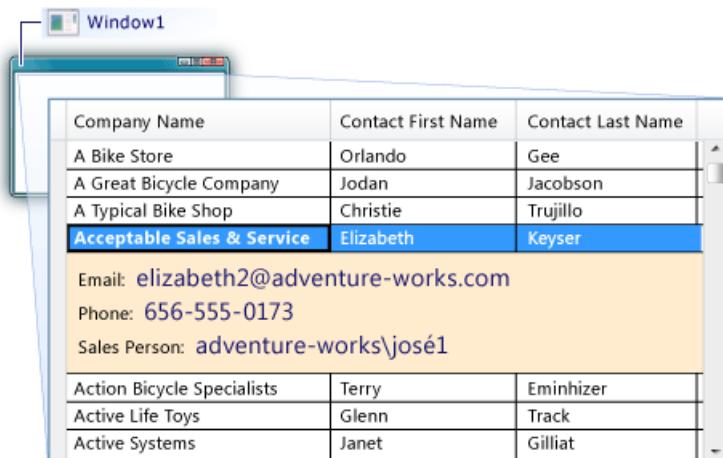
## See also

- [DataGrid](#)
- [SelectionMode](#)

# How to: Add Row Details to a DataGrid Control

3 minutes to read • [Edit Online](#)

When using the [DataGrid](#) control, you can customize the data presentation by adding a row details section. Adding a row details section enables you to group some data in a template that is optionally visible or collapsed. For example, you can add row details to a [DataGrid](#) that presents only a summary of the data for each row in the [DataGrid](#), but presents more data fields when the user selects a row. You define the template for the row details section in the [RowDetailsTemplate](#) property. The following illustration shows an example of a row details section.



You define the row details template as either inline XAML or as a resource. Both approaches are shown in the following procedures. A data template that is added as a resource can be used throughout the project without re-creating the template. A data template that is added as inline XAML is only accessible from the control where it is defined.

## To display row details by using inline XAML

1. Create a [DataGrid](#) that displays data from a data source.
2. In the [DataGrid](#) element, add a [RowDetailsTemplate](#) element.
3. Create a [DataTemplate](#) that defines the appearance of the row details section.

The following XAML shows the [DataGrid](#) and how to define the [RowDetailsTemplate](#) inline. The [DataGrid](#) displays three values in each row and three more values when the row is selected.

```

<Window x:Class="WpfApplication1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525"
        Loaded="Window_Loaded">
    <Grid>
        <DataGrid Name="dataGrid1" IsReadOnly="True" AutoGenerateColumns="False" >
            <DataGrid.Columns>
                <DataGridTextColumn Header="Company Name" Binding="{Binding CompanyName}">
                </DataGridTextColumn>
                <DataGridTextColumn Header="Contact First Name" Binding="{Binding FirstName}">
                </DataGridTextColumn>
                <DataGridTextColumn Header="Contact Last Name" Binding="{Binding LastName}">
                </DataGridTextColumn>
            </DataGrid.Columns>
            <DataGrid.RowDetailsTemplate>
                <DataTemplate>
                    <Border BorderThickness="0" Background="BlanchedAlmond" Padding="10">
                        <StackPanel Orientation="Vertical">
                            <StackPanel Orientation="Horizontal">
                                <TextBlock FontSize="12" Text="Email: " VerticalAlignment="Center" />
                                <TextBlock FontSize="16" Foreground="MidnightBlue" Text="{Binding EmailAddress}" VerticalAlignment="Center" />
                            </StackPanel>
                            <StackPanel Orientation="Horizontal">
                                <TextBlock FontSize="12" Text="Phone: " VerticalAlignment="Center" />
                                <TextBlock FontSize="16" Foreground="MidnightBlue" Text="{Binding Phone}" VerticalAlignment="Center" />
                            </StackPanel>
                            <StackPanel Orientation="Horizontal">
                                <TextBlock FontSize="12" Text="Sales Person: " VerticalAlignment="Center" />
                                <TextBlock FontSize="16" Foreground="MidnightBlue" Text="{Binding SalesPerson}" VerticalAlignment="Center" />
                            </StackPanel>
                        </StackPanel>
                    </Border>
                </DataTemplate>
            </DataGrid.RowDetailsTemplate>
        </DataGrid>
    </Grid>
</Window>

```

The following code shows the query that is used to select the data that is displayed in the [DataGrid](#). In this example, the query selects data from an entity that contains customer information.

```

private void Window_Loaded(object sender, RoutedEventArgs e)
{
    AdventureWorksLT2008Entities advenWorksEntities = new AdventureWorksLT2008Entities();

    ObjectQuery<Customer> customers = advenWorksEntities.Customers;

    var query =
        from customer in customers
        orderby customer.CompanyName
        select new
        {
            customer.LastName,
            customer.FirstName,
            customer.CompanyName,
            customer.Title,
            customer.EmailAddress,
            customer.Phone,
            customer.SalesPerson
        };
    dataGrid1.ItemsSource = query.ToList();
}

```

```

Dim advenWorksEntities As AdventureWorksLT2008Entities = New AdventureWorksLT2008Entities

Private Sub Window_Loaded(ByVal sender As System.Object, ByVal e As System.Windows.RoutedEventArgs)
Handles MyBase.Loaded
    Dim customers As ObjectQuery(Of Customer) = advenWorksEntities.Customers
    Dim query = _
        From customer In customers _
        Order By customer.CompanyName _
        Select _
            customer.LastName, _
            customer.FirstName, _
            customer.CompanyName, _
            customer.Title, _
            customer.EmailAddress, _
            customer.Phone, _
            customer.SalesPerson

    dataGrid1.ItemsSource = query.ToList()
End Sub

```

## To display row details by using a resource

1. Create a [DataGrid](#) that displays data from a data source.
2. Add a [Resources](#) element to the root element, such as a [Window](#) control or a [Page](#) control, or add a [Resources](#) element to the [Application](#) class in the App.xaml (or Application.xaml) file.
3. In the resources element, create a [DataTemplate](#) that defines the appearance of the row details section.

The following XAML shows the [RowDetailsTemplate](#) defined in the [Application](#) class.

```

<Application.Resources>
    <DataTemplate x:Key="CustomerDetail">
        <Border BorderThickness="0" Background="BlanchedAlmond" Padding="10">
            <StackPanel Orientation="Vertical">
                <StackPanel Orientation="Horizontal">
                    <TextBlock FontSize="12" Text="Email: " VerticalAlignment="Center" />
                    <TextBlock FontSize="16" Foreground="MidnightBlue" Text="{Binding EmailAddress}" VerticalAlignment="Center" />
                </StackPanel>
                <StackPanel Orientation="Horizontal">
                    <TextBlock FontSize="12" Text="Phone: " VerticalAlignment="Center" />
                    <TextBlock FontSize="16" Foreground="MidnightBlue" Text="{Binding Phone}" VerticalAlignment="Center" />
                </StackPanel>
                <StackPanel Orientation="Horizontal">
                    <TextBlock FontSize="12" Text="Sales Person: " VerticalAlignment="Center" />
                    <TextBlock FontSize="16" Foreground="MidnightBlue" Text="{Binding SalesPerson}" VerticalAlignment="Center" />
                </StackPanel>
            </StackPanel>
        </Border>
    </DataTemplate>
</Application.Resources>

```

4. On the [DataTemplate](#), set the [x:Key Directive](#) to a value that uniquely identifies the data template.
5. In the [DataGrid](#) element, set the [RowDetailsTemplate](#) property to the resource defined in the previous steps. Assign the resource as a static resource.

The following XAML shows the [RowDetailsTemplate](#) property set to the resource from the previous example.

```

<DataGrid Name="dataGrid1" IsReadOnly="True" AutoGenerateColumns="False" RowDetailsTemplate="{StaticResource CustomerDetail}">
    <DataGrid.Columns>
        <DataGridTextColumn Header="Company Name" Binding="{Binding CompanyName}" />
        <DataGridTextColumn Header="Contact First Name" Binding="{Binding FirstName}" />
        <DataGridTextColumn Header="Contact Last Name" Binding="{Binding LastName}" />
    </DataGrid.Columns>
</DataGrid>

```

### To set visibility and prevent horizontal scrolling for row details

1. If needed, set the [RowDetailsVisibilityMode](#) property to a [DataGridRowDetailsVisibilityMode](#) value.

By default, the value is set to [VisibleWhenSelected](#). You can set it to [Visible](#) to show the details for all of the rows or [Collapsed](#) to hide the details for all rows.

2. If needed, set the [AreRowDetailsFrozen](#) property to `true` to prevent the row details section from scrolling horizontally.

# How to: Group, sort, and filter data in the DataGrid control

13 minutes to read • [Edit Online](#)

It is often useful to view data in a [DataGrid](#) in different ways by grouping, sorting, and filtering the data. To group, sort, and filter the data in a [DataGrid](#), you bind it to a [CollectionView](#) that supports these functions. You can then work with the data in the [CollectionView](#) without affecting the underlying source data. The changes in the collection view are reflected in the [DataGrid](#) user interface (UI).

The [CollectionView](#) class provides grouping and sorting functionality for a data source that implements the [IEnumerable](#) interface. The [CollectionViewSource](#) class enables you to set the properties of a [CollectionView](#) from XAML.

In this example, a collection of [Task](#) objects is bound to a [CollectionViewSource](#). The [CollectionViewSource](#) is used as the [ItemsSource](#) for the [DataGrid](#). Grouping, sorting, and filtering are performed on the [CollectionViewSource](#) and are displayed in the [DataGrid](#) UI.

ProjectName	TaskName	DueDate	Complete
Project 1	4		
Active	2		
Project 1	Task 3	3/7/2010 10:51:13 AM	<input type="checkbox"/>
Project 1	Task 9	3/13/2010 10:51:13 AM	<input type="checkbox"/>
Complete	2		
Project 1	Task 6	3/10/2010 10:51:13 AM	<input checked="" type="checkbox"/>
Project 1	Task 12	3/16/2010 10:51:13 AM	<input checked="" type="checkbox"/>
Project 2	5		
Active	3		
Project 2	Task 1	3/5/2010 10:51:13 AM	<input type="checkbox"/>
Project 2	Task 7	3/11/2010 10:51:13 AM	<input type="checkbox"/>
Project 2	Task 13	3/17/2010 10:51:13 AM	<input type="checkbox"/>
Complete	2		
Project 2	Task 4	3/8/2010 10:51:13 AM	<input checked="" type="checkbox"/>
Project 2	Task 10	3/14/2010 10:51:13 AM	<input checked="" type="checkbox"/>
Project 3	5		
Active	2		
Project 3	Task 5	3/9/2010 10:51:13 AM	<input type="checkbox"/>
Project 3	Task 11	3/15/2010 10:51:13 AM	<input type="checkbox"/>
Project 3	Task 14	3/18/2010 10:51:13 AM	<input checked="" type="checkbox"/>
Complete	3		
Project 3	Task 2	3/6/2010 10:51:13 AM	<input checked="" type="checkbox"/>
Project 3	Task 8	3/12/2010 10:51:13 AM	<input checked="" type="checkbox"/>

Grouped data in a DataGrid

## Using a CollectionViewSource as an ItemsSource

To group, sort, and filter data in a [DataGrid](#) control, you bind the [DataGrid](#) to a [CollectionView](#) that supports these functions. In this example, the [DataGrid](#) is bound to a [CollectionViewSource](#) that provides these functions for a [List<T>](#) of [Task](#) objects.

### To bind a DataGrid to a CollectionViewSource

1. Create a data collection that implements the [IEnumerable](#) interface.

If you use `List<T>` to create your collection, you should create a new class that inherits from `List<T>` instead of instantiating an instance of `List<T>`. This enables you to data bind to the collection in XAML.

#### NOTE

The objects in the collection must implement the `INotifyPropertyChanged` changed interface and the `IEditableObject` interface in order for the `DataGrid` to respond correctly to property changes and edits. For more information, see [Implement Property Change Notification](#).

```
// Requires using System.Collections.ObjectModel;
public class Tasks : ObservableCollection<Task>
{
    // Creating the Tasks collection in this way enables data binding from XAML.
}
```

```
' Requires Imports System.Collections.ObjectModel
Public Class Tasks
    Inherits ObservableCollection(Of Task)
    ' Creating the Tasks collection in this way enables data binding from XAML.
End Class
```

2. In XAML, create an instance of the collection class and set the `x:Key` Directive.
3. In XAML, create an instance of the `CollectionViewSource` class, set the `x:Key` Directive, and set the instance of your collection class as the `Source`.

```
<Window.Resources>
    <local:Tasks x:Key="tasks" />
    <CollectionViewSource x:Key="cvsTasks" Source="{StaticResource tasks}"
        Filter="CollectionViewSource_Filter">
    </CollectionViewSource>
</Window.Resources>
```

4. Create an instance of the `DataGrid` class, and set the `ItemsSource` property to the `CollectionViewSource`.

```
<DataGrid x:Name="dataGrid1"
    ItemsSource="{Binding Source={StaticResource cvsTasks}}"
    CanUserAddRows="False">
```

5. To access the `CollectionViewSource` from your code, use the `GetDefaultView` method to get a reference to the `CollectionViewSource`.

```
ICollectionView cvTasks = CollectionViewSource.GetDefaultView(dataGrid1.ItemsSource);
```

```
Dim cvTasks As ICollectionView = CollectionViewSource.GetDefaultView(dataGrid1.ItemsSource)
```

## Grouping items in a DataGrid

To specify how items are grouped in a `DataGrid`, you use the `PropertyGroupDescription` type to group the items in the source view.

### To group items in a DataGrid using XAML

1. Create a [PropertyGroupDescription](#) that specifies the property to group by. You can specify the property in XAML or in code.
  - a. In XAML, set the [PropertyName](#) to the name of the property to group by.
  - b. In code, pass the name of the property to group by to the constructor.
2. Add the [PropertyGroupDescription](#) to the [CollectionViewSource.GroupDescriptions](#) collection.
3. Add additional instances of [PropertyGroupDescription](#) to the [GroupDescriptions](#) collection to add more levels of grouping.

```
<CollectionViewSource.GroupDescriptions>
  <PropertyGroupDescription PropertyName="ProjectName"/>
  <PropertyGroupDescription PropertyName="Complete"/>
</CollectionViewSource.GroupDescriptions>
```

```
ICollectionView cvTasks = CollectionViewSource.GetDefaultView(dataGrid1.ItemsSource);
if (cvTasks != null && cvTasks.CanGroup == true)
{
    cvTasks.GroupDescriptions.Clear();
    cvTasks.GroupDescriptions.Add(new PropertyGroupDescription("ProjectName"));
    cvTasks.GroupDescriptions.Add(new PropertyGroupDescription("Complete"));
}
```

```
Dim cvTasks As ICollectionView = CollectionViewSource.GetDefaultView(dataGrid1.ItemsSource)
If cvTasks IsNot Nothing And cvTasks.CanGroup = True Then
    cvTasks.GroupDescriptions.Clear()
    cvTasks.GroupDescriptions.Add(New PropertyGroupDescription("ProjectName"))
    cvTasks.GroupDescriptions.Add(New PropertyGroupDescription("Complete"))
End If
```

4. To remove a group, remove the [PropertyGroupDescription](#) from the [GroupDescriptions](#) collection.
5. To remove all groups, call the [Clear](#) method of the [GroupDescriptions](#) collection.

```
ICollectionView cvTasks = CollectionViewSource.GetDefaultView(dataGrid1.ItemsSource);
if (cvTasks != null)
{
    cvTasks.GroupDescriptions.Clear();
}
```

```
Dim cvTasks As ICollectionView = CollectionViewSource.GetDefaultView(dataGrid1.ItemsSource)
If cvTasks IsNot Nothing Then
    cvTasks.GroupDescriptions.Clear()
End If
```

When items are grouped in the [DataGrid](#), you can define a [GroupStyle](#) that specifies the appearance of each group. You apply the [GroupStyle](#) by adding it to the [GroupStyle](#) collection of the DataGrid. If you have multiple levels of grouping, you can apply different styles to each group level. Styles are applied in the order in which they are defined. For example, if you define two styles, the first will be applied to top level row groups. The second style will be applied to all row groups at the second level and lower. The [DataContext](#) of the [GroupStyle](#) is the [CollectionViewGroup](#) that the group represents.

### To change the appearance of row group headers

1. Create a [GroupStyle](#) that defines the appearance of the row group.

- Put the [GroupStyle](#) inside the `<DataGrid.GroupStyle>` tags.

```

<DataGrid.GroupStyle>
    <!-- Style for groups at top level. -->
    <GroupStyle>
        <GroupStyle.ContainerStyle>
            <Style TargetType="{x:Type GroupItem}">
                <Setter Property="Margin" Value="0,0,0,5"/>
                <Setter Property="Template">
                    <Setter.Value>
                        <ControlTemplate TargetType="{x:Type GroupItem}">
                            <Expander IsExpanded="True" Background="#FF112255" BorderBrush="#FF002255"
                            Foreground="#FFFFFF" BorderThickness="1,1,1,5">
                                <Expander.Header>
                                    <DockPanel>
                                        <TextBlock FontWeight="Bold" Text="{Binding Path=Name}"
                                        Margin="5,0,0,0" Width="100"/>
                                        <TextBlock FontWeight="Bold" Text="{Binding Path=ItemCount}"/>
                                    </DockPanel>
                                </Expander.Header>
                                <Expander.Content>
                                    <ItemsPresenter />
                                </Expander.Content>
                            </Expander>
                        </ControlTemplate>
                    </Setter.Value>
                </Setter>
            </Style>
        </GroupStyle.ContainerStyle>
    </GroupStyle>
    <!-- Style for groups under the top level. -->
    <GroupStyle>
        <GroupStyle.HeaderTemplate>
            <DataTemplate>
                <DockPanel Background="LightBlue">
                    <TextBlock Text="{Binding Path=Name, Converter={StaticResource completeConverter}}"
                    Foreground="Blue" Margin="30,0,0,0" Width="100"/>
                    <TextBlock Text="{Binding Path=ItemCount}" Foreground="Blue"/>
                </DockPanel>
            </DataTemplate>
        </GroupStyle.HeaderTemplate>
    </GroupStyle>
</DataGrid.GroupStyle>

```

## Sorting items in a DataGrid

To specify how items are sorted in a [DataGrid](#), you use the [SortDescription](#) type to sort the items in the source view.

### To sort items in a DataGrid

- Create a [SortDescription](#) that specifies the property to sort by. You can specify the property in XAML or in code.
  - In XAML, set the [PropertyName](#) to the name of the property to sort by.
  - In code, pass the name of the property to sort by and the [ListSortDirection](#) to the constructor.
- Add the [SortDescription](#) to the [CollectionViewSource.SortDescriptions](#) collection.
- Add additional instances of [SortDescription](#) to the [SortDescriptions](#) collection to sort by additional properties.

```

<CollectionViewSource.SortDescriptions>
    <!-- Requires 'xmlns:scm="clr-namespace:System.ComponentModel;assembly=WindowsBase"' declaration. -->
    <scm:SortDescription PropertyName="ProjectName" />
    <scm:SortDescription PropertyName="Complete" />
    <scm:SortDescription PropertyName="DueDate" />
</CollectionViewSource.SortDescriptions>

```

```

// Requires using System.ComponentModel;
ICollectionView cvTasks = CollectionViewSource.GetDefaultView(dataGrid1.ItemsSource);
if (cvTasks != null && cvTasks.CanSort == true)
{
    cvTasks.SortDescriptions.Clear();
    cvTasks.SortDescriptions.Add(new SortDescription("ProjectName", ListSortDirection.Ascending));
    cvTasks.SortDescriptions.Add(new SortDescription("Complete", ListSortDirection.Ascending));
    cvTasks.SortDescriptions.Add(new SortDescription("DueDate", ListSortDirection.Ascending));
}

```

```

Dim cvTasks As ICollectionView = CollectionViewSource.GetDefaultView(dataGrid1.ItemsSource)
If cvTasks IsNot Nothing And cvTasks.CanSort = True Then
    cvTasks.SortDescriptions.Clear()
    cvTasks.SortDescriptions.Add(New SortDescription("ProjectName", ListSortDirection.Ascending))
    cvTasks.SortDescriptions.Add(New SortDescription("Complete", ListSortDirection.Ascending))
    cvTasks.SortDescriptions.Add(New SortDescription("DueDate", ListSortDirection.Ascending))
End If

```

## Filtering items in a DataGrid

To filter items in a [DataGrid](#) using a [CollectionViewSource](#), you provide the filtering logic in the handler for the [CollectionViewSource.Filter](#) event.

### To filter items in a DataGrid

1. Add a handler for the [CollectionViewSource.Filter](#) event.
2. In the [Filter](#) event handler, define the filtering logic.

The filter will be applied every time the view is refreshed.

```

<CollectionViewSource x:Key="cvsTasks" Source="{StaticResource tasks}"
    Filter="CollectionViewSource_Filter">

```

```

private void CollectionViewSource_Filter(object sender, FilterEventArgs e)
{
    Task t = e.Item as Task;
    if (t != null)
        // If filter is turned on, filter completed items.
        {
            if (this.cbCompleteFilter.IsChecked == true && t.Complete == true)
                e.Accepted = false;
            else
                e.Accepted = true;
        }
}

```

```

Private Sub CollectionViewSource_Filter(ByVal sender As System.Object, ByVal e As
System.Windows.Data.FilterEventArgs)
    Dim t As Task = e.Item
    If t IsNot Nothing Then
        ' If filter is turned on, filter completed items.
        If Me.cbCompleteFilter.IsChecked = True And t.Complete = True Then
            e.Accepted = False
        Else
            e.Accepted = True
        End If
    End If
End Sub

```

Alternatively, you can filter items in a [DataGrid](#) by creating a method that provides the filtering logic and setting the [CollectionView.Filter](#) property to apply the filter. To see an example of this method, see [Filter Data in a View](#).

## Example

The following example demonstrates grouping, sorting, and filtering [Task](#) data in a [CollectionViewSource](#) and displaying the grouped, sorted, and filtered [Task](#) data in a [DataGrid](#). The [CollectionViewSource](#) is used as the [ItemsSource](#) for the [DataGrid](#). Grouping, sorting, and filtering are performed on the [CollectionViewSource](#) and are displayed in the [DataGrid](#) UI.

To test this example, you will need to adjust the `DGGroupSortFilterExample` name to match your project name. If you are using Visual Basic, you will need to change the class name for [Window](#) to the following.

```

<Window x:Class="MainWindow"

<Window x:Class="DGGroupSortFilterExample.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:DGGroupSortFilterExample"
    xmlns:scm="clr-namespace:System.ComponentModel;assembly=WindowsBase"
    Title="Group, Sort, and Filter Example" Height="575" Width="525">
<Window.Resources>
    <local:CompleteConverter x:Key="completeConverter" />
    <local:Tasks x:Key="tasks" />
    <CollectionViewSource x:Key="cvsTasks" Source="{StaticResource tasks}">
        Filter="CollectionViewSource_Filter">
        <CollectionViewSource.SortDescriptions>
            <!-- Requires 'xmlns:scm="clr-namespace:System.ComponentModel;assembly=WindowsBase"' declaration. -->
            <scm:SortDescription PropertyName="ProjectName"/>
            <scm:SortDescription PropertyName="Complete" />
            <scm:SortDescription PropertyName="DueDate" />
        </CollectionViewSource.SortDescriptions>
        <CollectionViewSource.GroupDescriptions>
            <PropertyGroupDescription PropertyName="ProjectName"/>
            <PropertyGroupDescription PropertyName="Complete"/>
        </CollectionViewSource.GroupDescriptions>
    </CollectionViewSource>
</Window.Resources>
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition Height="30" />
    </Grid.RowDefinitions>
    <DataGrid x:Name="dataGrid1"
        ItemsSource="{Binding Source={StaticResource cvsTasks}}"
        CanUserAddRows="False">
        <DataGrid.GroupStyle>
            <!-- Style for groups at top level. -->
            <GroupStyle>

```

```

<GroupStyle.ContainerStyle>
    <Style TargetType="{x:Type GroupItem}">
        <Setter Property="Margin" Value="0,0,0,5"/>
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="{x:Type GroupItem}">
                    <Expander IsExpanded="True" Background="#FF112255"
BorderBrush="#FF002255" Foreground="#FFEEEEEE" BorderThickness="1,1,1,5">
                        <Expander.Header>
                            <DockPanel>
                                <TextBlock FontWeight="Bold" Text="{Binding Path=Name}"
Margin="5,0,0,0" Width="100"/>
                                <TextBlock FontWeight="Bold" Text="{Binding
Path=ItemCount}"/>
                            </DockPanel>
                        </Expander.Header>
                        <Expander.Content>
                            <ItemsPresenter />
                        </Expander.Content>
                    </Expander>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
</GroupStyle.ContainerStyle>
</GroupStyle>
<!-- Style for groups under the top level. -->
<GroupStyle>
    <GroupStyle.HeaderTemplate>
        <DataTemplate>
            <DockPanel Background="LightBlue">
                <TextBlock Text="{Binding Path=Name, Converter={StaticResource
completeConverter}}" Foreground="Blue" Margin="30,0,0,0" Width="100"/>
                <TextBlock Text="{Binding Path=ItemCount}" Foreground="Blue"/>
            </DockPanel>
        </DataTemplate>
    </GroupStyle.HeaderTemplate>
</GroupStyle>
</DataGrid.GroupStyle>
<DataGrid.RowStyle>
    <Style TargetType="DataGridRow">
        <Setter Property="Foreground" Value="Black" />
        <Setter Property="Background" Value="White" />
    </Style>
</DataGrid.RowStyle>
</DataGrid>
<StackPanel Orientation="Horizontal" Grid.Row="1">
    <TextBlock Text=" Filter completed items " VerticalAlignment="Center" />
    <CheckBox x:Name="cbCompleteFilter" VerticalAlignment="Center"
Checked="CompleteFilter_Changed" Unchecked="CompleteFilter_Changed" />
    <Button Content="Remove Groups" Margin="10,2,2,2" Click="UngroupButton_Click" />
    <Button Content="Group by Project/Status" Margin="2" Click="GroupButton_Click" />
</StackPanel>
</Grid>
</Window>

```

```

using System;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Windows;
using System.Windows.Data;

namespace DGGroupSortFilterExample
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>

```

```

public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();

        // Get a reference to the tasks collection.
        Tasks _tasks = (Tasks)this.Resources["tasks"];

        // Generate some task data and add it to the task list.
        for (int i = 1; i <= 14; i++)
        {
            _tasks.Add(new Task()
            {
                ProjectName = "Project " + ((i % 3) + 1).ToString(),
                TaskName = "Task " + i.ToString(),
                DueDate = DateTime.Now.AddDays(i),
                Complete = (i % 2 == 0)
            });
        }
    }

    private void UngroupButton_Click(object sender, RoutedEventArgs e)
    {
        ICollectionView cvTasks = CollectionViewSource.GetDefaultView(dataGrid1.ItemsSource);
        if (cvTasks != null)
        {
            cvTasks.GroupDescriptions.Clear();
        }
    }

    private void GroupButton_Click(object sender, RoutedEventArgs e)
    {
        ICollectionView cvTasks = CollectionViewSource.GetDefaultView(dataGrid1.ItemsSource);
        if (cvTasks != null & cvTasks.CanGroup == true)
        {
            cvTasks.GroupDescriptions.Clear();
            cvTasks.GroupDescriptions.Add(new PropertyGroupDescription("ProjectName"));
            cvTasks.GroupDescriptions.Add(new PropertyGroupDescription("Complete"));
        }
    }

    private void CompleteFilter_Changed(object sender, RoutedEventArgs e)
    {
        // Refresh the view to apply filters.
        CollectionViewSource.GetDefaultView(dataGrid1.ItemsSource).Refresh();
    }

    private void CollectionViewSource_Filter(object sender, FilterEventArgs e)
    {
        Task t = e.Item as Task;
        if (t != null)
        // If filter is turned on, filter completed items.
        {
            if (this.cbCompleteFilter.IsChecked == true && t.Complete == true)
                e.Accepted = false;
            else
                e.Accepted = true;
        }
    }

    [ValueConversion(typeof(Boolean), typeof(String))]
    public class CompleteConverter : IValueConverter
    {
        // This converter changes the value of a Tasks Complete status from true/false to a string value of
        // "Complete"/"Active" for use in the row group header.
        public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)

```

```

{
    bool complete = (bool)value;
    if (complete)
        return "Complete";
    else
        return "Active";
}

public object ConvertBack(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
{
    string strComplete = (string)value;
    if (strComplete == "Complete")
        return true;
    else
        return false;
}
}

// Task Class
// Requires using System.ComponentModel;
public class Task : INotifyPropertyChanged, IEditableObject
{
    // The Task class implements INotifyPropertyChanged and IEditableObject
    // so that the datagrid can properly respond to changes to the
    // data collection and edits made in the DataGrid.

    // Private task data.
    private string m_ProjectName = string.Empty;
    private string m_TaskName = string.Empty;
    private DateTime m_DueDate = DateTime.Now;
    private bool m_Complete = false;

    // Data for undoing canceled edits.
    private Task temp_Task = null;
    private bool m_Editing = false;

    // Public properties.
    public string ProjectName
    {
        get { return this.m_ProjectName; }
        set
        {
            if (value != this.m_ProjectName)
            {
                this.m_ProjectName = value;
                NotifyPropertyChanged("ProjectName");
            }
        }
    }

    public string TaskName
    {
        get { return this.m_TaskName; }
        set
        {
            if (value != this.m_TaskName)
            {
                this.m_TaskName = value;
                NotifyPropertyChanged("TaskName");
            }
        }
    }

    public DateTime DueDate
    {
        get { return this.m_DueDate; }
        set
        {

```

```

        if (value != this.m_DueDate)
    {
        this.m_DueDate = value;
        NotifyPropertyChanged("DueDate");
    }
}

public bool Complete
{
    get { return this.m_Complete; }
    set
    {
        if (value != this.m_Complete)
        {
            this.m_Complete = value;
            NotifyPropertyChanged("Complete");
        }
    }
}

// Implement INotifyPropertyChanged interface.
public event PropertyChangedEventHandler PropertyChanged;

private void NotifyPropertyChanged(string propertyName)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }
}

// Implement IEditableObject interface.
public void BeginEdit()
{
    if (m_Editing == false)
    {
        temp_Task = this.MemberwiseClone() as Task;
        m_Editing = true;
    }
}

public void CancelEdit()
{
    if (m_Editing == true)
    {
        this.ProjectName = temp_Task.ProjectName;
        this.TaskName = temp_Task.TaskName;
        this.DueDate = temp_Task.DueDate;
        this.Complete = temp_Task.Complete;
        m_Editing = false;
    }
}

public void EndEdit()
{
    if (m_Editing == true)
    {
        temp_Task = null;
        m_Editing = false;
    }
}

// Requires using System.Collections.ObjectModel;
public class Tasks : ObservableCollection<Task>
{
    // Creating the Tasks collection in this way enables data binding from XAML.
}

```

```

Imports System.ComponentModel
Imports System.Collections.ObjectModel

Class MainWindow
    Public Sub New()
        InitializeComponent()

        ' Get a reference to the tasks collection.
        Dim _tasks As Tasks = Me.Resources("tasks")

        ' Generate some task data and add it to the task list.
        For index = 1 To 14
            _tasks.Add(New Task() With _
                {.ProjectName = "Project " & ((index Mod 3) + 1).ToString(), _
                 .TaskName = "Task " & index.ToString(), _
                 .DueDate = Date.Now.AddDays(index), _
                 .Complete = (index Mod 2 = 0) _})
        Next
    End Sub

    Private Sub UngroupButton_Click(ByVal sender As System.Object, ByVal e As System.Windows.RoutedEventArgs)
        Dim cvTasks As ICollectionView = CollectionViewSource.GetDefaultView(dataGrid1.ItemsSource)
        If cvTasks IsNot Nothing Then
            cvTasks.GroupDescriptions.Clear()
        End If
    End Sub

    Private Sub GroupButton_Click(ByVal sender As System.Object, ByVal e As System.Windows.RoutedEventArgs)
        Dim cvTasks As ICollectionView = CollectionViewSource.GetDefaultView(dataGrid1.ItemsSource)
        If cvTasks IsNot Nothing And cvTasks.CanGroup = True Then
            cvTasks.GroupDescriptions.Clear()
            cvTasks.GroupDescriptions.Add(New PropertyGroupDescription("ProjectName"))
            cvTasks.GroupDescriptions.Add(New PropertyGroupDescription("Complete"))
        End If
    End Sub

    Private Sub CompleteFilter_Changed(ByVal sender As System.Object, ByVal e As System.Windows.RoutedEventArgs)
        ' Refresh the view to apply filters.
        CollectionViewSource.GetDefaultView(dataGrid1.ItemsSource).Refresh()
    End Sub

    Private Sub CollectionViewSource_Filter(ByVal sender As System.Object, ByVal e As System.Windows.Data.FilterEventArgs)
        Dim t As Task = e.Item
        If t IsNot Nothing Then
            ' If filter is turned on, filter completed items.
            If Me.cbCompleteFilter.IsChecked = True And t.Complete = True Then
                e.Accepted = False
            Else
                e.Accepted = True
            End If
        End If
    End Sub
End Class

Public Class CompleteConverter
    Implements IValueConverter
    ' This converter changes the value of a Tasks Complete status from true/false to a string value of
    ' "Complete"/"Active" for use in the row group header.
    Public Function Convert(ByVal value As Object, ByVal targetType As System.Type, ByVal parameter As Object,
    ByVal culture As System.Globalization.CultureInfo) As Object Implements
        System.Windows.Data.IValueConverter.Convert
        Dim complete As Boolean = value
        If complete = True Then

```

```

    End If
End Function

Public Function ConvertBack1(ByVal value As Object, ByVal targetType As System.Type, ByVal parameter As
Object, ByVal culture As System.Globalization.CultureInfo) As Object Implements
System.Windows.Data.IValueConverter.ConvertBack
    Dim strComplete As String = value
    If strComplete = "Complete" Then
        Return True
    Else
        Return False
    End If
End Function
End Class

' Task class
' Requires Imports System.ComponentModel
Public Class Task
    Implements INotifyPropertyChanged, IEditableObject
    ' The Task class implements INotifyPropertyChanged and IEditableObject
    ' so that the datagrid can properly respond to changes to the
    ' data collection and edits made in the DataGrid.

    ' Private task data.
    Private m_ProjectName As String = String.Empty
    Private m_TaskName As String = String.Empty
    Private m_DueDate As DateTime = Date.Now
    Private m_Complete As Boolean = False

    ' Data for undoing canceled edits.
    Private temp_Task As Task = Nothing
    Private m_Editing As Boolean = False

    ' Public properties.
    Public Property ProjectName() As String
        Get
            Return Me.m_ProjectName
        End Get
        Set(ByVal value As String)
            If Not value = Me.m_ProjectName Then
                Me.m_ProjectName = value
                NotifyPropertyChanged("ProjectName")
            End If
        End Set
    End Property

    Public Property TaskName() As String
        Get
            Return Me.m_TaskName
        End Get
        Set(ByVal value As String)
            If Not value = Me.m_TaskName Then
                Me.m_TaskName = value
                NotifyPropertyChanged("TaskName")
            End If
        End Set
    End Property

    Public Property DueDate() As Date
        Get
            Return Me.m_DueDate
        End Get
        Set(ByVal value As Date)
            If Not value = Me.m_DueDate Then
                Me.m_DueDate = value
                NotifyPropertyChanged("DueDate")
            End If
        End Set
    End Property

```

```

        NOTIFYPROPERTYCHANGED(DueDate)
    End If
End Set
End Property

Public Property Complete() As Boolean
Get
    Return Me.m_Complete
End Get
Set(ByVal value As Boolean)
    If Not value = Me.m_Complete Then
        Me.m_Complete = value
        NotifyPropertyChanged("Complete")
    End If
End Set
End Property

' Implement INotifyPropertyChanged interface.
Public Event PropertyChanged As PropertyChangedEventHandler Implements
INotifyPropertyChanged.PropertyChanged

Public Sub NotifyPropertyChanged(ByVal propertyName As String)
    RaiseEvent PropertyChanged(Me, New PropertyChangedEventArgs(propertyName))
End Sub

' Implement IEditableObject interface.
Public Sub BeginEdit() Implements IEditableObject.BeginEdit
    If Not Me.m_Editing Then
        Me.temp_Task = Me.MemberwiseClone()
        Me.m_Editing = True
    End If
End Sub

Public Sub CancelEdit() Implements IEditableObject.CancelEdit
    If m_Editing = True Then
        Me.ProjectName = Me.temp_Task.ProjectName
        Me.TaskName = Me.temp_Task.TaskName
        Me.DueDate = Me.temp_Task.DueDate
        Me.Complete = Me.temp_Task.Complete
        Me.m_Editing = False
    End If
End Sub

Public Sub EndEdit() Implements IEditableObject.EndEdit
    If m_Editing = True Then
        Me.temp_Task = Nothing
        Me.m_Editing = False
    End If
End Sub
End Class

' Requires Imports System.Collections.ObjectModel
Public Class Tasks
    Inherits ObservableCollection(Of Task)
    ' Creating the Tasks collection in this way enables data binding from XAML.
End Class

```

## See also

- [Data Binding Overview](#)
- [Create and Bind to an ObservableCollection](#)
- [Filter Data in a View](#)
- [Sort Data in a View](#)
- [Sort and Group Data Using a View in XAML](#)

# How to: Implement Validation with the DataGrid Control

9 minutes to read • [Edit Online](#)

The [DataGrid](#) control enables you to perform validation at both the cell and row level. With cell-level validation, you validate individual properties of a bound data object when a user updates a value. With row-level validation, you validate entire data objects when a user commits changes to a row. You can also provide customized visual feedback for validation errors, or use the default visual feedback that the [DataGrid](#) control provides.

The following procedures describe how to apply validation rules to [DataGrid](#) bindings and customize the visual feedback.

## To validate individual cell values

- Specify one or more validation rules on the binding used with a column. This is similar to validating data in simple controls, as described in [Data Binding Overview](#).

The following example shows a [DataGrid](#) control with four columns bound to different properties of a business object. Three of the columns specify the [ExceptionValidationRule](#) by setting the [ValidatesOnExceptions](#) property to `true`.

```
<Grid>

    <Grid.Resources>
        <local:Courses x:Key="courses"/>
    </Grid.Resources>

    <DataGrid Name="dataGrid1" FontSize="20"
        ItemsSource="{StaticResource courses}"
        AutoGenerateColumns="False">
        <DataGrid.Columns>
            <DataGridTextColumn Header="Course Name"
                Binding="{Binding Name, TargetNullValue=(enter a course name)}/>
            <DataGridTextColumn Header="Course ID"
                Binding="{Binding Id, ValidatesOnExceptions=True}"/>
            <DataGridTextColumn Header="Start Date"
                Binding="{Binding StartDate, ValidatesOnExceptions=True,
                    StringFormat=d}"/>
            <DataGridTextColumn Header="End Date"
                Binding="{Binding EndDate, ValidatesOnExceptions=True,
                    StringFormat=d}"/>
        </DataGrid.Columns>
    </DataGrid>

</Grid>
```

When a user enters an invalid value (such as a non-integer in the Course ID column), a red border appears around the cell. You can change this default validation feedback as described in the following procedure.

## To customize cell validation feedback

- Set the column's [EditingStyle](#) property to a style appropriate for the column's editing control. Because the editing controls are created at run time, you cannot use the [Validation.ErrorTemplate](#) attached property like you would with simple controls.

The following example updates the previous example by adding an error style shared by the three columns

with validation rules. When a user enters an invalid value, the style changes the cell background color and adds a ToolTip. Note the use of a trigger to determine whether there is a validation error. This is required because there is currently no dedicated error template for cells.

```
<DataGrid.Resources>
<Style x:Key="errorStyle" TargetType="{x:Type TextBox}">
    <Setter Property="Padding" Value="-2"/>
    <Style.Triggers>
        <Trigger Property="Validation.HasError" Value="True">
            <Setter Property="Background" Value="Red"/>
            <Setter Property="ToolTip"
                Value="{Binding RelativeSource={RelativeSource Self},
                    Path=(Validation.Errors)[0].ErrorContent}"/>
        </Trigger>
    </Style.Triggers>
</Style>
</DataGrid.Resources>

<DataGrid.Columns>
    <DataGridTextColumn Header="Course Name"
        Binding="{Binding Name, TargetNullValue=(enter a course name)}"/>
    <DataGridTextColumn Header="Course ID"
        EditingElementStyle="{StaticResource errorStyle}"
        Binding="{Binding Id, ValidatesOnExceptions=True}"/>
    <DataGridTextColumn Header="Start Date"
        EditingElementStyle="{StaticResource errorStyle}"
        Binding="{Binding StartDate, ValidatesOnExceptions=True,
            StringFormat=d}"/>
    <DataGridTextColumn Header="End Date"
        EditingElementStyle="{StaticResource errorStyle}"
        Binding="{Binding EndDate, ValidatesOnExceptions=True,
            StringFormat=d}"/>
</DataGrid.Columns>
```

You can implement more extensive customization by replacing the [CellStyle](#) used by the column.

### To validate multiple values in a single row

1. Implement a [ValidationRule](#) subclass that checks multiple properties of the bound data object. In your [Validate](#) method implementation, cast the `value` parameter value to a [BindingGroup](#) instance. You can then access the data object through the [Items](#) property.

The following example demonstrates this process to validate whether the `StartDate` property value for a `Course` object is earlier than its `EndDate` property value.

```
public class CourseValidationRule : ValidationRule
{
    public override ValidationResult Validate(object value,
        System.Globalization.CultureInfo cultureInfo)
    {
        Course course = (value as BindingGroup).Items[0] as Course;
        if (course.StartDate > course.EndDate)
        {
            return new ValidationResult(false,
                "Start Date must be earlier than End Date.");
        }
        else
        {
            return ValidationResult.ValidResult;
        }
    }
}
```

```

Public Class CourseValidationRule
    Inherits ValidationRule

    Public Overrides Function Validate(ByVal value As Object, _
        ByVal cultureInfo As System.Globalization.CultureInfo) _ 
        As ValidationResult

        Dim course As Course = _
            CType(CType(value, BindingGroup).Items(0), Course)

        If course.StartDate > course.EndDate Then
            Return New ValidationResult(False, _
                "Start Date must be earlier than End Date.")
        Else
            Return ValidationResult.ValidResult
        End If
    End Function

End Class

```

2. Add the validation rule to the [DataGrid.RowValidationRules](#) collection. The [RowValidationRules](#) property provides direct access to the [ValidationRules](#) property of a [BindingGroup](#) instance that groups all the bindings used by the control.

The following example sets the [RowValidationRules](#) property in XAML. The [ValidationStep](#) property is set to [UpdatedValue](#) so that the validation occurs only after the bound data object is updated.

```

<DataGrid.RowValidationRules>
    <local:CourseValidationRule ValidationStep="UpdatedValue"/>
</DataGrid.RowValidationRules>

```

When a user specifies an end date that is earlier than the start date, a red exclamation mark (!) appears in the row header. You can change this default validation feedback as described in the following procedure.

### To customize row validation feedback

- Set the [DataGrid.RowValidationErrorTemplate](#) property. This property enables you to customize row validation feedback for individual [DataGrid](#) controls. You can also affect multiple controls by using an implicit row style to set the [DataGridRow.ValidationErrorTemplate](#) property.

The following example replaces the default row validation feedback with a more visible indicator. When a user enters an invalid value, a red circle with a white exclamation mark appears in the row header. This occurs for both row and cell validation errors. The associated error message is displayed in a ToolTip.

```

<DataGrid.RowValidationErrorTemplate>
    <ControlTemplate>
        <Grid Margin="0,-2,0,-2"
            ToolTip="{Binding RelativeSource={RelativeSource
                FindAncestor, AncestorType={x:Type DataGridRow}},
            Path=(Validation.Errors)[0].ErrorContent}">
            <Ellipse StrokeThickness="0" Fill="Red"
                Width="{TemplateBinding FontSize}"
                Height="{TemplateBinding FontSize}" />
            <TextBlock Text="!" FontSize="{TemplateBinding FontSize}"
                FontWeight="Bold" Foreground="White"
                HorizontalAlignment="Center" />
        </Grid>
    </ControlTemplate>
</DataGrid.RowValidationErrorTemplate>

```

## Example

The following example provides a complete demonstration for cell and row validation. The `Course` class provides a sample data object that implements `IEditableObject` to support transactions. The `DataGrid` control interacts with `IEditableObject` to enable users to revert changes by pressing ESC.

### NOTE

If you are using Visual Basic, in the first line of `MainWindow.xaml`, replace `x:Class="DataGridValidation.MainWindow"` with `x:Class="MainWindow"`.

To test the validation, try the following:

- In the Course ID column, enter a non-integer value.
- In the End Date column, enter a date that is earlier than the Start Date.
- Delete the value in Course ID, Start Date, or End Date.
- To undo an invalid cell value, put the cursor back in the cell and press the ESC key.
- To undo changes for an entire row when the current cell is in edit mode, press the ESC key twice.
- When a validation error occurs, move your mouse pointer over the indicator in the row header to see the associated error message.

```
using System;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;

namespace DataGridValidation
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            dataGrid1.InitializingNewItem += (sender, e) =>
            {
                Course newCourse = e.NewItem as Course;
                newCourse.StartDate = newCourse.EndDate = DateTime.Today;
            };
        }
    }

    public class Courses : ObservableCollection<Course>
    {
        public Courses()
        {
            this.Add(new Course
            {
                Name = "Learning WPF",
                Id = 1001,
                StartDate = new DateTime(2010, 1, 11),
                EndDate = new DateTime(2010, 1, 22)
            });
            this.Add(new Course
            {
                Name = "Learning Silverlight",
                Id = 1002,
                StartDate = new DateTime(2010, 1, 25)
            });
        }
    }
}
```

```

        StartDate = new DateTime(2010, 1, 25),
        EndDate = new DateTime(2010, 2, 5)
    });
    this.Add(new Course
    {
        Name = "Learning Expression Blend",
        Id = 1003,
        StartDate = new DateTime(2010, 2, 8),
        EndDate = new DateTime(2010, 2, 19)
    });
    this.Add(new Course
    {
        Name = "Learning LINQ",
        Id = 1004,
        StartDate = new DateTime(2010, 2, 22),
        EndDate = new DateTime(2010, 3, 5)
    });
}
}

public class Course : IEditableObject, INotifyPropertyChanged
{
    private string _name;
    public string Name
    {
        get
        {
            return _name;
        }
        set
        {
            if (_name == value) return;
            _name = value;
            OnPropertyChanged("Name");
        }
    }

    private int _number;
    public int Id
    {
        get
        {
            return _number;
        }
        set
        {
            if (_number == value) return;
            _number = value;
            OnPropertyChanged("Id");
        }
    }

    private DateTime _startDate;
    public DateTime StartDate
    {
        get
        {
            return _startDate;
        }
        set
        {
            if (_startDate == value) return;
            _startDate = value;
            OnPropertyChanged("StartDate");
        }
    }

    private DateTime _endDate;
    public DateTime EndDate
    
```

```

    {
        get
        {
            return _endDate;
        }
        set
        {
            if (_endDate == value) return;
            _endDate = value;
            OnPropertyChanged("EndDate");
        }
    }

#region IEditableObject

private Course backupCopy;
private bool inEdit;

public void BeginEdit()
{
    if (inEdit) return;
    inEdit = true;
    backupCopy = this.MemberwiseClone() as Course;
}

public void CancelEdit()
{
    if (!inEdit) return;
    inEdit = false;
    this.Name = backupCopy.Name;
    this.Id = backupCopy.Id;
    this.StartDate = backupCopy.StartDate;
    this.EndDate = backupCopy.EndDate;
}

public void EndEdit()
{
    if (!inEdit) return;
    inEdit = false;
    backupCopy = null;
}

#endregion

#region INotifyPropertyChanged

public event PropertyChangedEventHandler PropertyChanged;

private void OnPropertyChanged(string propertyName)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this,
            new PropertyChangedEventArgs(propertyName));
    }
}

#endregion

}

public class CourseValidationRule : ValidationRule
{
    public override ValidationResult Validate(object value,
        System.Globalization.CultureInfo cultureInfo)
    {
        Course course = (value as BindingGroup).Items[0] as Course;
        if (course.StartDate > course.EndDate)
        {

```

```
        return new ValidationResult(false,
            "Start Date must be earlier than End Date.");
    }
    else
    {
        return ValidationResult.ValidResult;
    }
}
}
```

```

Imports System.Collections.ObjectModel
Imports System.ComponentModel

Public Class MainWindow

    Private Sub dataGrid1_InitializingNewItem(ByVal sender As System.Object, _
        ByVal e As System.Windows.Controls.InitializingNewItemEventArgs) _
        Handles dataGrid1.InitializingNewItem

        Dim newCourse As Course = CType(e.NewItem, Course)
        newCourse.StartDate = DateTime.Today
        newCourse.EndDate = DateTime.Today

    End Sub

End Class

Public Class Courses
    Inherits ObservableCollection(Of Course)

    Sub New()
        Me.Add(New Course With {
            .Name = "Learning WPF", _
            .Id = 1001, _
            .StartDate = New DateTime(2010, 1, 11), _
            .EndDate = New DateTime(2010, 1, 22) _
        })
        Me.Add(New Course With {
            .Name = "Learning Silverlight", _
            .Id = 1002, _
            .StartDate = New DateTime(2010, 1, 25), _
            .EndDate = New DateTime(2010, 2, 5) _
        })
        Me.Add(New Course With {
            .Name = "Learning Expression Blend", _
            .Id = 1003, _
            .StartDate = New DateTime(2010, 2, 8), _
            .EndDate = New DateTime(2010, 2, 19) _
        })
        Me.Add(New Course With {
            .Name = "Learning LINQ", _
            .Id = 1004, _
            .StartDate = New DateTime(2010, 2, 22), _
            .EndDate = New DateTime(2010, 3, 5) _
        })
    End Sub

End Class

Public Class Course
    Implements IEditableObject, INotifyPropertyChanged

    Private _name As String
    Public Property Name As String
        Get
            Return _name
        End Get
        Set
            _name = Value
        End Set
    End Property

```

```

    End Get
    Set(ByVal value As String)
        If _name = value Then Return
        _name = value
        OnPropertyChanged("Name")
    End Set
End Property

Private _number As Integer
Public Property Id As Integer
    Get
        Return _number
    End Get
    Set(ByVal value As Integer)
        If _number = value Then Return
        _number = value
        OnPropertyChanged("Id")
    End Set
End Property

Private _startDate As DateTime
Public Property StartDate As DateTime
    Get
        Return _startDate
    End Get
    Set(ByVal value As DateTime)
        If _startDate = value Then Return
        _startDate = value
        OnPropertyChanged("StartDate")
    End Set
End Property

Private _endDate As DateTime
Public Property EndDate As DateTime
    Get
        Return _endDate
    End Get
    Set(ByVal value As DateTime)
        If _endDate = value Then Return
        _endDate = value
        OnPropertyChanged("EndDate")
    End Set
End Property

#Region "IEditableObject"

Private backupCopy As Course
Private inEdit As Boolean

Public Sub BeginEdit() Implements IEditableObject.BeginEdit
    If inEdit Then Return
    inEdit = True
    backupCopy = CType(Me.MemberwiseClone(), Course)
End Sub

Public Sub CancelEdit() Implements IEditableObject.CancelEdit
    If Not inEdit Then Return
    inEdit = False
    Me.Name = backupCopy.Name
    Me.Id = backupCopy.Id
    Me.StartDate = backupCopy.StartDate
    Me.EndDate = backupCopy.EndDate
End Sub

Public Sub EndEdit() Implements IEditableObject.EndEdit
    If Not inEdit Then Return
    inEdit = False
    backupCopy = Nothing
End Sub

```

```

#End Region

#Region "INotifyPropertyChanged"

    Public Event PropertyChanged As PropertyChangedEventHandler _
        Implements INotifyPropertyChanged.PropertyChanged

    Private Sub OnPropertyChanged(ByVal propertyName As String)
        RaiseEvent PropertyChanged(Me, _
            New PropertyChangedEventArgs(propertyName))
    End Sub

#End Region

End Class

Public Class CourseValidationRule
    Inherits ValidationRule

    Public Overrides Function Validate(ByVal value As Object, _
        ByVal cultureInfo As System.Globalization.CultureInfo) _
        As ValidationResult

        Dim course As Course = _
            CType(CType(value, BindingGroup).Items(0), Course)

        If course.StartDate > course.EndDate Then
            Return New ValidationResult(False, _
                "Start Date must be earlier than End Date.")
        Else
            Return ValidationResult.ValidResult
        End If
    End Function
End Class

```

```

<Window x:Class="DataGridValidation.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:DataGridValidation"
    Title="DataGrid Validation Example" Height="240" Width="600">

    <Grid>

        <Grid.Resources>
            <local:Courses x:Key="courses"/>
        </Grid.Resources>

        <DataGrid Name="dataGrid1" FontSize="20" RowHeaderWidth="27"
            ItemsSource="{StaticResource courses}"
            AutoGenerateColumns="False">

            <DataGrid.Resources>
                <Style x:Key="errorStyle" TargetType="{x:Type TextBox}">
                    <Setter Property="Padding" Value="-2"/>
                    <Style.Triggers>
                        <Trigger Property="Validation.HasError" Value="True">
                            <Setter Property="Background" Value="Red"/>
                            <Setter Property="ToolTip"
                                Value="{Binding RelativeSource={RelativeSource Self},
                                    Path=(Validation.Errors)[0].ErrorContent}"/>
                        </Trigger>
                    </Style.Triggers>
                </Style>
            </DataGrid.Resources>
        
```

```

<DataGrid.Columns>
    <DataGridTextColumn Header="Course Name"
        Binding="{Binding Name, TargetNullValue=(enter a course name)}"/>
    <DataGridTextColumn Header="Course ID"
        EditingElementStyle="{StaticResource errorStyle}"
        Binding="{Binding Id, ValidatesOnExceptions=True}"/>
    <DataGridTextColumn Header="Start Date"
        EditingElementStyle="{StaticResource errorStyle}"
        Binding="{Binding StartDate, ValidatesOnExceptions=True,
            StringFormat=d}"/>
    <DataGridTextColumn Header="End Date"
        EditingElementStyle="{StaticResource errorStyle}"
        Binding="{Binding EndDate, ValidatesOnExceptions=True,
            StringFormat=d}"/>
</DataGrid.Columns>

<DataGrid.RowValidationRules>
    <local:CourseValidationRule ValidationStep="UpdatedValue"/>
</DataGrid.RowValidationRules>

<DataGrid.RowValidationErrorTemplate>
    <ControlTemplate>
        <Grid Margin="0,-2,0,-2"
            ToolTip="{Binding RelativeSource={RelativeSource
                FindAncestor, AncestorType={x:Type DataGridRow}},
                Path=(Validation.Errors)[0].ErrorContent}">
            <Ellipse StrokeThickness="0" Fill="Red"
                Width="{TemplateBinding FontSize}"
                Height="{TemplateBinding FontSize}" />
            <TextBlock Text="!" FontSize="{TemplateBinding FontSize}"
                FontWeight="Bold" Foreground="White"
                HorizontalAlignment="Center" />
        </Grid>
    </ControlTemplate>
</DataGrid.RowValidationErrorTemplate>

</DataGrid>

</Grid>
</Window>

```

## See also

- [DataGridView](#)
- [DataGridView](#)
- [Data Binding](#)
- [Implement Binding Validation](#)
- [Implement Validation Logic on Custom Objects](#)

# Walkthrough: Display data from a SQL Server database in a DataGrid control

2 minutes to read • [Edit Online](#)

In this walkthrough, you retrieve data from a SQL Server database and display that data in a [DataGrid](#) control. You use the ADO.NET Entity Framework to create the entity classes that represent the data, and use LINQ to write a query that retrieves the specified data from an entity class.

## Prerequisites

You need the following components to complete this walkthrough:

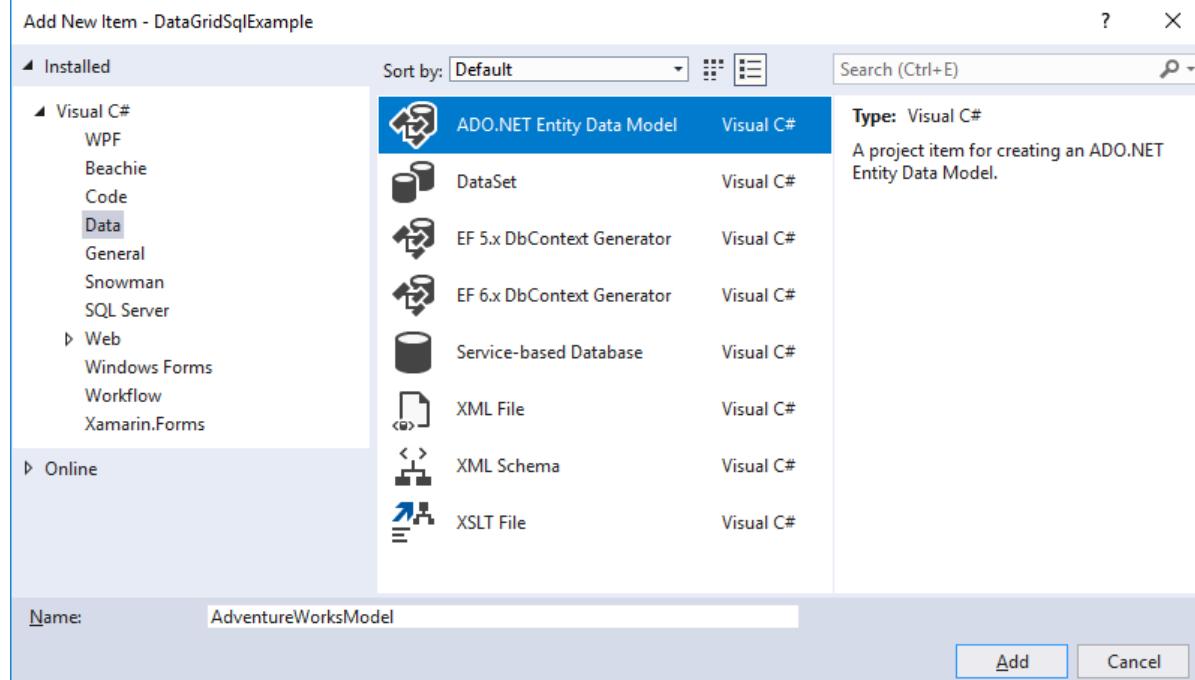
- Visual Studio.
- Access to a running instance of SQL Server or SQL Server Express that has the AdventureWorks sample database attached to it. You can download the AdventureWorks database from the [GitHub](#).

## Create entity classes

1. Create a new WPF Application project in Visual Basic or C#, and name it `DataGridSQLExample`.
2. In Solution Explorer, right-click your project, point to **Add**, and then select **New Item**.

The Add New Item dialog box appears.

3. In the Installed Templates pane, select **Data** and in the list of templates, select **ADO.NET Entity Data Model**.



4. Name the file `AdventureWorksModel.edmx` and then click **Add**.

The Entity Data Model Wizard appears.

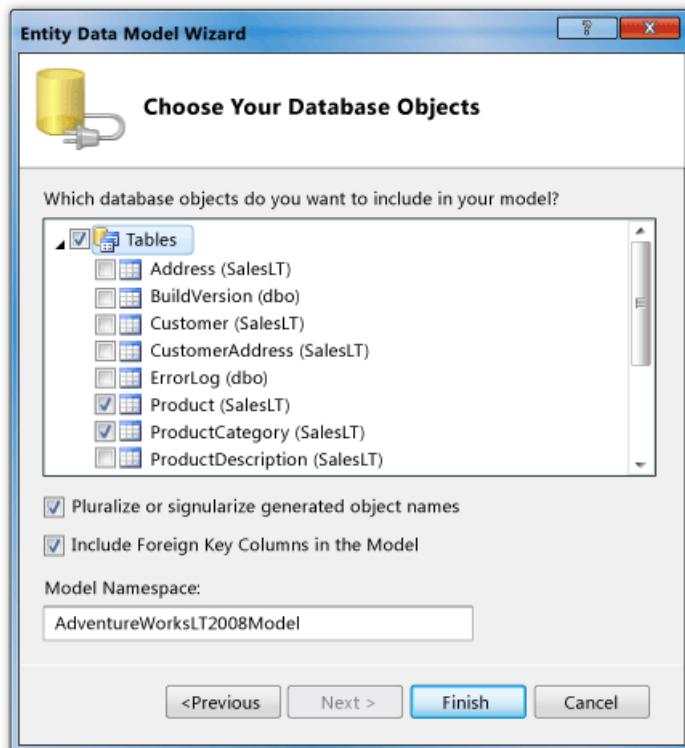
5. In the Choose Model Contents screen, select **EF Designer from database** and then click **Next**.

6. In the Choose Your Data Connection screen, provide the connection to your AdventureWorksLT2008 database. For more information, see [Choose Your Data Connection Dialog Box](#).

Make sure that the name is **AdventureWorksLT2008Entities** and that the **Save entity connection settings in App.Config as** check box is selected, and then click **Next**.

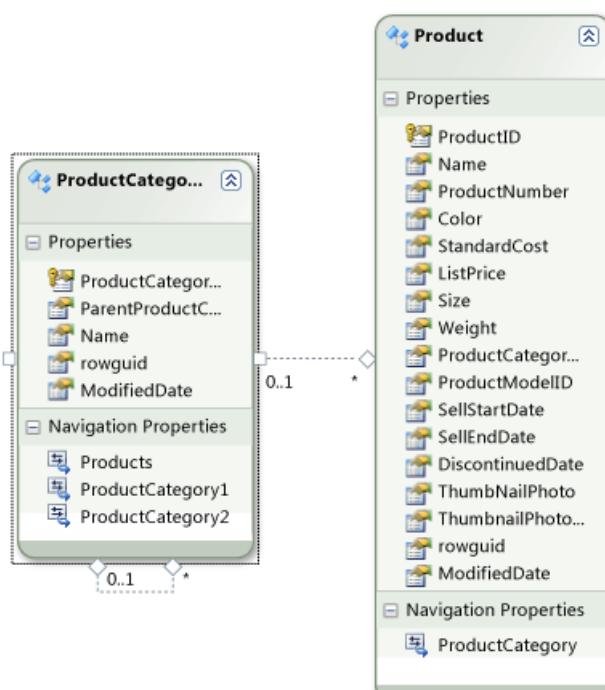
7. In the Choose Your Database Objects screen, expand the Tables node, and select the **Product** and **ProductCategory** tables.

You can generate entity classes for all of the tables; however, in this example you only retrieve data from those two tables.



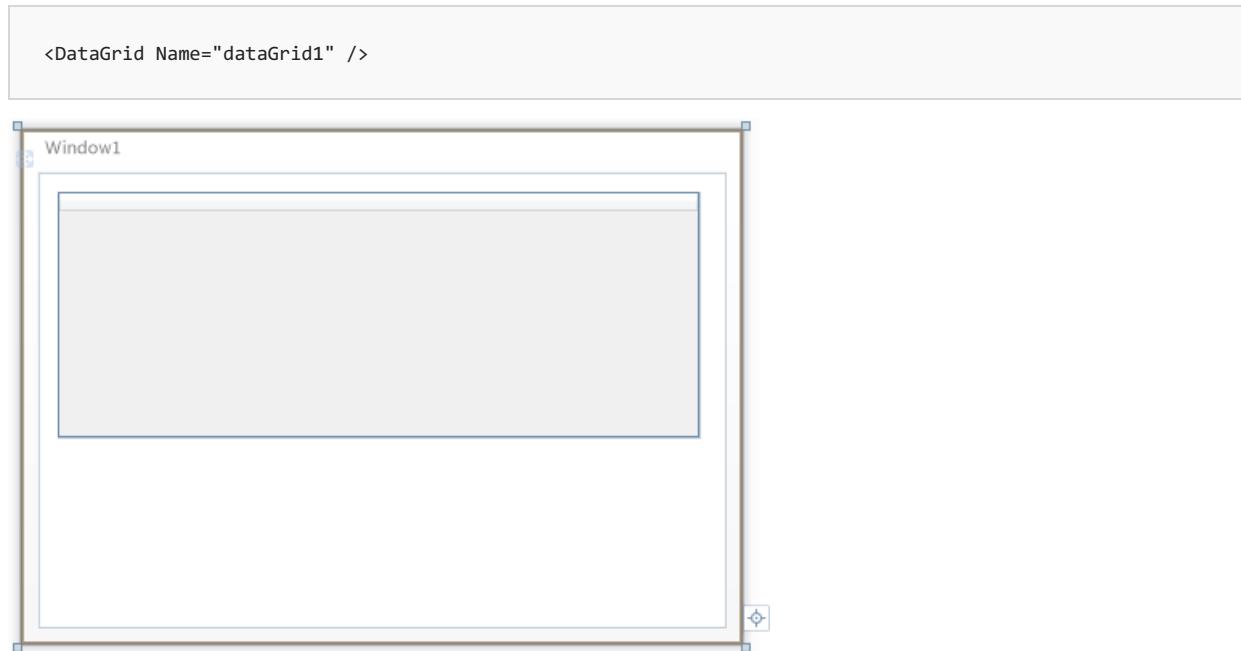
8. Click **Finish**.

The Product and ProductCategory entities are displayed in the Entity Designer.



## Retrieve and present the data

1. Open the MainWindow.xaml file.
2. Set the [Width](#) property on the [Window](#) to 450.
3. In the XAML editor, add the following [DataGrid](#) tag between the `<Grid>` and `</Grid>` tags to add a [DataGrid](#) named `dataGrid1`.



4. Select the [Window](#).
5. Using the Properties window or XAML editor, create an event handler for the [Window](#) named `Window_Loaded` for the [Loaded](#) event. For more information, see [How to: Create a Simple Event Handler](#).

The following shows the XAML for MainWindow.xaml.

### NOTE

If you are using Visual Basic, in the first line of MainWindow.xaml, replace `x:Class="DataGridSQLExample.MainWindow"` with `x:Class="MainWindow"`.

```
<Window x:Class="DataGridSQLExample.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="350" Width="450"
    Loaded="Window_Loaded">
    <Grid>
        <DataGrid Name="dataGrid1" />
    </Grid>
</Window>
```

6. Open the code-behind file (MainWindow.xaml.vb or MainWindow.xaml.cs) for the [Window](#).
7. Add the following code to retrieve only specific values from the joined tables and set the [ItemsSource](#) property of the [DataGrid](#) to the results of the query.

```

using System.Data.Objects;
using System.Linq;
using System.Windows;

namespace DataGridSQLExample
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        AdventureWorksLT2008Entities dataEntities = new AdventureWorksLT2008Entities();

        public MainWindow()
        {
            InitializeComponent();
        }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            ObjectQuery<Product> products = dataEntities.Products;

            var query =
                from product in products
                where product.Color == "Red"
                orderby product.ListPrice
                select new { product.Name, product.Color, CategoryName = product.ProductCategory.Name,
product.ListPrice };

            dataGrid1.ItemsSource = query.ToList();
        }
    }
}

```

```

Imports System.Data.Objects

Class MainWindow
    Dim dataEntities As AdventureWorksLT2008Entities = New AdventureWorksLT2008Entities

    Private Sub Window_Loaded(ByVal sender As System.Object, ByVal e As System.Windows.RoutedEventArgs)
Handles MyBase.Loaded
        Dim products As ObjectQuery(Of Product) = dataEntities.Products

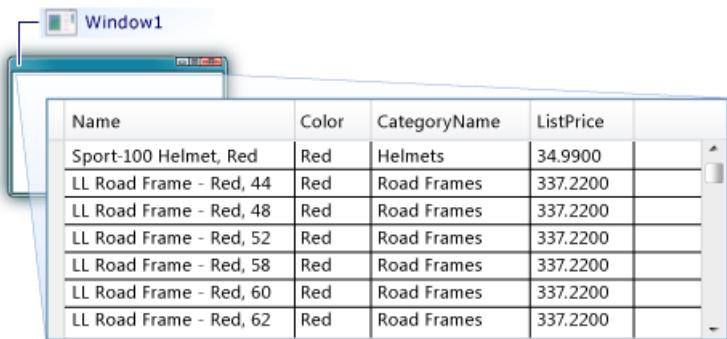
        Dim query = _
            From product In products _
            Where product.Color = "Red" _
            Order By product.ListPrice _
            Select product.Name, product.Color, CategoryName = product.ProductCategory.Name,
product.ListPrice

        dataGrid1.ItemsSource = query.ToList()
    End Sub
End Class

```

8. Run the example.

You should see a [DataGrid](#) that displays data.



A screenshot of a Windows application window titled "Window1". Inside the window, there is a DataGrid control displaying a table of product data. The table has columns for Name, Color, CategoryName, and ListPrice. The data shows various items like helmets and road frames in red, with prices ranging from \$34.99 to \$337.22.

Name	Color	CategoryName	ListPrice	
Sport-100 Helmet, Red	Red	Helmets	34.9900	
LL Road Frame - Red, 44	Red	Road Frames	337.2200	
LL Road Frame - Red, 48	Red	Road Frames	337.2200	
LL Road Frame - Red, 52	Red	Road Frames	337.2200	
LL Road Frame - Red, 58	Red	Road Frames	337.2200	
LL Road Frame - Red, 60	Red	Road Frames	337.2200	
LL Road Frame - Red, 62	Red	Road Frames	337.2200	

## See also

- [DataGridView](#)

# Sizing Options in the DataGrid Control

4 minutes to read • [Edit Online](#)

Various options are available to control how the [DataGrid](#) sizes itself. The [DataGrid](#), and individual rows and columns in the [DataGrid](#), can be set to size automatically to their contents or can be set to specific values. By default, the [DataGrid](#) will grow and shrink to fit the size of its contents.

## Sizing the DataGrid

### Cautions When Using Automatic Sizing

By default, the [Height](#) and [Width](#) properties of the [DataGrid](#) are set to [Double.NaN](#) ("Auto" in XAML), and the [DataGrid](#) will adjust to the size of its contents.

When placed inside a container that does not restrict the size of its children, such as a [Canvas](#) or [StackPanel](#), the [DataGrid](#) will stretch beyond the visible bounds of the container and scrollbars will not be shown. This condition has both usability and performance implications.

When bound to a data set, if the [Height](#) of the [DataGrid](#) is not restricted, it will continue to add a row for each data item in the bound data set. This can cause the [DataGrid](#) to grow outside the visible bounds of your application as rows are added. The [DataGrid](#) will not show scrollbars in this case because its [Height](#) will continue to grow to accommodate the new rows.

An object is created for each row in the [DataGrid](#). If you are working with a large data set and you allow the [DataGrid](#) to automatically size itself, the creation of a large number of objects may affect the performance of your application.

To avoid these issues when you work with large data sets, it is recommended that you specifically set the [Height](#) of the [DataGrid](#) or place it in a container that will restrict its [Height](#), such as a [Grid](#). When the [Height](#) is restricted, the [DataGrid](#) will only create the rows that will fit within its specified [Height](#), and will recycle those rows as needed to display new data.

### Setting the DataGrid Size

The [DataGrid](#) can be set to automatically size within specified boundaries, or the [DataGrid](#) can be set to a specific size. The following table shows the properties that can be set to control the [DataGrid](#) size.

PROPERTY	DESCRIPTION
<a href="#">Height</a>	Sets a specific height for the <a href="#">DataGrid</a> .
<a href="#">MaxHeight</a>	Sets the upper bound for the height of the <a href="#">DataGrid</a> . The <a href="#">DataGrid</a> will grow vertically until it reaches this height.
<a href="#">MinHeight</a>	Sets the lower bound for the height of the <a href="#">DataGrid</a> . The <a href="#">DataGrid</a> will shrink vertically until it reaches this height.
<a href="#">Width</a>	Sets a specific width for the <a href="#">DataGrid</a> .
<a href="#">MaxWidth</a>	Sets the upper bound for the width of the <a href="#">DataGrid</a> . The <a href="#">DataGrid</a> will grow horizontally until it reaches this width.

PROPERTY	DESCRIPTION
MinWidth	Sets the lower bound for the width of the <a href="#">DataGrid</a> . The <a href="#">DataGrid</a> will shrink horizontally until it reaches this width.

## Sizing Rows and Row Headers

### DataGrid Rows

By default, a [DataGrid](#) row's [Height](#) property is set to [Double.NaN](#) ("Auto" in XAML), and the row height will expand to the size of its contents. The height of all rows in the [DataGrid](#) can be specified by setting the [DataGrid.RowHeight](#) property. Users can change the row height by dragging the row header dividers.

### DataGrid Row Headers

To display row headers, the [HeadersVisibility](#) property must be set to [DataGridHeadersVisibility.Row](#) or [DataGridHeadersVisibility.All](#). By default, row headers are displayed and they automatically size to fit their content. The row headers can be given a specific width by setting the [DataGrid.RowHeaderWidth](#) property.

## Sizing Columns and Column Headers

### DataGrid Columns

The [DataGrid](#) uses values of the [DataGridLength](#) and the [DataGridLengthUnitType](#) structure to specify absolute or automatic sizing modes.

The following table shows the values provided by the [DataGridLengthUnitType](#) structure.

NAME	DESCRIPTION
Auto	The default automatic sizing mode sizes <a href="#">DataGrid</a> columns based on the contents of both cells and column headers.
SizeToCells	The cell-based automatic sizing mode sizes <a href="#">DataGrid</a> columns based on the contents of cells in the column, not including column headers.
SizeToHeader	The header-based automatic sizing mode sizes <a href="#">DataGrid</a> columns based on the contents of column headers only.
Pixel	The pixel-based sizing mode sizes <a href="#">DataGrid</a> columns based on the numeric value provided.
Star	<p>The star sizing mode is used to distribute available space by weighted proportions.</p> <p>In XAML, star values are expressed as <math>n*</math> where <math>n</math> represents a numeric value. <math>1*</math> is equivalent to <math>*</math>. For example, if two columns in a <a href="#">DataGrid</a> had widths of <math>*</math> and <math>2*</math>, the first column would receive one portion of the available space and the second column would receive two portions of the available space.</p>

The [DataGridLengthConverter](#) class can be used to convert data between numeric or string values and [DataGridLength](#) values.

By default, the [DataGrid.ColumnWidth](#) property is set to [SizeToHeader](#), and the [DataGridColumn.Width](#) property is set to [Auto](#). When the sizing mode is set to [Auto](#) or [SizeToCells](#), columns will grow to the width of their widest

visible content. When scrolling, these sizing modes will cause columns to expand if content that is larger than the current column size is scrolled into view. The column will not shrink after the content is scrolled out of view.

Columns in the [DataGrid](#) can also be set to automatically size only within specified boundaries, or columns can be set to a specific size. The following table shows the properties that can be set to control column sizes.

PROPERTY	DESCRIPTION
<a href="#">DataGrid.MaxColumnWidth</a>	Sets the upper bound for all columns in the <a href="#">DataGrid</a> .
<a href="#">DataGridColumn.MaxWidth</a>	Sets the upper bound for an individual column. Overrides <a href="#">DataGrid.MaxColumnWidth</a> .
<a href="#">DataGrid.MinColumnWidth</a>	Sets the lower bound for all columns in the <a href="#">DataGrid</a> .
<a href="#">DataGridColumn.MinWidth</a>	Sets the lower bound for an individual column. Overrides <a href="#">DataGrid.MinColumnWidth</a> .
<a href="#">DataGrid.ColumnWidth</a>	Sets a specific width for all columns in the <a href="#">DataGrid</a> .
<a href="#">DataGridColumn.Width</a>	Sets a specific width for an individual column. Overrides <a href="#">DataGrid.ColumnWidth</a> .

## DataGrid Column Headers

By default, [DataGrid](#) column headers are displayed. To hide column headers, the [HeadersVisibility](#) property must be set to [DataGridHeadersVisibility.Row](#) or [DataGridHeadersVisibility.None](#). By default, when column headers are displayed, they automatically size to fit their content. The column headers can be given a specific height by setting the [DataGrid.ColumnHeaderHeight](#) property.

## Resizing with the Mouse

Users can resize [DataGrid](#) rows and columns by dragging the row or column header dividers. The [DataGrid](#) also supports automatic resizing of rows and columns by double-clicking the row or column header divider. To prevent a user from resizing particular columns, set the [DataGridColumn.CanUserResize](#) property to `false` for the individual columns. To prevent users from resizing all columns, set the [DataGrid.CanUserResizeColumns](#) property to `false`. To prevent users from resizing all rows, set the [DataGrid.CanUserResizeRows](#) property to `false`.

## See also

- [DataGrid](#)
- [DataGridColumn](#)
- [DataGridLength](#)
- [DataGridLengthConverter](#)

# DatePicker

2 minutes to read • [Edit Online](#)

The [DatePicker](#) control allows the user to select a date by either typing it into a text field or by using a drop-down [Calendar](#) control.

The following illustration shows a [DatePicker](#).



## DatePicker Control

Many of a [DatePicker](#) control's properties are for managing its built-in [Calendar](#), and function identically to the equivalent property in [Calendar](#). In particular, the [DatePicker.IsTodayHighlighted](#), [DatePicker.FirstDayOfWeek](#), [DatePicker.BlackoutDates](#), [DatePicker.DisplayDateStart](#), [DatePicker.DisplayDateEnd](#), [DatePicker.DisplayDate](#), and [DatePicker.SelectedDate](#) properties function identically to their [Calendar](#) counterparts. For more information, see [Calendar](#).

Users can type a date directly into a text field, which sets the [Text](#) property. If the [DatePicker](#) cannot convert the entered string to a valid date, the [DateValidationError](#) event will be raised. By default, this causes an exception, but an event handler for [DateValidationError](#) can set the [ThrowException](#) property to `false` and prevent an exception from being raised.

## See also

- [Controls](#)
- [Styling and Templating](#)

# DockPanel

2 minutes to read • [Edit Online](#)

The [DockPanel](#) element is used to position child content along the edge of a layout container.

## In This Section

[How-to Topics](#)

## Reference

[Panel](#)

[Canvas](#)

[DockPanel](#)

[Grid](#)

[StackPanel](#)

[VirtualizingStackPanel](#)

[WrapPanel](#)

## Related Sections

[Layout](#)

[Walkthrough: My first WPF desktop application](#)

[ScrollViewer Overview](#)

# DockPanel How-to Topics

2 minutes to read • [Edit Online](#)

The topics in this section describe how to use the [DockPanel](#) element to position child elements along the edge of a layout container.

## In This Section

[Get or Set a Dock Value](#)

[Create a DockPanel](#)

[Partition Space by Using the DockPanel Element](#)

## Reference

[Panel](#)

[Canvas](#)

[DockPanel](#)

[Grid](#)

[StackPanel](#)

[VirtualizingStackPanel](#)

[WrapPanel](#)

## Related Sections

[Layout](#)

[Walkthrough: My first WPF desktop application](#)

[ScrollViewer Overview](#)

# How to: Get or Set a Dock Value

2 minutes to read • [Edit Online](#)

The following example shows how to assign a `Dock` value for an object. The example uses the `GetDock` and `SetDock` methods of `DockPanel`.

## Example

The example creates an instance of the `TextBlock` element, `txt1`, and assigns a `Dock` value of `Top` by using the `SetDock` method of `DockPanel`. It then appends the value of the `Dock` property to the `Text` of the `TextBlock` element by using the `GetDock` method. Finally, the example adds the `TextBlock` element to the parent `DockPanel`, `dp1`.

```
// Create the Panel DockPanel
dp1 = new DockPanel();

// Create a Text Control and then set its Dock property
txt1 = new TextBlock();
DockPanel.SetDock(txt1, System.Windows.Controls.Dock.Top);
txt1.Text = "The Dock Property is set to " + DockPanel.GetDock(txt1);
dp1.Children.Add(txt1);
mainWindow.Content = dp1;
mainWindow.Show();
```

```
' Create the Panel DockPanel
dp1 = New DockPanel()

' Create a Text Control and then set its Dock property
txt1 = New TextBlock()
DockPanel.SetDock(txt1, Dock.Top)
txt1.Text = "The Dock Property is set to " & DockPanel.GetDock(txt1).ToString()
dp1.Children.Add(txt1)
_mainWindow.Content = dp1
_mainWindow.Show()
```

## See also

- [DockPanel](#)
- [GetDock](#)
- [SetDock](#)
- [Panels Overview](#)

# How to: Create a DockPanel

2 minutes to read • [Edit Online](#)

## Example

The following example creates and uses an instance of [DockPanel](#) by using code. The example shows you how to partition space by creating five [Rectangle](#) elements and positioning (docking) them inside a parent [DockPanel](#). If you retain the default setting, the final rectangle fills all the remaining unallocated space.

```
private void CreateAndShowMainWindow()
{
    // Create the application's main window
    mainWindow = new Window ();

    // Create a DockPanel
    DockPanel myDockPanel = new DockPanel();

    // Add the first rectangle to the DockPanel
    Rectangle rect1 = new Rectangle();
    rect1.Stroke = Brushes.Black;
    rect1.Fill = Brushes.SkyBlue;
    rect1.Height = 25;
    DockPanel.SetDock(rect1, Dock.Top);
    myDockPanel.Children.Add(rect1);

    // Add the second rectangle to the DockPanel
    Rectangle rect2 = new Rectangle();
    rect2.Stroke = Brushes.Black;
    rect2.Fill = Brushes.SkyBlue;
    rect2.Height = 25;
    DockPanel.SetDock(rect2, Dock.Top);
    myDockPanel.Children.Add(rect2);

    // Add the third rectangle to the DockPanel
    Rectangle rect4 = new Rectangle();
    rect4.Stroke = Brushes.Black;
    rect4.Fill = Brushes.Khaki;
    rect4.Height = 25;
    DockPanel.SetDock(rect4, Dock.Bottom);
    myDockPanel.Children.Add(rect4);

    // Add the fourth rectangle to the DockPanel
    Rectangle rect3 = new Rectangle();
    rect3.Stroke = Brushes.Black;
    rect3.Fill = Brushes.PaleGreen;
    rect3.Width = 200;
    DockPanel.SetDock(rect3, Dock.Left);
    myDockPanel.Children.Add(rect3);

    // Add the final rectangle to the DockPanel
    Rectangle rect5 = new Rectangle();
    rect5.Stroke = Brushes.Black;
    rect5.Fill = Brushes.White;
    myDockPanel.Children.Add(rect5);

    // Add the DockPanel to the Window as Content and show the Window
    mainWindow.Content = myDockPanel;
    mainWindow.Title = "DockPanel Sample";
    mainWindow.Show();
}
}
```

```
WindowTitle = "DockPanel Sample"
'Create a DockPanel as the root Panel
Dim myDP As New DockPanel()
' Add the first Rectangle to the DockPanel
Dim rect1 As New Rectangle
rect1.Stroke = Brushes.Black
rect1.Fill = Brushes.SkyBlue
rect1.Height = 25
DockPanel.SetDock(rect1, Dock.Top)
myDP.Children.Add(rect1)

' Add the second Rectangle to the DockPanel
Dim rect2 As New Rectangle
rect2.Stroke = Brushes.Black
rect2.Fill = Brushes.SkyBlue
rect2.Height = 25
DockPanel.SetDock(rect2, Dock.Top)
myDP.Children.Add(rect2)

' Add the third Rectangle to the DockPanel
Dim rect3 As New Rectangle
rect3.Stroke = Brushes.Black
rect3.Fill = Brushes.Khaki
rect3.Height = 25
DockPanel.SetDock(rect3, Dock.Bottom)
myDP.Children.Add(rect3)

' Add the fourth Rectangle to the DockPanel
Dim rect4 As New Rectangle
rect4.Stroke = Brushes.Black
rect4.Fill = Brushes.PaleGreen
rect4.Width = 200
rect4.VerticalAlignment = VerticalAlignment.Stretch
DockPanel.SetDock(rect4, Dock.Left)
myDP.Children.Add(rect4)

' Add the fifth Rectangle to the DockPanel
Dim rect5 As New Rectangle
rect5.Stroke = Brushes.Black
rect5.Fill = Brushes.White
myDP.Children.Add(rect5)
Me.Content = myDP
```

## See also

- [DockPanel](#)
- [Dock](#)
- [Panels Overview](#)

# How to: Partition Space by Using the DockPanel Element

4 minutes to read • [Edit Online](#)

The following example creates a simple user interface (UI) framework using a `DockPanel` element. The `DockPanel` partitions available space to its child elements.

## Example

This example uses the `Dock` property, which is an attached property, to dock two identical `Border` elements at the `Top` of the partitioned space. A third `Border` element is docked to the `Left`, with its width set to 200 pixels. A fourth `Border` is docked to the `Bottom` of the screen. The last `Border` element automatically fills the remaining space.

```
// Create the application's main window
mainWindow = gcnew Window();
mainWindow->Title = "DockPanel Sample";

// Create the DockPanel
DockPanel^ myDockPanel = gcnew DockPanel();
myDockPanel->LastChildFill = true;

// Define the child content
Border^ myBorder1 = gcnew Border();
myBorder1->Height = 25;
myBorder1->Background = Brushes::SkyBlue;
myBorder1->BorderBrush = Brushes::Black;
myBorder1->BorderThickness = Thickness(1);
DockPanel::SetDock(myBorder1, Dock::Top);
TextBlock^ myTextBlock1 = gcnew TextBlock();
myTextBlock1->Foreground = Brushes::Black;
myTextBlock1->Text = "Dock = Top";
myBorder1->Child = myTextBlock1;

Border^ myBorder2 = gcnew Border();
myBorder2->Height = 25;
myBorder2->Background = Brushes::SkyBlue;
myBorder2->BorderBrush = Brushes::Black;
myBorder2->BorderThickness = Thickness(1);
DockPanel::SetDock(myBorder2, Dock::Top);
TextBlock^ myTextBlock2 = gcnew TextBlock();
myTextBlock2->Foreground = Brushes::Black;
myTextBlock2->Text = "Dock = Top";
myBorder2->Child = myTextBlock2;

Border^ myBorder3 = gcnew Border();
myBorder3->Height = 25;
myBorder3->Background = Brushes::LemonChiffon;
myBorder3->BorderBrush = Brushes::Black;
myBorder3->BorderThickness = Thickness(1);
DockPanel::SetDock(myBorder3, Dock::Bottom);
TextBlock^ myTextBlock3 = gcnew TextBlock();
myTextBlock3->Foreground = Brushes::Black;
myTextBlock3->Text = "Dock = Bottom";
myBorder3->Child = myTextBlock3;

Border^ myBorder4 = gcnew Border();
myBorder4->Width = 200;
myBorder4->Background = Brushes::PaleGreen;
```

```

myBorder4->BorderBrush = Brushes::Black;
myBorder4->BorderThickness = Thickness(1);
DockPanel::SetDock(myBorder4, Dock::Left);
TextBlock^ myTextBlock4 = gcnew TextBlock();
myTextBlock4->Foreground = Brushes::Black;
myTextBlock4->Text = "Dock = Left";
myBorder4->Child = myTextBlock4;

Border^ myBorder5 = gcnew Border();
myBorder5->Background = Brushes::White;
myBorder5->BorderBrush = Brushes::Black;
myBorder5->BorderThickness = Thickness(1);
TextBlock^ myTextBlock5 = gcnew TextBlock();
myTextBlock5->Foreground = Brushes::Black;
myTextBlock5->Text = "This content will Fill the remaining space";
myBorder5->Child = myTextBlock5;

// Add child elements to the DockPanel Children collection
myDockPanel->Children->Add(myBorder1);
myDockPanel->Children->Add(myBorder2);
myDockPanel->Children->Add(myBorder3);
myDockPanel->Children->Add(myBorder4);
myDockPanel->Children->Add(myBorder5);

// Add the parent Canvas as the Content of the Window Object
mainWindow->Content = myDockPanel;
mainWindow->Show();

```

```

// Create the application's main window
mainWindow = new Window ();
mainWindow.Title = "DockPanel Sample";

// Create the DockPanel
DockPanel myDockPanel = new DockPanel();
myDockPanel.LastChildFill = true;

// Define the child content
Border myBorder1 = new Border();
myBorder1.Height = 25;
myBorder1.Background = Brushes.SkyBlue;
myBorder1.BorderBrush = Brushes.Black;
myBorder1.BorderThickness = new Thickness(1);
DockPanel.SetDock(myBorder1, Dock.Top);
TextBlock myTextBlock1 = new TextBlock();
myTextBlock1.Foreground = Brushes.Black;
myTextBlock1.Text = "Dock = Top";
myBorder1.Child = myTextBlock1;

Border myBorder2 = new Border();
myBorder2.Height = 25;
myBorder2.Background = Brushes.SkyBlue;
myBorder2.BorderBrush = Brushes.Black;
myBorder2.BorderThickness = new Thickness(1);
DockPanel.SetDock(myBorder2, Dock.Top);
TextBlock myTextBlock2 = new TextBlock();
myTextBlock2.Foreground = Brushes.Black;
myTextBlock2.Text = "Dock = Top";
myBorder2.Child = myTextBlock2;

Border myBorder3 = new Border();
myBorder3.Height = 25;
myBorder3.Background = Brushes.LemonChiffon;
myBorder3.BorderBrush = Brushes.Black;
myBorder3.BorderThickness = new Thickness(1);
DockPanel.SetDock(myBorder3, Dock.Bottom);

```

```
TextBlock myTextBlock3 = new TextBlock();
myTextBlock3.Foreground = Brushes.Black;
myTextBlock3.Text = "Dock = Bottom";
myBorder3.Child = myTextBlock3;

Border myBorder4 = new Border();
myBorder4.Width = 200;
myBorder4.Background = Brushes.PaleGreen;
myBorder4.BorderBrush = Brushes.Black;
myBorder4.BorderThickness = new Thickness(1);
DockPanel.SetDock(myBorder4, Dock.Left);
TextBlock myTextBlock4 = new TextBlock();
myTextBlock4.Foreground = Brushes.Black;
myTextBlock4.Text = "Dock = Left";
myBorder4.Child = myTextBlock4;

Border myBorder5 = new Border();
myBorder5.Background = Brushes.White;
myBorder5.BorderBrush = Brushes.Black;
myBorder5.BorderThickness = new Thickness(1);
TextBlock myTextBlock5 = new TextBlock();
myTextBlock5.Foreground = Brushes.Black;
myTextBlock5.Text = "This content will Fill the remaining space";
myBorder5.Child = myTextBlock5;

// Add child elements to the DockPanel Children collection
myDockPanel.Children.Add(myBorder1);
myDockPanel.Children.Add(myBorder2);
myDockPanel.Children.Add(myBorder3);
myDockPanel.Children.Add(myBorder4);
myDockPanel.Children.Add(myBorder5);

// Add the parent Canvas as the Content of the Window Object
mainWindow.Content = myDockPanel;
mainWindow.Show();
```

```

WindowTitle = "DockPanel Sample"
'Create a DockPanel as the root Panel
Dim myDockPanel As New DockPanel()
myDockPanel.LastChildFill = True

' Define the child content
Dim myBorder1 As New Border()
myBorder1.Height = 25
myBorder1.Background = Brushes.SkyBlue
myBorder1.BorderBrush = Brushes.Black
myBorder1.BorderThickness = New Thickness(1)
DockPanel.SetDock(myBorder1, Dock.Top)
Dim myTextBlock1 As New TextBlock()
myTextBlock1.Foreground = Brushes.Black
myTextBlock1.Text = "Dock = Top"
myBorder1.Child = myTextBlock1

Dim myBorder2 As New Border()
myBorder2.Height = 25
myBorder2.Background = Brushes.SkyBlue
myBorder2.BorderBrush = Brushes.Black
myBorder2.BorderThickness = New Thickness(1)
DockPanel.SetDock(myBorder2, Dock.Top)
Dim myTextBlock2 As New TextBlock()
myTextBlock2.Foreground = Brushes.Black
myTextBlock2.Text = "Dock = Top"
myBorder2.Child = myTextBlock2

Dim myBorder3 As New Border()
myBorder3.Height = 25
myBorder3.Background = Brushes.LemonChiffon
myBorder3.BorderBrush = Brushes.Black
myBorder3.BorderThickness = New Thickness(1)
DockPanel.SetDock(myBorder3, Dock.Bottom)
Dim myTextBlock3 As New TextBlock()
myTextBlock3.Foreground = Brushes.Black
myTextBlock3.Text = "Dock = Bottom"
myBorder3.Child = myTextBlock3

Dim myBorder4 As New Border()
myBorder4.Width = 200
myBorder4.Background = Brushes.PaleGreen
myBorder4.BorderBrush = Brushes.Black
myBorder4.BorderThickness = New Thickness(1)
DockPanel.SetDock(myBorder4, Dock.Left)
Dim myTextBlock4 As New TextBlock()
myTextBlock4.Foreground = Brushes.Black
myTextBlock4.Text = "Dock = Left"
myBorder4.Child = myTextBlock4

Dim myBorder5 As New Border()
myBorder5.Background = Brushes.White
myBorder5.BorderBrush = Brushes.Black
myBorder5.BorderThickness = New Thickness(1)
Dim myTextBlock5 As New TextBlock()
myTextBlock5.Foreground = Brushes.Black
myTextBlock5.Text = "This content will Fill the remaining space"
myBorder5.Child = myTextBlock5

' Add child elements to the DockPanel Children collection
myDockPanel.Children.Add(myBorder1)
myDockPanel.Children.Add(myBorder2)
myDockPanel.Children.Add(myBorder3)
myDockPanel.Children.Add(myBorder4)
myDockPanel.Children.Add(myBorder5)
Me.Content = myDockPanel

```

```

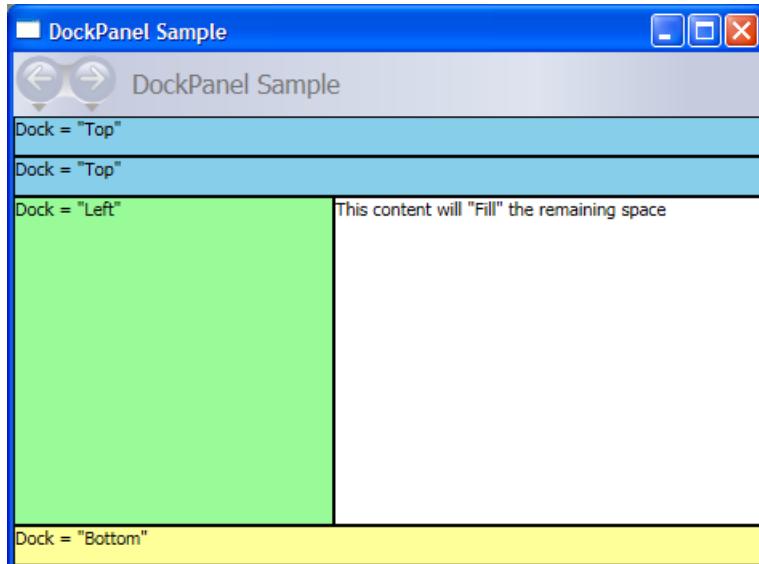
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" WindowTitle="DockPanel Sample">
    <DockPanel LastChildFill="True">
        <Border Height="25" Background="SkyBlue" BorderBrush="Black" BorderThickness="1" DockPanel.Dock="Top">
            <TextBlock Foreground="Black">Dock = "Top"</TextBlock>
        </Border>
        <Border Height="25" Background="SkyBlue" BorderBrush="Black" BorderThickness="1" DockPanel.Dock="Top">
            <TextBlock Foreground="Black">Dock = "Top"</TextBlock>
        </Border>
        <Border Height="25" Background="LemonChiffon" BorderBrush="Black" BorderThickness="1" DockPanel.Dock="Bottom">
            <TextBlock Foreground="Black">Dock = "Bottom"</TextBlock>
        </Border>
        <Border Width="200" Background="PaleGreen" BorderBrush="Black" BorderThickness="1" DockPanel.Dock="Left">
            <TextBlock Foreground="Black">Dock = "Left"</TextBlock>
        </Border>
        <Border Background="White" BorderBrush="Black" BorderThickness="1">
            <TextBlock Foreground="Black">This content will "Fill" the remaining space</TextBlock>
        </Border>
    </DockPanel>
</Page>

```

#### NOTE

By default, the last child of a [DockPanel](#) element fills the remaining unallocated space. If you do not want this behavior, set `LastChildFill="False"`.

The compiled application yields a new UI that looks like this.



## See also

- [DockPanel](#)
- [Panels Overview](#)

# DocumentViewer

2 minutes to read • [Edit Online](#)

The [DocumentViewer](#) control is used to view [FixedDocument](#) content (such as XML Paper Specification (XPS) documents) in a paginated format.

## Reference

[DocumentViewer](#)

[FixedDocument](#)

## See also

- [Documents](#)
- [Document Serialization and Storage](#)
- [Printing Overview](#)

# Expander

2 minutes to read • [Edit Online](#)

An [Expander](#) allows a user to view a header and expand that header to see further details, or to collapse a section up to a header.

The following illustration provides an example of this control in its expanded position.



## In This Section

[Expander Overview](#)

[How-to Topics](#)

## Reference

[Expander](#)

## Related Sections

# Expander Overview

5 minutes to read • [Edit Online](#)

An [Expander](#) control provides a way to provide content in an expandable area that resembles a window and includes a header.

## Creating a Simple Expander

The following example shows how to create a simple [Expander](#) control. This example creates an [Expander](#) that looks like the previous illustration.

```
<Expander Name="myExpander" Background="Tan"
    HorizontalAlignment="Left" Header="My Expander"
    ExpandDirection="Down" IsExpanded="True" Width="100">
    <TextBlock TextWrapping="Wrap">
        Lorem ipsum dolor sit amet, consectetur
        adipisicing elit, sed do eiusmod tempor incididunt ut
        labore et dolore magna aliqua
    </TextBlock>
</Expander>
```

The [Content](#) and [Header](#) of an [Expander](#) can also contain complex content, such as [RadioButton](#) and [Image](#) objects.

## Setting the Direction of the Expanding Content Area

You can set the content area of an [Expander](#) control to expand in one of four directions ([Down](#), [Up](#), [Left](#), or [Right](#)) by using the [ExpandDirection](#) property. When the content area is collapsed, only the [ExpanderHeader](#) and its toggle button appear. A [Button](#) control that displays a directional arrow is used as a toggle button to expand or collapse the content area. When expanded, the [Expander](#) tries to display all of its content in a window-like area.

## Controlling the Size of an Expander in a Panel

If an [Expander](#) control is inside a layout control that inherits from [Panel](#), such as [StackPanel](#), do not specify a [Height](#) on the [Expander](#) when the [ExpandDirection](#) property is set to [Down](#) or [Up](#). Similarly, do not specify a [Width](#) on the [Expander](#) when the [ExpandDirection](#) property is set to [Left](#) or [Right](#).

When you set a size dimension on an [Expander](#) control in the direction that the expanded content is displayed, the [Expander](#) takes control of the area that is used by the content and displays a border around it. The border shows even when the content is collapsed. To set the size of the expanded content area, set size dimensions on the content of the [Expander](#), or if you want scrolling capability, on the [ScrollViewer](#) that encloses the content.

When an [Expander](#) control is the last element in a [DockPanel](#), Windows Presentation Foundation (WPF) automatically sets the [Expander](#) dimensions to equal the remaining area of the [DockPanel](#). To prevent this default behavior, set the [LastChildFill](#) property on the [DockPanel](#) object to `false`, or make sure that the [Expander](#) is not the last element in a [DockPanel](#).

## Creating Scrollable Content

If the content is too large for the size of the content area, you can wrap the content of an [Expander](#) in a [ScrollViewer](#) in order to provide scrollable content. The [Expander](#) control does not automatically provide scrolling capability. The following illustration shows an [Expander](#) control that contains a [ScrollViewer](#) control.

## Expander in a ScrollViewer



When you place an [Expander](#) control in a [ScrollViewer](#), set the [ScrollViewer](#) dimension property that corresponds to the direction in which the [Expander](#) content opens to the size of the [Expander](#) content area. For example, if you set the [ExpandDirection](#) property on the [Expander](#) to [Down](#) (the content area opens down), set the [Height](#) property on the [ScrollViewer](#) control to the required height for the content area. If you instead set the height dimension on the content itself, [ScrollViewer](#) does not recognize this setting and therefore, does not provide scrollable content.

The following example shows how to create an [Expander](#) control that has complex content and that contains a [ScrollViewer](#) control. This example creates an [Expander](#) that is like the illustration at the beginning of this section.

```

void MakeExpander()
{
    //Create containing stack panel and assign to Grid row/col
    StackPanel sp = new StackPanel();
    Grid.SetRow(sp, 0);
    Grid.SetColumn(sp, 1);
    sp.Background = Brushes.LightSalmon;

    //Create column title
    TextBlock colTitle = new TextBlock();
    colTitle.Text = "EXPANDER CREATED FROM CODE";
    colTitle.HorizontalAlignment= HorizontalAlignment.Center;
    colTitle.Margin.Bottom.Equals(20);
    sp.Children.Add(colTitle);

    //Create Expander object
    Expander exp = new Expander();

    //Create Bullet Panel for Expander Header
    BulletDecorator bp = new BulletDecorator();
    Image i = new Image();
    BitmapImage bi= new BitmapImage();
    bi.UriSource = new Uri(@"pack://application:,,,/images/icon.jpg");
    i.Source = bi;
    i.Width = 10;
    bp.Bullet = i;
    TextBlock tb = new TextBlock();
    tb.Text = "My Expander";
    tb.Margin = new Thickness(20,0,0,0);
    bp.Child = tb;
    exp.Header = bp;

    //Create TextBlock with ScrollViewer for Expander Content
    StackPanel spScroll = new StackPanel();
    TextBlock tbc = new TextBlock();
    tbc.Text =
        "Lorem ipsum dolor sit amet, consectetur adipisicing elit," +
        "sed do eiusmod tempor incididunt ut labore et dolore magna" +
        "aliqua. Ut enim ad minim veniam, quis nostrud exercitation" +
        "ullamco laboris nisi ut aliquip ex ea commodo consequat." +
        "Duis aute irure dolor in reprehenderit in voluptate velit" +
        "esse cillum dolore eu fugiat nulla pariatur. Excepteur sint" +
        "occaecat cupidatat non proident, sunt in culpa qui officia" +
        "deserunt mollit anim id est laborum.";
    tbc.TextWrapping = TextWrapping.Wrap;

    spScroll.Children.Add(tbc);
    ScrollViewer scr = new ScrollViewer();
    scr.Content = spScroll;
    scr.Height = 50;
    exp.Content = scr;

    exp.Width=200;
    exp.HorizontalContentAlignment= HorizontalAlignment.Stretch;
    //Insert Expander into the StackPanel and add it to the
    //Grid
    sp.Children.Add(exp);
    myGrid.Children.Add(sp);
}

```

```

Sub MakeExpander()

    'Create containing stack panel and assign to Grid row/col
    Dim sp As StackPanel = New StackPanel()
    Grid.SetRow(sp, 0)
    Grid.SetColumn(sp, 1)
    sp.Background = Brushes.LightSalmon

    'Create column title
    Dim colTitle As TextBlock = New TextBlock()
    colTitle.Text = "EXPANDER CREATED FROM CODE"
    colTitle.HorizontalAlignment = HorizontalAlignment.Center
    colTitle.Margin.Bottom.Equals(20)
    sp.Children.Add(colTitle)

    'Create Expander object
    Dim exp As Expander = New Expander()

    'Create Bullet Panel for Expander Header
    Dim bp As BulletDecorator = New BulletDecorator()
    Dim i As Image = New Image()
    Dim bi As BitmapImage = New BitmapImage()
    bi.UriSource = New Uri("pack://application:,,./images/icon.jpg")
    i.Source = bi
    i.Width = 10
    bp.Bullet = i
    Dim tb As TextBlock = New TextBlock()
    tb.Text = "My Expander"
    tb.Margin = New Thickness(20, 0, 0, 0)
    bp.Child = tb
    exp.Header = bp

    'Create TextBlock with ScrollViewer for Expander Content
    Dim spScroll As StackPanel = New StackPanel()
    Dim tbc As TextBlock = New TextBlock()
    tbc.Text = _
        "Lorem ipsum dolor sit amet, consectetur adipisicing elit," + _
        "sed do eiusmod tempor incididunt ut labore et dolore magna" + _
        "aliqua. Ut enim ad minim veniam, quis nostrud exercitation" + _
        "ullamco laboris nisi ut aliquip ex ea commodo consequat." + _
        "Duis aute irure dolor in reprehenderit in voluptate velit" + _
        "esse cillum dolore eu fugiat nulla pariatur. Excepteur sint" + _
        "occaecat cupidatat non proident, sunt in culpa qui officia" + _
        "deserunt mollit anim id est laborum."
    tbc.TextWrapping = TextWrapping.Wrap

    spScroll.Children.Add(tbc)
    Dim scr As ScrollViewer = New ScrollViewer()
    scr.Content = spScroll
    scr.Height = 50
    exp.Content = scr

    'Insert Expander into the StackPanel and add it to the
    'Grid
    exp.Width = 200
    exp.HorizontalContentAlignment = HorizontalAlignment.Stretch
    sp.Children.Add(exp)
    myGrid.Children.Add(sp)
End Sub

```

```
<Expander Width="200" HorizontalContentAlignment="Stretch">
    <Expander.Header>
        <BulletDecorator>
            <BulletDecorator.Bullet>
                <Image Width="10" Source="images\icon.jpg"/>
            </BulletDecorator.Bullet>
            <TextBlock Margin="20,0,0,0">My Expander</TextBlock>
        </BulletDecorator>
    </Expander.Header>
    <Expander.Content>
        <ScrollViewer Height="50">
            <TextBlock TextWrapping="Wrap">
                Lorem ipsum dolor sit amet, consectetur adipisicing elit,
                sed do eiusmod tempor incididunt ut labore et dolore magna
                aliqua. Ut enim ad minim veniam, quis nostrud exercitation
                ullamco laboris nisi ut aliquip ex ea commodo consequat.
                Duis aute irure dolor in reprehenderit in voluptate velit
                esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
                occaecat cupidatat non proident, sunt in culpa qui officia
                deserunt mollit anim id est laborum.
            </TextBlock>
        </ScrollViewer>
    </Expander.Content>
</Expander>
```

## Using the Alignment Properties

You can align content by setting the [HorizontalContentAlignment](#) and [VerticalContentAlignment](#) properties on the [Expander](#) control. When you set these properties, the alignment applies to the header and also to the expanded content.

## See also

- [Expander](#)
- [ExpandDirection](#)
- [How-to Topics](#)

# Expander How-to Topics

2 minutes to read • [Edit Online](#)

The topics in this section describe how to use the [Expander](#) control.

## In This Section

[Create an Expander with a ScrollViewer](#)

## Reference

[Expander](#)

## Related Sections

[Expander Overview](#)

# How to: Create an Expander with a ScrollViewer

2 minutes to read • [Edit Online](#)

This example shows how to create an [Expander](#) control that contains complex content, such as an image and text. The example also encloses the content of the [Expander](#) in a [ScrollViewer](#) control.

## Example

The following example shows how to create an [Expander](#). The example uses a [BulletDecorator](#) control, which contains an image and text, in order to define the [Header](#). A [ScrollViewer](#) control provides a method for scrolling the expanded content.

Note that the example sets the [Height](#) property on the [ScrollViewer](#) instead of on the content. If the [Height](#) is set on the content, the [ScrollViewer](#) does not allow the user to scroll the content. The [Width](#) property is set on the [Expander](#) control and this setting applies to the [Header](#) and the expanded content.

```
<Expander Width="200" HorizontalContentAlignment="Stretch">
    <Expander.Header>
        <BulletDecorator>
            <BulletDecorator.Bullet>
                <Image Width="10" Source="images\icon.jpg"/>
            </BulletDecorator.Bullet>
            <TextBlock Margin="20,0,0,0">My Expander</TextBlock>
        </BulletDecorator>
    </Expander.Header>
    <Expander.Content>
        <ScrollViewer Height="50">
            <TextBlock TextWrapping="Wrap">
                Lorem ipsum dolor sit amet, consectetur adipisicing elit,
                sed do eiusmod tempor incididunt ut labore et dolore magna
                aliqua. Ut enim ad minim veniam, quis nostrud exercitation
                ullamco laboris nisi ut aliquip ex ea commodo consequat.
                Duis aute irure dolor in reprehenderit in voluptate velit
                esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
                occaecat cupidatat non proident, sunt in culpa qui officia
                deserunt mollit anim id est laborum.
            </TextBlock>
        </ScrollViewer>
    </Expander.Content>
</Expander>
```

```

//Create Expander object
Expander exp = new Expander();

//Create Bullet Panel for Expander Header
BulletDecorator bp = new BulletDecorator();
Image i = new Image();
BitmapImage bi= new BitmapImage();
bi.UriSource = new Uri(@"pack://application:,,/images/icon.jpg");
i.Source = bi;
i.Width = 10;
bp.Bullet = i;
TextBlock tb = new TextBlock();
tb.Text = "My Expander";
tb.Margin = new Thickness(20,0,0,0);
bp.Child = tb;
exp.Header = bp;

//Create TextBlock with ScrollViewer for Expander Content
StackPanel spScroll = new StackPanel();
TextBlock tbc = new TextBlock();
tbc.Text =
    "Lorem ipsum dolor sit amet, consectetur adipisicing elit," +
    "sed do eiusmod tempor incididunt ut labore et dolore magna" +
    "aliqua. Ut enim ad minim veniam, quis nostrud exercitation" +
    "ullamco laboris nisi ut aliquip ex ea commodo consequat." +
    "Duis aute irure dolor in reprehenderit in voluptate velit" +
    "esse cillum dolore eu fugiat nulla pariatur. Excepteur sint" +
    "occaecat cupidatat non proident, sunt in culpa qui officia" +
    "deserunt mollit anim id est laborum.";
tbc.TextWrapping = TextWrapping.Wrap;

spScroll.Children.Add(tbc);
ScrollViewer scr = new ScrollViewer();
scr.Content = spScroll;
scr.Height = 50;
exp.Content = scr;

exp.Width=200;
exp.HorizontalContentAlignment= HorizontalAlignment.Stretch;

```

## See also

- [Expander](#)
- [Expander Overview](#)
- [How-to Topics](#)

# FlowDocumentPageViewer

2 minutes to read • [Edit Online](#)

The [FlowDocumentPageViewer](#) control is used to view [FlowDocument](#) content on a per page basis. Contrast with the [FlowDocumentScrollView](#), which presents [FlowDocument](#) content in a scrolling viewer.

## See also

- [FlowDocument](#)
- [Flow Document Overview](#)
- [How-to Topics](#)
- [Documents in WPF](#)

# FlowDocumentReader

2 minutes to read • [Edit Online](#)

The [FlowDocumentReader](#) control is used to view [FlowDocument](#) content. It supports multiple viewing modes.

## See also

- [FlowDocumentReader](#)
- [FlowDocumentPageViewer](#)
- [FlowDocumentScrollViewer](#)
- [Documents in WPF](#)
- [Flow Document Overview](#)

# FlowDocumentScrollViewer

2 minutes to read • [Edit Online](#)

The [FlowDocumentScrollViewer](#) control is used to view [FlowDocument](#) content in a scrolling container. Contrast with [FlowDocumentPageViewer](#), which views content on a per page basis.

## See also

- [FlowDocumentReader](#)
- [FlowDocumentPageViewer](#)
- [FlowDocumentScrollViewer](#)
- [FlowDocument](#)
- [Documents in WPF](#)
- [Flow Document Overview](#)

# Frame

2 minutes to read • [Edit Online](#)

The [Frame](#) control supports content navigation within content. [Frame](#) can be hosted by a root element like [Window](#), [NavigationWindow](#), [Page](#), [UserControl](#), [FlowDocument](#), or as an island within a content tree that belongs to a root element.

## Reference

[Frame](#)

## Related Sections

[Navigation Overview](#)

# Grid

2 minutes to read • [Edit Online](#)

The [Grid](#) element is used to precisely position content in rows and columns.

## In This Section

### [How-to Topics](#)

### Reference

[Panel](#)

[Canvas](#)

[DockPanel](#)

[Grid](#)

[StackPanel](#)

[VirtualizingStackPanel](#)

[WrapPanel](#)

## Related Sections

[Layout](#)

[Walkthrough: My first WPF desktop application](#)

[ScrollViewer Overview](#)

# Grid How-to Topics

2 minutes to read • [Edit Online](#)

The topics in this section describe how to position elements using the `Grid` element.

## In This Section

[Build a Standard UI Dialog Box by Using Grid](#)

[Create a Complex Grid](#)

[Create a Grid Element](#)

[Create and Use a GridLengthConverter Object](#)

[Manipulate Columns and Rows by Using ColumnDefinitionsCollections and RowDefinitionsCollections](#)

[Position the Child Elements of a Grid](#)

[Share Sizing Properties Between Grids](#)

## Reference

[Panel](#)

[Canvas](#)

[DockPanel](#)

[Grid](#)

[StackPanel](#)

[VirtualizingStackPanel](#)

[WrapPanel](#)

## Related Sections

[Layout](#)

[Walkthrough: My first WPF desktop application](#)

[ScrollViewer Overview](#)

# How to: Build a Standard UI Dialog Box by Using Grid

3 minutes to read • [Edit Online](#)

This example shows how to create a standard user interface (UI) dialog box by using the **Grid** element.

## Example

The following example creates a dialog box like the **Run** dialog box in the Windows operating system.

The example creates a **Grid** and uses the **ColumnDefinition** and **RowDefinition** classes to define five columns and four rows.

The example then adds and positions an **Image**, `RunIcon.png`, to represent the image that is found in the dialog box. The image is placed in the first column and row of the **Grid** (the upper-left corner).

Next, the example adds a **TextBlock** element to the first column, which spans the remaining columns of the first row. It adds another **TextBlock** element to the second row in the first column, to represent the **Open** text box. A **TextBlock** follows, which represents the data entry area.

Finally, the example adds three **Button** elements to the final row, which represent the **OK**, **Cancel**, and **Browse** events.

```
// Create the Grid.  
grid1 = new Grid();  
grid1.Background = Brushes.Gainsboro;  
grid1.HorizontalAlignment = HorizontalAlignment.Left;  
grid1.VerticalAlignment = VerticalAlignment.Top;  
grid1.ShowGridLines = true;  
grid1.Width = 425;  
grid1.Height = 165;  
  
// Define the Columns.  
colDef1 = new ColumnDefinition();  
colDef1.Width = new GridLength(1, GridUnitType.Auto);  
colDef2 = new ColumnDefinition();  
colDef2.Width = new GridLength(1, GridUnitType.Star);  
colDef3 = new ColumnDefinition();  
colDef3.Width = new GridLength(1, GridUnitType.Star);  
colDef4 = new ColumnDefinition();  
colDef4.Width = new GridLength(1, GridUnitType.Star);  
colDef5 = new ColumnDefinition();  
colDef5.Width = new GridLength(1, GridUnitType.Star);  
grid1.ColumnDefinitions.Add(colDef1);  
grid1.ColumnDefinitions.Add(colDef2);  
grid1.ColumnDefinitions.Add(colDef3);  
grid1.ColumnDefinitions.Add(colDef4);  
grid1.ColumnDefinitions.Add(colDef5);  
  
// Define the Rows.  
rowDef1 = new RowDefinition();  
rowDef1.Height = new GridLength(1, GridUnitType.Auto);  
rowDef2 = new RowDefinition();  
rowDef2.Height = new GridLength(1, GridUnitType.Auto);  
rowDef3 = new RowDefinition();  
rowDef3.Height = new GridLength(1, GridUnitType.Star);  
rowDef4 = new RowDefinition();
```

```

rowDef4.Height = new GridLength(1, GridUnitType.Auto);
grid1.RowDefinitions.Add(rowDef1);
grid1.RowDefinitions.Add(rowDef2);
grid1.RowDefinitions.Add(rowDef3);
grid1.RowDefinitions.Add(rowDef4);

// Add the Image.
img1 = new Image();
img1.Source = new System.Windows.Media.Imaging.BitmapImage(new Uri("runicon.png", UriKind.Relative));
Grid.SetRow(img1, 0);
Grid.SetColumn(img1, 0);

// Add the main application dialog.
txt1 = new TextBlock();
txt1.Text = "Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.";
txt1.TextWrapping = TextWrapping.Wrap;
Grid.SetColumnSpan(txt1, 4);
Grid.SetRow(txt1, 0);
Grid.SetColumn(txt1, 1);

// Add the second text cell to the Grid.
txt2 = new TextBlock();
txt2.Text = "Open:";
Grid.SetRow(txt2, 1);
Grid.SetColumn(txt2, 0);

// Add the TextBox control.
tb1 = new TextBox();
Grid.SetRow(tb1, 1);
Grid.SetColumn(tb1, 1);
Grid.SetColumnSpan(tb1, 5);

// Add the buttons.
button1 = new Button();
button2 = new Button();
button3 = new Button();
button1.Content = "OK";
button2.Content = "Cancel";
button3.Content = "Browse ...";
Grid.SetRow(button1, 3);
Grid.SetColumn(button1, 2);
button1.Margin = new Thickness(10, 0, 10, 15);
button2.Margin = new Thickness(10, 0, 10, 15);
button3.Margin = new Thickness(10, 0, 10, 15);
Grid.SetRow(button2, 3);
Grid.SetColumn(button2, 3);
Grid.SetRow(button3, 3);
Grid.SetColumn(button3, 4);

grid1.Children.Add(img1);
grid1.Children.Add(txt1);
grid1.Children.Add(txt2);
grid1.Children.Add(tb1);
grid1.Children.Add(button1);
grid1.Children.Add(button2);
grid1.Children.Add(button3);

mainWindow.Content = grid1;

```

```

'Create a Grid as the root Panel element.
Dim myGrid As New Grid()
myGrid.Height = 165
myGrid.Width = 425
myGrid.Background = Brushes.Gainsboro
myGrid.ShowGridLines = True

```

```

myGrid.HorizontalAlignment = Windows.HorizontalAlignment.Left
myGrid.VerticalAlignment = Windows.VerticalAlignment.Top

' Define and Add the Rows and Columns.
Dim colDef1 As New ColumnDefinition
colDef1.Width = New GridLength(1, GridUnitType.Auto)
Dim colDef2 As New ColumnDefinition
colDef2.Width = New GridLength(1, GridUnitType.Star)
Dim colDef3 As New ColumnDefinition
colDef3.Width = New GridLength(1, GridUnitType.Star)
Dim colDef4 As New ColumnDefinition
colDef4.Width = New GridLength(1, GridUnitType.Star)
Dim colDef5 As New ColumnDefinition
colDef5.Width = New GridLength(1, GridUnitType.Star)
myGrid.ColumnDefinitions.Add(colDef1)
myGrid.ColumnDefinitions.Add(colDef2)
myGrid.ColumnDefinitions.Add(colDef3)
myGrid.ColumnDefinitions.Add(colDef4)
myGrid.ColumnDefinitions.Add(colDef5)

Dim rowDef1 As New RowDefinition
rowDef1.Height = New GridLength(1, GridUnitType.Auto)
Dim rowDef2 As New RowDefinition
rowDef2.Height = New GridLength(1, GridUnitType.Auto)
Dim rowDef3 As New Controls.RowDefinition
rowDef3.Height = New GridLength(1, GridUnitType.Star)
Dim rowDef4 As New RowDefinition
rowDef4.Height = New GridLength(1, GridUnitType.Auto)
myGrid.RowDefinitions.Add(rowDef1)
myGrid.RowDefinitions.Add(rowDef2)
myGrid.RowDefinitions.Add(rowDef3)
myGrid.RowDefinitions.Add(rowDef4)

' Add the Image.
Dim img1 As New Image
img1.Source = New System.Windows.Media.Imaging.BitmapImage(New Uri("runicon.png", UriKind.Relative))
Grid.SetRow(img1, 0)
Grid.SetColumn(img1, 0)
myGrid.Children.Add(img1)

' Add the main application dialog.
Dim txt1 As New TextBlock
txt1.Text = "Type the name of a program, document, or Internet resource, and Windows will open it for you."
txt1.TextWrapping = TextWrapping.Wrap
Grid.SetColumnSpan(txt1, 4)
Grid.SetRow(txt1, 0)
Grid.SetColumn(txt1, 1)
myGrid.Children.Add(txt1)

' Add the second TextBlock Cell to the Grid.
Dim txt2 As New TextBlock
txt2.Text = "Open:"
Grid.SetRow(txt2, 1)
Grid.SetColumn(txt2, 0)
myGrid.Children.Add(txt2)

' Add the TextBox control.
Dim tb1 As New TextBox
Grid.SetRow(tb1, 1)
Grid.SetColumn(tb1, 1)
Grid.SetColumnSpan(tb1, 5)
myGrid.Children.Add(tb1)

' Add the Button controls.
Dim button1 As New Button
Dim button2 As New Button
Dim button3 As New Button
button1.Content = "OK"
button1.Margin = New Thickness(10, 0, 10, 15)

```

```
button2.Content = "Cancel"
button2.Margin = New Thickness(10, 0, 10, 15)
button3.Content = "Browse ..."
button3.Margin = New Thickness(10, 0, 10, 15)

Grid.SetRow(button1, 3)
Grid.SetColumn(button1, 2)
Grid.SetRow(button2, 3)
Grid.SetColumn(button2, 3)
Grid.SetRow(button3, 3)
Grid.SetColumn(button3, 4)
myGrid.Children.Add(button1)
myGrid.Children.Add(button2)
myGrid.Children.Add(button3)

Me.Content = myGrid
```

## See also

- [Grid](#)
- [GridUnitType](#)
- [Panels Overview](#)
- [How-to Topics](#)

# How to create a complex Grid

2 minutes to read • [Edit Online](#)

This example shows how to use a [Grid](#) control to create a layout that looks like a monthly calendar.

## Example

The following example defines eight rows and eight columns by using the [RowDefinition](#) and [ColumnDefinition](#) classes. It uses the [Grid.ColumnSpan](#) and [Grid.RowSpan](#) attached properties, together with [Rectangle](#) elements, which fill the backgrounds of various columns and rows. This design is possible because more than one element can exist in each cell in a [Grid](#), a principle difference between [Grid](#) and [Table](#).

The example uses vertical gradients to [Fill](#) the columns and rows to improve the visual presentation and readability of the calendar. Styled [TextBlock](#) elements represent the dates and days of the week. [TextBlock](#) elements are absolutely positioned within their cells by using the [Margin](#) property and alignment properties that are defined within the style for the application.

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      WindowTitle="Complex Grid Example">
<Border BorderBrush="Black">

    <Grid ShowGridLines="false" Background="White">
        <Grid.Resources>
            <Style TargetType="{x:Type ColumnDefinition}">
                <Setter Property="Width" Value="30"/>
            </Style>
            <Style TargetType="{x:Type Rectangle}">
                <Setter Property="RadiusX" Value="6"/>
                <Setter Property="RadiusY" Value="6"/>
            </Style>
            <Style x:Key="DayOfWeek">
                <Setter Property="Grid.Row" Value="1"/></Setter>
                <Setter Property="TextBlock.Margin" Value="5"/></Setter>
            </Style>
            <Style x:Key="OneDate">
                <Setter Property="TextBlock.Margin" Value="5"/></Setter>
            </Style>
        </Grid.Resources>
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
            <ColumnDefinition/>
            <ColumnDefinition/>
            <ColumnDefinition/>
            <ColumnDefinition/>
            <ColumnDefinition/>
            <!-- This column will receive all remaining width -->
            <ColumnDefinition Width="*"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <!-- This row will receive all remaining Height -->
        </Grid.RowDefinitions>
    </Grid>
</Border>
</Page>
```

```

<RowDefinition/>
</Grid.RowDefinitions>

<!-- These Rectangles constitute the backgrounds of the various Rows and Columns --&gt;

&lt;Rectangle Grid.ColumnSpan="7" Fill="#73B2F5"/&gt;
&lt;Rectangle Grid.Row="1" Grid.RowSpan="6" Fill="#73B2F5"/&gt;
&lt;Rectangle Grid.Column="6" Grid.Row="1" Grid.RowSpan="6" Fill="#73B2F5"/&gt;
&lt;Rectangle Grid.Column="1" Grid.Row="1" Grid.ColumnSpan="5" Grid.RowSpan="6"
Fill="#efefef"/&gt;

<!-- Month row --&gt;
&lt;TextBlock Grid.ColumnSpan="7" Margin="0,5,0,5" HorizontalAlignment="Center"&gt;
    January 2004
&lt;/TextBlock&gt;

<!-- Draws a separator under the days-of-the-week row --&gt;

&lt;Rectangle Grid.Row="1" Grid.ColumnSpan="7"
Fill="Black" RadiusX="1" RadiusY="1" Height="2" Margin="0,20,0,0"/&gt;

<!-- Day-of-the-week row --&gt;
&lt;TextBlock Grid.Column="0" Style="{StaticResource DayOfWeek}"&gt;Sun&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="1" Style="{StaticResource DayOfWeek}"&gt;Mon&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="2" Style="{StaticResource DayOfWeek}"&gt;Tue&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="3" Style="{StaticResource DayOfWeek}"&gt;Wed&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="4" Style="{StaticResource DayOfWeek}"&gt;Thu&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="5" Style="{StaticResource DayOfWeek}"&gt;Fri&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="6" Style="{StaticResource DayOfWeek}"&gt;Sat&lt;/TextBlock&gt;

<!-- Dates go here --&gt;
&lt;TextBlock Grid.Column="4" Grid.Row="2" Style="{StaticResource OneDate}"&gt;1&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="5" Grid.Row="2" Style="{StaticResource OneDate}"&gt;2&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="6" Grid.Row="2" Style="{StaticResource OneDate}"&gt;3&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="0" Grid.Row="3" Style="{StaticResource OneDate}"&gt;4&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="1" Grid.Row="3" Style="{StaticResource OneDate}"&gt;5&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="2" Grid.Row="3" Style="{StaticResource OneDate}"&gt;6&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="3" Grid.Row="3" Style="{StaticResource OneDate}"&gt;7&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="4" Grid.Row="3" Style="{StaticResource OneDate}"&gt;8&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="5" Grid.Row="3" Style="{StaticResource OneDate}"&gt;9&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="6" Grid.Row="3" Style="{StaticResource OneDate}"&gt;10&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="0" Grid.Row="4" Style="{StaticResource OneDate}"&gt;11&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="1" Grid.Row="4" Style="{StaticResource OneDate}"&gt;12&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="2" Grid.Row="4" Style="{StaticResource OneDate}"&gt;13&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="3" Grid.Row="4" Style="{StaticResource OneDate}"&gt;14&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="4" Grid.Row="4" Style="{StaticResource OneDate}"&gt;15&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="5" Grid.Row="4" Style="{StaticResource OneDate}"&gt;16&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="6" Grid.Row="4" Style="{StaticResource OneDate}"&gt;17&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="0" Grid.Row="5" Style="{StaticResource OneDate}"&gt;18&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="1" Grid.Row="5" Style="{StaticResource OneDate}"&gt;19&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="2" Grid.Row="5" Style="{StaticResource OneDate}"&gt;20&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="3" Grid.Row="5" Style="{StaticResource OneDate}"&gt;21&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="4" Grid.Row="5" Style="{StaticResource OneDate}"&gt;22&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="5" Grid.Row="5" Style="{StaticResource OneDate}"&gt;23&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="6" Grid.Row="5" Style="{StaticResource OneDate}"&gt;24&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="0" Grid.Row="6" Style="{StaticResource OneDate}"&gt;25&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="1" Grid.Row="6" Style="{StaticResource OneDate}"&gt;26&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="2" Grid.Row="6" Style="{StaticResource OneDate}"&gt;27&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="3" Grid.Row="6" Style="{StaticResource OneDate}"&gt;28&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="4" Grid.Row="6" Style="{StaticResource OneDate}"&gt;29&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="5" Grid.Row="6" Style="{StaticResource OneDate}"&gt;30&lt;/TextBlock&gt;
&lt;TextBlock Grid.Column="6" Grid.Row="6" Style="{StaticResource OneDate}"&gt;31&lt;/TextBlock&gt;
&lt;/Grid&gt;
&lt;/Border&gt;
&lt;/Page&gt;
</pre>

```

The following image shows the resulting control, a customizable calendar:



## See also

- [System.Windows.Controls.Grid](#)
- [System.Windows.Documents.TableCell](#)
- [Painting with Solid Colors and Gradients Overview](#)
- [Panels Overview](#)
- [Table Overview](#)

# How to: Create a Grid Element

3 minutes to read • [Edit Online](#)

## Example

The following example shows how to create and use an instance of [Grid](#) by using either Extensible Application Markup Language (XAML) or code. This example uses three [ColumnDefinition](#) objects and three [RowDefinition](#) objects to create a grid that has nine cells, such as in a worksheet. Each cell contains a [TextBlock](#) element that represents data, and the top row contains a [TextBlock](#) with the [ColumnSpan](#) property applied. To show the boundaries of each cell, the [ShowGridLines](#) property is enabled.

```
// Create the application's main window
mainWindow = new Window();
mainWindow.Title = "Grid Sample";

// Create the Grid
Grid myGrid = new Grid();
myGrid.Width = 250;
myGrid.Height = 100;
myGrid.HorizontalAlignment = HorizontalAlignment.Left;
myGrid.VerticalAlignment = VerticalAlignment.Top;
myGrid.ShowGridLines = true;

// Define the Columns
ColumnDefinition colDef1 = new ColumnDefinition();
ColumnDefinition colDef2 = new ColumnDefinition();
ColumnDefinition colDef3 = new ColumnDefinition();
myGrid.ColumnDefinitions.Add(colDef1);
myGrid.ColumnDefinitions.Add(colDef2);
myGrid.ColumnDefinitions.Add(colDef3);

// Define the Rows
RowDefinition rowDef1 = new RowDefinition();
RowDefinition rowDef2 = new RowDefinition();
RowDefinition rowDef3 = new RowDefinition();
RowDefinition rowDef4 = new RowDefinition();
myGrid.RowDefinitions.Add(rowDef1);
myGrid.RowDefinitions.Add(rowDef2);
myGrid.RowDefinitions.Add(rowDef3);
myGrid.RowDefinitions.Add(rowDef4);

// Add the first text cell to the Grid
TextBlock txt1 = new TextBlock();
txt1.Text = "2005 Products Shipped";
txt1.FontSize = 20;
txt1.FontWeight = FontWeights.Bold;
Grid.SetColumnSpan(txt1, 3);
Grid.SetRow(txt1, 0);

// Add the second text cell to the Grid
TextBlock txt2 = new TextBlock();
txt2.Text = "Quarter 1";
txt2.FontSize = 12;
txt2.FontWeight = FontWeights.Bold;
Grid.SetRow(txt2, 1);
Grid.SetColumn(txt2, 0);

// Add the third text cell to the Grid
TextBlock txt3 = new TextBlock();
txt3.Text = "Quarter 2";
```

```

txt3.Text = "Quarter 2";
txt3.FontSize = 12;
txt3.FontWeight = FontWeights.Bold;
Grid.SetRow(txt3, 1);
Grid.SetColumn(txt3, 1);

// Add the fourth text cell to the Grid
TextBlock txt4 = new TextBlock();
txt4.Text = "Quarter 3";
txt4.FontSize = 12;
txt4.FontWeight = FontWeights.Bold;
Grid.SetRow(txt4, 1);
Grid.SetColumn(txt4, 2);

// Add the sixth text cell to the Grid
TextBlock txt5 = new TextBlock();
Double db1 = new Double();
db1 = 50000;
txt5.Text = db1.ToString();
Grid.SetRow(txt5, 2);
Grid.SetColumn(txt5, 0);

// Add the seventh text cell to the Grid
TextBlock txt6 = new TextBlock();
Double db2 = new Double();
db2 = 100000;
txt6.Text = db2.ToString();
Grid.SetRow(txt6, 2);
Grid.SetColumn(txt6, 1);

// Add the final text cell to the Grid
TextBlock txt7 = new TextBlock();
Double db3 = new Double();
db3 = 150000;
txt7.Text = db3.ToString();
Grid.SetRow(txt7, 2);
Grid.SetColumn(txt7, 2);

// Total all Data and Span Three Columns
TextBlock txt8 = new TextBlock();
txt8.FontSize = 16;
txt8.FontWeight = FontWeights.Bold;
txt8.Text = "Total Units: " + (db1 + db2 + db3).ToString();
Grid.SetRow(txt8, 3);
Grid.SetColumnSpan(txt8, 3);

// Add the TextBlock elements to the Grid Children collection
myGrid.Children.Add(txt1);
myGrid.Children.Add(txt2);
myGrid.Children.Add(txt3);
myGrid.Children.Add(txt4);
myGrid.Children.Add(txt5);
myGrid.Children.Add(txt6);
myGrid.Children.Add(txt7);
myGrid.Children.Add(txt8);

// Add the Grid as the Content of the Parent Window Object
mainWindow.Content = myGrid;
mainWindow.Show ();

```

```

Public Sub New()
    WindowTitle = "Grid Sample"
    'Create a Grid as the root Panel element
    Dim myGrid As New Grid()
    myGrid.Height = 100
    myGrid.Width = 250
    myGrid.ShowGridLines = True

```

```
myGrid.ShowGridLines = True
myGrid.HorizontalAlignment = Windows.HorizontalAlignment.Left
myGrid.VerticalAlignment = Windows.VerticalAlignment.Top

' Define and Add the Rows and Columns
Dim colDef1 As New ColumnDefinition
Dim colDef2 As New ColumnDefinition
Dim colDef3 As New ColumnDefinition
myGrid.ColumnDefinitions.Add(colDef1)
myGrid.ColumnDefinitions.Add(colDef2)
myGrid.ColumnDefinitions.Add(colDef3)

Dim rowDef1 As New RowDefinition
Dim rowDef2 As New RowDefinition
Dim rowDef3 As New RowDefinition
Dim rowDef4 As New RowDefinition
myGrid.RowDefinitions.Add(rowDef1)
myGrid.RowDefinitions.Add(rowDef2)
myGrid.RowDefinitions.Add(rowDef3)
myGrid.RowDefinitions.Add(rowDef4)

Dim txt1 As New TextBlock
txt1.Text = "2004 Products Shipped"
txt1.FontSize = 20
txt1.FontWeight = FontWeights.Bold
Grid.SetColumnSpan(txt1, 3)
Grid.SetRow(txt1, 0)
myGrid.Children.Add(txt1)

Dim txt2 As New TextBlock
txt2.Text = "Quarter 1"
txt2.FontSize = 12
txt2.FontWeight = FontWeights.Bold
Grid.SetRow(txt2, 1)
Grid.SetColumn(txt2, 0)
myGrid.Children.Add(txt2)

Dim txt3 As New TextBlock
txt3.Text = "Quarter 2"
txt3.FontSize = 12
txt3.FontWeight = FontWeights.Bold
Grid.SetRow(txt3, 1)
Grid.SetColumn(txt3, 1)
myGrid.Children.Add(txt3)

Dim txt4 As New TextBlock
txt4.Text = "Quarter 3"
txt4.FontSize = 12
txt4.FontWeight = FontWeights.Bold
Grid.SetRow(txt4, 1)
Grid.SetColumn(txt4, 2)
myGrid.Children.Add(txt4)

Dim txt5 As New TextBlock
txt5.Text = "50,000"
Grid.SetRow(txt5, 2)
Grid.SetColumn(txt5, 0)
myGrid.Children.Add(txt5)

Dim txt6 As New Controls.TextBlock
txt6.Text = "100,000"
Grid.SetRow(txt6, 2)
Grid.SetColumn(txt6, 1)
myGrid.Children.Add(txt6)

Dim txt7 As New TextBlock
txt7.Text = "150,000"
Grid.SetRow(txt7, 2)
Grid.SetColumn(txt7, 2)
```

```

myGrid.Children.Add(txt7)

' Add the final TextBlock Cell to the Grid
Dim txt8 As New TextBlock
txt8.FontSize = 16
txt8.FontWeight = FontWeights.Bold
txt8.Text = "Total Units: 300000"
Grid.SetRow(txt8, 3)
Grid.SetColumnSpan(txt8, 3)
myGrid.Children.Add(txt8)

Me.Content = myGrid
End Sub

```

```

<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" WindowTitle="Grid Sample">
    <Grid VerticalAlignment="Top" HorizontalAlignment="Left" ShowGridLines="True" Width="250" Height="100">
        <Grid.ColumnDefinitions>
            <ColumnDefinition />
            <ColumnDefinition />
            <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
        </Grid.RowDefinitions>

        <TextBlock FontSize="20" FontWeight="Bold" Grid.ColumnSpan="3" Grid.Row="0">2005 Products
Shipped</TextBlock>
        <TextBlock FontSize="12" FontWeight="Bold" Grid.Row="1" Grid.Column="0">Quarter 1</TextBlock>
        <TextBlock FontSize="12" FontWeight="Bold" Grid.Row="1" Grid.Column="1">Quarter 2</TextBlock>
        <TextBlock FontSize="12" FontWeight="Bold" Grid.Row="1" Grid.Column="2">Quarter 3</TextBlock>
        <TextBlock Grid.Row="2" Grid.Column="0">50000</TextBlock>
        <TextBlock Grid.Row="2" Grid.Column="1">100000</TextBlock>
        <TextBlock Grid.Row="2" Grid.Column="2">150000</TextBlock>
        <TextBlock FontSize="16" FontWeight="Bold" Grid.ColumnSpan="3" Grid.Row="3">Total Units:
300000</TextBlock>
    </Grid>
</Page>

```

Either approach will generate a user interface that looks much the same, like the one below.

Window1

## 2018 Products Shipped

Quarter 1	Quarter 2	Quarter 3
-----------	-----------	-----------

50000	100000	150000
-------	--------	--------

**Total Units: 300000**

### See also

- [Grid](#)
- [Panels Overview](#)

# How to: Create and Use a GridLengthConverter Object

2 minutes to read • [Edit Online](#)

## Example

The following example shows how to create and use an instance of `GridLengthConverter`. The example defines a custom method called `changeCol`, which passes the `ListBoxItem` to a `GridLengthConverter` that converts the `Content` of a `ListBoxItem` to an instance of `GridLength`. The converted value is then passed back as the value of the `Width` property of the `ColumnDefinition` element.

The example also defines a second custom method, called `changeColVal`. This custom method converts the `Value` of a `Slider` to a `String` and then passes that value back to the `ColumnDefinition` as the `Width` of the element.

Note that a separate Extensible Application Markup Language (XAML) file defines the contents of a `ListBoxItem`.

```
private void changeColVal(object sender, RoutedEventArgs e)
{
    txt1.Text = "Current Grid Column is " + hs1.Value.ToString();
}

private void changeCol(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    GridLengthConverter myGridLengthConverter = new GridLengthConverter();
    if (hs1.Value == 0)
    {
        GridLength gl1 = (GridLength)myGridLengthConverter.ConvertFromString(li.Content.ToString());
        col1.Width = gl1;
    }
    else if (hs1.Value == 1)
    {
        GridLength gl2 = (GridLength)myGridLengthConverter.ConvertFromString(li.Content.ToString());
        col2.Width = gl2;
    }
    else if (hs1.Value == 2)
    {
        GridLength gl3 = (GridLength)myGridLengthConverter.ConvertFromString(li.Content.ToString());
        col3.Width = gl3;
    }
}
```

```
Private Sub changeColVal(ByVal sender As Object, ByVal args As RoutedEventArgs)(Of Double))

    txt1.Text = "Current Grid Column is " + hs1.Value.ToString()
End Sub

Private Sub changeCol(ByVal sender As Object, ByVal args As SelectionChangedEventArgs)

    Dim li1 As ListBoxItem = CType(CType(sender, ListBox).SelectedItem, ListBoxItem)
    Dim myGridLengthConverter As System.Windows.GridLengthConverter = New System.Windows.GridLengthConverter()
    If (hs1.Value = 0) Then

        Dim g11 As GridLength = CType(myGridLengthConverter.ConvertFromString(li1.Content.ToString()),
GridLength)
        col1.Width = g11

    ElseIf (hs1.Value = 1) Then

        Dim g12 As GridLength = CType(myGridLengthConverter.ConvertFromString(li1.Content.ToString()),
GridLength)
        col2.Width = g12

    ElseIf (hs1.Value = 2) Then

        Dim g13 As GridLength = CType(myGridLengthConverter.ConvertFromString(li1.Content.ToString()),
GridLength)
        col3.Width = g13
    End If
End Sub
```

## See also

- [GridLengthConverter](#)
- [GridLength](#)

# How to: Manipulate Columns and Rows by Using ColumnDefinitionsCollections and RowDefinitionsCollections

4 minutes to read • [Edit Online](#)

This example shows how to use the methods in the [ColumnDefinitionCollection](#) and [RowDefinitionCollection](#) classes to perform actions like adding, clearing, or counting the contents of rows or columns. For example, you can [Add](#), [Clear](#), or [Count](#) the items that are included in a [ColumnDefinition](#) or [RowDefinition](#).

## Example

The following example creates a [Grid](#) element with a [Name](#) of `grid1`. The [Grid](#) contains a [StackPanel](#) that holds [Button](#) elements, each controlled by a different collection method. When you click a [Button](#), it activates a method call in the code-behind file.

```
<Grid DockPanel.Dock="Top" HorizontalAlignment="Left" Name="grid1" ShowGridLines="true" Width="625" Height="400" Background="#b0e0e6">
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <StackPanel HorizontalAlignment="Left" Orientation="Horizontal" Width="625" DockPanel.Dock="Top">
        <Button Width="125" Click="addCol">Add Column</Button>
        <Button Width="125" Click="addRow">Add Row</Button>
        <Button Width="125" Click="clearCol">Clear All Columns</Button>
        <Button Width="125" Click="clearRow">Clear All Rows</Button>
        <Button Width="125" Click="removeCol">Remove One Column</Button>
    </StackPanel>
    <StackPanel HorizontalAlignment="Left" Orientation="Horizontal" Width="625" DockPanel.Dock="Top">
        <Button Width="125" Click="removeRow">Remove One Row</Button>
        <Button Width="125" Click="colCount">How Many Columns?</Button>
        <Button Width="125" Click="rowCount">How Many Rows?</Button>
        <Button Width="125" Click="rem5Col">Remove 5 Columns</Button>
        <Button Width="125" Click="rem5Row">Remove 5 Rows</Button>
    </StackPanel>
    <StackPanel HorizontalAlignment="Left" Orientation="Horizontal" Width="625" DockPanel.Dock="Top">
        <Button Width="125" Click="containsRow">Contains Row?</Button>
        <Button Width="125" Click="containsCol">Contains Column?</Button>
        <Button Width="125" Click="insertRowAt">Insert Row</Button>
        <Button Width="125" Click="insertColAt">Insert Column</Button>
        <Button Width="125" Click="colReadOnly">IsReadOnly?</Button>
    </StackPanel>
</Grid>
```

This example defines a series of custom methods, each corresponding to a [Click](#) event in the Extensible Application Markup Language (XAML) file. You can change the number of columns and rows in the [Grid](#) in several ways, which includes adding or removing rows and columns; and counting the total number of rows and columns. To prevent [ArgumentOutOfRangeException](#) and [ArgumentException](#) exceptions, you can use the error-checking

functionality that the [RemoveRange](#) method provides.

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;

namespace columndefinitions_grid
{
    public partial class Window1 : Window
    {
        RowDefinition rowDef1;
        ColumnDefinition colDef1;

        private void addCol(object sender, RoutedEventArgs e)
        {
            colDef1 = new ColumnDefinition();
            grid1.ColumnDefinitions.Add(colDef1);
        }

        private void addRow(object sender, RoutedEventArgs e)
        {
            rowDef1 = new RowDefinition();
            grid1.RowDefinitions.Add(rowDef1);
        }

        private void clearCol(object sender, RoutedEventArgs e)
        {
            grid1.ColumnDefinitions.Clear();
        }

        private void clearRow(object sender, RoutedEventArgs e)
        {
            grid1.RowDefinitions.Clear();
        }

        private void removeCol(object sender, RoutedEventArgs e)
        {
            if (grid1.ColumnDefinitions.Count <= 0)
            {
                tp1.Text = "No More Columns to Remove!";
            }
            else
            {
                grid1.ColumnDefinitions.RemoveAt(0);
            }
        }

        private void removeRow(object sender, RoutedEventArgs e)
        {
            if (grid1.RowDefinitions.Count <= 0)
            {
                tp1.Text = "No More Rows to Remove!";
            }
            else
            {
                grid1.RowDefinitions.RemoveAt(0);
            }
        }

        private void colCount(object sender, RoutedEventArgs e)
        {
            tp2.Text = "The current number of Columns is: " + grid1.ColumnDefinitions.Count;
        }

        private void rowCount(object sender, RoutedEventArgs e)
        {
            tp2.Text = "The current number of Rows is: " + grid1.RowDefinitions.Count;
        }
    }
}
```

```

}

private void rem5Col(object sender, RoutedEventArgs e)
{
    if (grid1.ColumnDefinitions.Count < 5)
    {
        tp1.Text = "There aren't five Columns to Remove!";
    }
    else
    {
        grid1.ColumnDefinitions.RemoveRange(0,5);
    }
}

private void rem5Row(object sender, RoutedEventArgs e)
{
    if (grid1.RowDefinitions.Count < 5)
    {
        tp1.Text = "There aren't five Rows to Remove!";
    }
    else
    {
        grid1.RowDefinitions.RemoveRange(0, 5);
    }
}

private void containsRow(object sender, RoutedEventArgs e)
{
    if (grid1.RowDefinitions.Contains(rowDef1))
    {
        tp2.Text = "Grid Contains RowDefinition rowDef1";
    }
    else
    {
        tp2.Text = "Grid Does Not Contain RowDefinition rowDef1";
    }
}

private void containsCol(object sender, RoutedEventArgs e)
{
    if (grid1.ColumnDefinitions.Contains(colDef1))
    {
        tp3.Text = "Grid Contains ColumnDefinition colDef1";
    }
    else
    {
        tp3.Text = "Grid Does Not Contain ColumnDefinition colDef1";
    }
}

private void colReadOnly(object sender, RoutedEventArgs e)
{
    tp4.Text = "RowDefinitionsCollection IsReadOnly?: " + grid1.RowDefinitions.IsReadOnly.ToString();
    tp5.Text = "ColumnDefinitionsCollection IsReadOnly?: " +
grid1.ColumnDefinitions.IsReadOnly.ToString();
}

private void insertRowAt(object sender, RoutedEventArgs e)
{
    rowDef1 = new RowDefinition();
    grid1.RowDefinitions.Insert(grid1.RowDefinitions.Count, rowDef1);
    tp1.Text = "RowDefinition added at index position " +
grid1.RowDefinitions.IndexOf(rowDef1).ToString();
}

private void insertColAt(object sender, RoutedEventArgs e)
{
    colDef1 = new ColumnDefinition();
    grid1.ColumnDefinitions.Insert(grid1.ColumnDefinitions.Count, colDef1);
}

```

```

        tp2.Text = "ColumnDefinition added at index position " +
grid1.ColumnDefinitions.IndexOf(colDef1).ToString();
    }
}
}

```

```

Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Documents

Namespace SDKSample

'@ <summary>
'@ Interaction logic for Window1.xaml
'@ </summary>

Partial Public Class Window1
Inherits Window

Dim rowDef1 As New RowDefinition
Dim colDef1 As New ColumnDefinition

Private Sub addCol(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Dim colDef1 As New ColumnDefinition
    grid1.ColumnDefinitions.Add(colDef1)
End Sub

Private Sub addRow(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Dim rowDef1 As New RowDefinition()
    grid1.RowDefinitions.Add(rowDef1)
End Sub

Private Sub clearCol(ByVal sender As Object, ByVal e As RoutedEventArgs)
    grid1.ColumnDefinitions.Clear()
End Sub

Private Sub clearRow(ByVal sender As Object, ByVal e As RoutedEventArgs)
    grid1.RowDefinitions.Clear()
End Sub

Private Sub removeCol(ByVal sender As Object, ByVal e As RoutedEventArgs)

    If (grid1.ColumnDefinitions.Count <= 0) Then
        tp1.Text = "No More Columns to Remove!"
    Else
        grid1.ColumnDefinitions.RemoveAt(0)
    End If
End Sub

Private Sub removeRow(ByVal sender As Object, ByVal e As RoutedEventArgs)
    If (grid1.RowDefinitions.Count <= 0) Then
        tp1.Text = "No More Rows to Remove!"
    Else
        grid1.RowDefinitions.RemoveAt(0)
    End If
End Sub

Private Sub colCount(ByVal sender As Object, ByVal e As RoutedEventArgs)
    tp2.Text = "The current number of Columns is: " + grid1.ColumnDefinitions.Count.ToString()
End Sub

Private Sub rowCount(ByVal sender As Object, ByVal e As RoutedEventArgs)
    tp2.Text = "The current number of Rows is: " + grid1.RowDefinitions.Count.ToString()
End Sub

Private Sub rem5Col(ByVal sender As Object, ByVal e As RoutedEventArgs)
    If (grid1.ColumnDefinitions.Count < 5) Then

```

```

        tp1.Text = "There aren't five Columns to Remove!"
    Else
        grid1.ColumnDefinitions.RemoveRange(0, 5)
    End If
End Sub

Private Sub rem5Row(ByVal sender As Object, ByVal e As RoutedEventArgs)
    If (grid1.RowDefinitions.Count < 5) Then
        tp1.Text = "There aren't five Rows to Remove!"
    Else
        grid1.RowDefinitions.RemoveRange(0, 5)
    End If
End Sub

Private Sub containsRow(ByVal sender As Object, ByVal e As RoutedEventArgs)
    If (grid1.RowDefinitions.Contains(rowDef1)) Then
        tp2.Text = "Grid Contains RowDefinition rowDef1"
    Else
        tp2.Text = "Grid Does Not Contain RowDefinition rowDef1"
    End If
End Sub

Private Sub containsCol(ByVal sender As Object, ByVal e As RoutedEventArgs)
    If (grid1.ColumnDefinitions.Contains(colDef1)) Then
        tp3.Text = "Grid Contains ColumnDefinition colDef1"
    Else
        tp3.Text = "Grid Does Not Contain ColumnDefinition colDef1"
    End If
End Sub

Private Sub colReadOnly(ByVal sender As Object, ByVal e As RoutedEventArgs)
    tp4.Text = "RowDefinitionsCollection IsReadOnly?: " + grid1.RowDefinitions.IsReadOnly.ToString()
    tp5.Text = "ColumnDefinitionsCollection IsReadOnly?: " +
grid1.ColumnDefinitions.IsReadOnly.ToString()
End Sub

Private Sub insertRowAt(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Dim rowDef1 As New RowDefinition
    grid1.RowDefinitions.Insert(grid1.RowDefinitions.Count, rowDef1)
    tp1.Text = "RowDefinition added at index position " +
grid1.RowDefinitions.IndexOf(rowDef1).ToString()
End Sub

Private Sub insertColAt(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Dim colDef1 As New ColumnDefinition()
    grid1.ColumnDefinitions.Insert(grid1.ColumnDefinitions.Count, colDef1)
    tp2.Text = "ColumnDefinition added at index position " +
grid1.ColumnDefinitions.IndexOf(colDef1).ToString()
End Sub

End Class
End Namespace

```

## See also

- [Grid](#)
- [ColumnDefinitionCollection](#)
- [RowDefinitionCollection](#)

# How to: Position the Child Elements of a Grid

3 minutes to read • [Edit Online](#)

This example shows how to use the get and set methods that are defined on [Grid](#) to position child elements.

## Example

The following example defines a parent [Grid](#) element (`grid1`) that has three columns and three rows. A child [Rectangle](#) element (`rect1`) is added to the [Grid](#) in column position zero, row position zero. [Button](#) elements represent methods that can be called to reposition the [Rectangle](#) element within the [Grid](#). When a user clicks a button, the related method is activated.

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="grid_getset_methods.Window1"
    Title="Grid Methods Sample">
    <Border BorderBrush="Black" Background="White" BorderThickness="2">
        <DockPanel VerticalAlignment="Top" HorizontalAlignment="Left" Margin="10">
            <TextBlock FontSize="20" FontWeight="Bold" DockPanel.Dock="Top">Grid Methods Sample</TextBlock>
            <TextBlock DockPanel.Dock="Top">Click the buttons on the left to reposition the Rectangle below using
methods defined on Grid.</TextBlock>
            <Grid Margin="0,10,15,0" DockPanel.Dock="Left">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition/>
                    <ColumnDefinition/>
                </Grid.ColumnDefinitions>
                <Grid.RowDefinitions>
                    <RowDefinition/>
                </Grid.RowDefinitions>
                <!-- <Snippet1> -->
                <StackPanel Grid.Column="0" Grid.Row="0" HorizontalAlignment="Left" Orientation="Vertical">
                    <Button Click="setCol0">Move Rectangle to Column 0</Button>
                    <Button Click="setCol1">Move Rectangle to Column 1</Button>
                    <Button Click="setCol2" Margin="0,0,0,10">Move Rectangle to Column 2</Button>
                    <Button Click="setRow0">Move Rectangle to Row 0</Button>
                    <Button Click="setRow1">Move Rectangle to Row 1</Button>
                    <Button Click="setRow2" Margin="0,0,0,10">Move Rectangle to Row 2</Button>
                    <Button Click="setColspan">Span All Columns</Button>
                    <Button Click="setRowspan">Span All Rows</Button>
                    <Button Click="clearAll">Clear All</Button>
                </StackPanel>
            </Grid>
            <Grid DockPanel.Dock="Top" Margin="0,10,15,0" HorizontalAlignment="Left" Name="grid1">
                ShowGridLines="True" Width="400" Height="400" Background="LightSteelBlue">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition/>
                        <ColumnDefinition/>
                        <ColumnDefinition/>
                    </Grid.ColumnDefinitions>
                    <Grid.RowDefinitions>
                        <RowDefinition/>
                        <RowDefinition/>
                        <RowDefinition/>
                    </Grid.RowDefinitions>
                    <Rectangle Name="rect1" Fill="Silver" Grid.Column="0" Grid.Row="0"/>
                    <TextBlock FontSize="15" HorizontalAlignment="Right" VerticalAlignment="Bottom" Grid.Column="0"
Grid.Row="0" Margin="5">Column 0, Row 0</TextBlock>
                    <TextBlock FontSize="15" HorizontalAlignment="Right" VerticalAlignment="Bottom" Grid.Column="1"
Grid.Row="0" Margin="5">Column 1, Row 0</TextBlock>
                </Grid>
            </Grid>
        </DockPanel>
    </Border>
</Window>
```

```

Grid.Row="0" Margin="5">>Column 1, Row 0</TextBlock>
    <TextBlock FontSize="15" HorizontalAlignment="Right" VerticalAlignment="Bottom" Grid.Column="2">
Grid.Row="0" Margin="5">>Column 2, Row 0</TextBlock>
    <TextBlock FontSize="15" HorizontalAlignment="Right" VerticalAlignment="Bottom" Grid.Column="0">
Grid.Row="1" Margin="5">>Column 0, Row 1</TextBlock>
    <TextBlock FontSize="15" HorizontalAlignment="Right" VerticalAlignment="Bottom" Grid.Column="1">
Grid.Row="1" Margin="5">>Column 1, Row 1</TextBlock>
    <TextBlock FontSize="15" HorizontalAlignment="Right" VerticalAlignment="Bottom" Grid.Column="2">
Grid.Row="1" Margin="5">>Column 2, Row 1</TextBlock>
    <TextBlock FontSize="15" HorizontalAlignment="Right" VerticalAlignment="Bottom" Grid.Column="0">
Grid.Row="2" Margin="5">>Column 0, Row 2</TextBlock>
    <TextBlock FontSize="15" HorizontalAlignment="Right" VerticalAlignment="Bottom" Grid.Column="1">
Grid.Row="2" Margin="5">>Column 1, Row 2</TextBlock>
    <TextBlock FontSize="15" HorizontalAlignment="Right" VerticalAlignment="Bottom" Grid.Column="2">
Grid.Row="2" Margin="5">>Column 2, Row 2</TextBlock>
</Grid>
<!-- </Snippet1> -->

<TextBlock DockPanel.Dock="Top" Name="txt1"/>
<TextBlock DockPanel.Dock="Top" Name="txt2"/>
<TextBlock DockPanel.Dock="Top" Name="txt3"/>
<TextBlock DockPanel.Dock="Top" Name="txt4"/>
</DockPanel>
</Border>
</Window>

```

The following code-behind example handles the methods that the button [Click](#) events raise. The example writes these method calls to [TextBlock](#) elements that use related get methods to output the new property values as strings.

```
private void setCol0(object sender, RoutedEventArgs e)
{
    Grid.SetColumn(rect1, 0);
    txt1.Text = "Rectangle is in Column " + Grid.GetColumn(rect1).ToString();
}
private void setCol1(object sender, RoutedEventArgs e)
{
    Grid.SetColumn(rect1, 1);
    txt1.Text = "Rectangle is in Column " + Grid.GetColumn(rect1).ToString();
}
private void setCol2(object sender, RoutedEventArgs e)
{
    Grid.SetColumn(rect1, 2);
    txt1.Text = "Rectangle is in Column " + Grid.GetColumn(rect1).ToString();
}
private void setRow0(object sender, RoutedEventArgs e)
{
    Grid.SetRow(rect1, 0);
    txt2.Text = "Rectangle is in Row " + Grid.GetRow(rect1).ToString();
}
private void setRow1(object sender, RoutedEventArgs e)
{
    Grid.SetRow(rect1, 1);
    txt2.Text = "Rectangle is in Row " + Grid.GetRow(rect1).ToString();
}
private void setRow2(object sender, RoutedEventArgs e)
{
    Grid.SetRow(rect1, 2);
    txt2.Text = "Rectangle is in Row " + Grid.GetRow(rect1).ToString();
}
private void setColspan(object sender, RoutedEventArgs e)
{
    Grid.SetColumnSpan(rect1, 3);
    txt3.Text = "ColumnSpan is set to " + Grid.GetColumnSpan(rect1).ToString();
}
private void setRowspan(object sender, RoutedEventArgs e)
{
    Grid.SetRowSpan(rect1, 3);
    txt4.Text = "RowSpan is set to " + Grid.GetRowSpan(rect1).ToString();
}
```

```

Private Sub setCol0(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Grid.SetColumn(rect1, 0)
    txt1.Text = "Rectangle is in Column " + Grid.GetColumn(rect1).ToString()
End Sub

Private Sub setCol1(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Grid.SetColumn(rect1, 1)
    txt1.Text = "Rectangle is in Column " + Grid.GetColumn(rect1).ToString()
End Sub

Private Sub setCol2(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Grid.SetColumn(rect1, 2)
    txt1.Text = "Rectangle is in Column " + Grid.GetColumn(rect1).ToString()
End Sub

Private Sub setRow0(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Grid.SetRow(rect1, 0)
    txt2.Text = "Rectangle is in Row " + Grid.GetRow(rect1).ToString()
End Sub

Private Sub setRow1(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Grid.SetRow(rect1, 1)
    txt2.Text = "Rectangle is in Row " + Grid.GetRow(rect1).ToString()
End Sub

Private Sub setRow2(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Grid.SetRow(rect1, 2)
    txt2.Text = "Rectangle is in Row " + Grid.GetRow(rect1).ToString()
End Sub

Private Sub setColspan(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Grid.SetColumnSpan(rect1, 3)
    txt3.Text = "ColumnSpan is set to " + Grid.GetColumnSpan(rect1).ToString()
End Sub

Private Sub setRowspan(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Grid.SetRowSpan(rect1, 3)
    txt4.Text = "RowSpan is set to " + Grid.GetRowSpan(rect1).ToString()
End Sub

```

Here is the finished result!

## Grid Methods Sample

Click the buttons on the left to reposition the Rectangle below using methods defined on Grid.

Move Rectangle to Column 0  
Move Rectangle to Column 1  
Move Rectangle to Column 2

Move Rectangle to Row 0  
Move Rectangle to Row 1  
Move Rectangle to Row 2

Span All Columns  
Span All Rows  
Clear All

Column 0, Row 0	Column 1, Row 0	Column 2, Row 0
Column 0, Row 1	Column 1, Row 1	Column 2, Row 1
Column 0, Row 2	Column 1, Row 2	Column 2, Row 2

## See also

- [Grid](#)
- [Panels Overview](#)

# How to: Share Sizing Properties Between Grids

2 minutes to read • [Edit Online](#)

This example shows how to share the sizing data of columns and rows between `Grid` elements in order to keep sizing consistent.

## Example

The following example introduces two `Grid` elements as child elements of a parent `DockPanel`. The `IsSharedSizeScope` attached property of `Grid` is defined on the parent `DockPanel`.

The example manipulates the property value by using two `Button` elements; each element represents one of the Boolean property values. When the `IsSharedSizeScope` property value is set to `true`, each column or row member of a `SharedSizeGroup` shares sizing information, regardless of the content of a row or column.

```
<DockPanel Name="dp1" Grid.IsSharedSizeScope="False" VerticalAlignment="Top" HorizontalAlignment="Left"
Margin="10">
```

...

```

<StackPanel Orientation="Horizontal" DockPanel.Dock="Top">
    <Button Click="setTrue" Margin="0,0,10,10">Set IsSharedSizeScope="True"</Button>
    <Button Click="setFalse" Margin="0,0,10,10">Set IsSharedSizeScope="False"</Button>
</StackPanel>

<StackPanel Orientation="Horizontal" DockPanel.Dock="Top">

    <Grid ShowGridLines="True" Margin="0,0,10,0">
        <Grid.ColumnDefinitions>
            <ColumnDefinition SharedSizeGroup="FirstColumn"/>
            <ColumnDefinition SharedSizeGroup="SecondColumn"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" SharedSizeGroup="FirstRow"/>
        </Grid.RowDefinitions>

        <Rectangle Fill="Silver" Grid.Column="0" Grid.Row="0" Width="200" Height="100"/>
        <Rectangle Fill="Blue" Grid.Column="1" Grid.Row="0" Width="150" Height="100"/>

        <TextBlock Grid.Column="0" Grid.Row="0" FontWeight="Bold">First Column</TextBlock>
        <TextBlock Grid.Column="1" Grid.Row="0" FontWeight="Bold">Second Column</TextBlock>
    </Grid>

    <Grid ShowGridLines="True">
        <Grid.ColumnDefinitions>
            <ColumnDefinition SharedSizeGroup="FirstColumn"/>
            <ColumnDefinition SharedSizeGroup="SecondColumn"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" SharedSizeGroup="FirstRow"/>
        </Grid.RowDefinitions>

        <Rectangle Fill="Silver" Grid.Column="0" Grid.Row="0"/>
        <Rectangle Fill="Blue" Grid.Column="1" Grid.Row="0"/>

        <TextBlock Grid.Column="0" Grid.Row="0" FontWeight="Bold">First Column</TextBlock>
        <TextBlock Grid.Column="1" Grid.Row="0" FontWeight="Bold">Second Column</TextBlock>
    </Grid>
</StackPanel>

<TextBlock Margin="10" DockPanel.Dock="Top" Name="txt1"/>

```

The following code-behind example handles the methods that the button [Click](#) event raises. The example writes the results of these method calls to [TextBlock](#) elements that use related get methods to output the new property values as strings.

```

private void setTrue(object sender, System.Windows.RoutedEventArgs e)
{
    Grid.SetIsSharedSizeScope(dp1, true);
    txt1.Text = "IsSharedSizeScope Property is set to " + Grid.GetIsSharedSizeScope(dp1).ToString();
}

private void setFalse(object sender, System.Windows.RoutedEventArgs e)
{
    Grid.SetIsSharedSizeScope(dp1, false);
    txt1.Text = "IsSharedSizeScope Property is set to " + Grid.GetIsSharedSizeScope(dp1).ToString();
}

```

```
Private Sub setTrue(ByVal sender As Object, ByVal args As RoutedEventArgs)
    Grid.SetIsSharedSizeScope(dp1, True)
    txt1.Text = "IsSharedSizeScope Property is set to " + Grid.GetIsSharedSizeScope(dp1).ToString()
End Sub

Private Sub setFalse(ByVal sender As Object, ByVal args As RoutedEventArgs)
    Grid.SetIsSharedSizeScope(dp1, False)
    txt1.Text = "IsSharedSizeScope Property is set to " + Grid.GetIsSharedSizeScope(dp1).ToString()
End Sub
```

## See also

- [Grid](#)
- [IsSharedSizeScope](#)
- [Panels Overview](#)

# GridSplitter

2 minutes to read • [Edit Online](#)

The [GridSplitter](#) redistributes space between columns or rows of a [Grid](#) control.

## In This Section

[How-to Topics](#)

## Reference

[GridSplitter](#)

## Related Sections

# GridSplitter How-to Topics

2 minutes to read • [Edit Online](#)

The topics in this section describe how to use the [GridSplitter](#) control.

## In This Section

- [Resize Rows with a GridSplitter](#)
- [Resize Columns with a GridSplitter](#)
- [Make Sure That a GridSplitter Is Visible](#)

## Reference

[GridSplitter](#)

[Grid](#)

## Related Sections

# How to: Resize Rows with a GridSplitter

2 minutes to read • [Edit Online](#)

This example shows how to use a horizontal [GridSplitter](#) to redistribute the space between two rows in a [Grid](#) without changing the dimensions of the [Grid](#).

## Example

### How to create a GridSplitter that overlays the edge of a row

To specify a [GridSplitter](#) that resizes adjacent rows in a [Grid](#), set the [Row](#) attached property to one of the rows that you want to resize. If your [Grid](#) has more than one column, set the [ColumnSpan](#) attached property to specify the number of columns. Then set the [VerticalAlignment](#) to [Top](#) or [Bottom](#) (which alignment you set depends on which two rows you want to resize). Finally, set the [HorizontalAlignment](#) property to [Stretch](#).

The following example shows how to define a horizontal [GridSplitter](#) that resizes adjacent rows.

```
<GridSplitter Grid.Row="1"
    Grid.ColumnSpan="3"
    HorizontalAlignment="Stretch"
    VerticalAlignment="Top"
    Background="Black"
    ShowsPreview="true"
    ResizeDirection="Rows"
    Height="5"/>
```

A [GridSplitter](#) that does not occupy its own row may be obscured by other controls in the [Grid](#). For more information about how to prevent this issue, see [Make Sure That a GridSplitter Is Visible](#).

### How to create a GridSplitter that occupies a row

To specify a [GridSplitter](#) that occupies a row in a [Grid](#), set the [Row](#) attached property to one of the rows that you want to resize. If your [Grid](#) has more than one column, set the [ColumnSpan](#) attached property to the number of columns. Then set the [VerticalAlignment](#) to [Center](#), set the [HorizontalAlignment](#) property to [Stretch](#), and set the [Height](#) of the row that contains the [GridSplitter](#) to [Auto](#).

The following example shows how to define a horizontal [GridSplitter](#) that occupies a row and resizes the rows on either side of it.

```
<Grid.RowDefinitions>
    <RowDefinition Height="50*" />
    <RowDefinition Height="Auto" />
    <RowDefinition Height="50*" />
</Grid.RowDefinitions>
```

```
<StackPanel Grid.Row="0" Grid.Column="1" Background="Tan"/>
<GridSplitter Grid.Row="1"
    HorizontalAlignment="Stretch"
    VerticalAlignment="Center"
    Background="Black"
    ShowsPreview="True"
    Height="5"
/>
<StackPanel Grid.Row="2" Grid.Column="0" Background="Brown"/>
```

## See also

- [GridSplitter](#)
- [How-to Topics](#)

# How to: Resize Columns with a GridSplitter

2 minutes to read • [Edit Online](#)

This example shows how to create a vertical [GridSplitter](#) in order to redistribute the space between two columns in a [Grid](#) without changing the dimensions of the [Grid](#).

## Example

### How to create a GridSplitter that overlays the edge of a column

To specify a [GridSplitter](#) that resizes adjacent columns in a [Grid](#), set the [Column](#) attached property to one of the columns that you want to resize. If your [Grid](#) has more than one row, set the [RowSpan](#) attached property to the number of rows. Then set the [HorizontalAlignment](#) property to [Left](#) or [Right](#) (which alignment you set depends on which two columns you want to resize). Finally, set the [VerticalAlignment](#) property to [Stretch](#).

```
<GridSplitter Grid.Column="1"
              Grid.RowSpan="3"
              HorizontalAlignment="Left"
              VerticalAlignment="Stretch"
              Background="Black"
              ShowsPreview="true"
              Width="5"/>
```

A [GridSplitter](#) that does not have its own column may be obscured by other controls in the [Grid](#). For more information about how to prevent this issue, see [Make Sure That a GridSplitter Is Visible](#).

### How to create a GridSplitter that occupies a column

To specify a [GridSplitter](#) that occupies a column in a [Grid](#), set the [Column](#) attached property to one of the columns that you want to resize. If your [Grid](#) has more than one row, set the [RowSpan](#) attached property to the number of rows. Then set the [HorizontalAlignment](#) to [Center](#), set the [VerticalAlignment](#) property to [Stretch](#), and set the [Width](#) of the column that contains the [GridSplitter](#) to [Auto](#).

The following example shows how to define a vertical [GridSplitter](#) that occupies a column and resizes the columns on either side of it.

```
<Grid.ColumnDefinitions>
  <ColumnDefinition/>
  <ColumnDefinition Width="Auto" />
  <ColumnDefinition/>
</Grid.ColumnDefinitions>
```

```
<GridSplitter Grid.Column="1"
              HorizontalAlignment="Center"
              VerticalAlignment="Stretch"
              Background="Black"
              ShowsPreview="True"
              Width="5"
              />
```

## See also

- [GridSplitter](#)
- [How-to Topics](#)

# How to: Make Sure That a GridSplitter Is Visible

2 minutes to read • [Edit Online](#)

This example shows how to make sure a [GridSplitter](#) control is not hidden by the other controls in a [Grid](#).

## Example

The [Children](#) of a [Grid](#) control are rendered in the order that they are defined in markup or code. [GridSplitter](#) controls can be hidden by other controls if you do not define them as the last elements in the [Children](#) collection or if you give other controls a higher [ZIndexProperty](#).

To prevent hidden [GridSplitter](#) controls, do one of the following.

- Make sure that [GridSplitter](#) controls are the last [Children](#) added to the [Grid](#). The following example shows the [GridSplitter](#) as the last element in the [Children](#) collection of the [Grid](#).

```
<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>
  <Button Grid.Column="0"/>
  <GridSplitter Grid.Column ="0" Background="Blue"/>
</Grid>
```

- Set the [ZIndexProperty](#) on the [GridSplitter](#) to be higher than a control that would otherwise hide it. The following example gives the [GridSplitter](#) control a higher [ZIndexProperty](#) than the [Button](#) control.

```
<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>
  <GridSplitter Grid.Column="0" Background="Blue"
                Panel.ZIndex="1"/>
  <Button Grid.Column="0"/>
</Grid>
```

- Set margins on the control that would otherwise hide the [GridSplitter](#) so that the [GridSplitter](#) is exposed. The following example sets margins on a control that would otherwise overlay and hide the [GridSplitter](#).

```
<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>
  <GridSplitter Grid.Column ="0" Background="Blue"/>
  <Button Grid.Column="0" Margin="0,0,5,0"/>
</Grid>
```

## See also

- [GridSplitter](#)

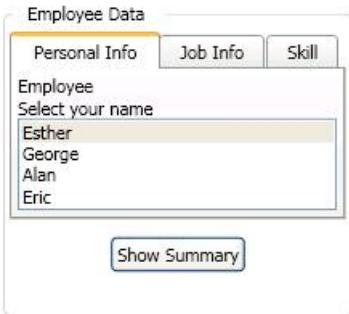
- How-to Topics

# GroupBox

2 minutes to read • [Edit Online](#)

The **GroupBox** control is a [HeaderedContentControl](#) that provides a titled container for graphical user interface (GUI) content.

The following illustration shows a **GroupBox** that contains a **TabControl** and a **Button** that are enclosed in a **StackPanel**.



## In This Section

[Define a GroupBox Template](#)

## Reference

[GroupBox](#)

## Related Sections

# How to: Define a GroupBox Template

2 minutes to read • [Edit Online](#)

This example shows how to create a template for a [GroupBox](#) control.

## Example

The following example defines a [GroupBox](#) control template by using a [Grid](#) control for layout. The template uses a [BorderGapMaskConverter](#) to define the border of the [GroupBox](#) so that the border does not obscure the [Header](#) content.

```
<!-----  
    GroupBox Template Example  
----->  
<BorderGapMaskConverter x:Key="BorderGapMaskConverter"/>  
<Style x:Key="{x:Type GroupBox}">  
    TargetType="{x:Type GroupBox}">  
        <Setter Property="BorderBrush"  
            Value="Gray"/>  
        <Setter Property="Foreground"  
            Value="White"/>  
        <Setter Property="BorderThickness"  
            Value="1"/>  
        <Setter Property="Template">  
            <Setter.Value>  
                <ControlTemplate TargetType="{x:Type GroupBox}">  
                    <Grid>  
                        <Grid.ColumnDefinitions>  
                            <ColumnDefinition Width="4"/>  
                            <ColumnDefinition Width="Auto"/>  
                            <ColumnDefinition Width="*"/>  
                            <ColumnDefinition Width="4"/>  
                        </Grid.ColumnDefinitions>  
                        <Grid.RowDefinitions>  
                            <RowDefinition Height="Auto"/>  
                            <RowDefinition Height="Auto"/>  
                            <RowDefinition Height="*"/>  
                            <RowDefinition Height="4"/>  
                        </Grid.RowDefinitions>  
                        <Border CornerRadius="4"  
                            Grid.Row="1"  
                            Grid.RowSpan="3"  
                            Grid.Column="0"  
                            Grid.ColumnSpan="4"  
                            BorderThickness="{TemplateBinding BorderThickness}"  
                            BorderBrush="Transparent"  
                            Background="{TemplateBinding Background}">  
                            <!-- ContentPresenter for the header -->  
                            <Border x:Name="Header"  
                                Padding="6,0,6,0"  
                                Grid.Row="0"  
                                Grid.RowSpan="2"  
                                Grid.Column="1">  
                                <ContentPresenter ContentSource="Header"  
                                    RecognizesAccessKey="True" />  
                            </Border>  
                            <!-- Primary content for GroupBox -->  
                            <ContentPresenter Grid.Row="2"  
                                Grid.Column="1"  
                                Grid.ColumnSpan="2"  
                                Margin="{TemplateBinding Padding}"/>  
                        </Border>  
                    </Grid>  
                </ControlTemplate>  
            </Setter.Value>  
        </Setter>  
    </Style>
```

```
    ...

```

```
<Border CornerRadius="0"
        Grid.Row="1"
        Grid.RowSpan="3"
        Grid.ColumnSpan="4"
        BorderThickness="{TemplateBinding BorderThickness}"
        BorderBrush="{TemplateBinding BorderBrush}">
    <Border.OpacityMask>
        <MultiBinding Converter=
            "{StaticResource BorderGapMaskConverter}"
            ConverterParameter="6">
            <Binding ElementName="Header"
                    Path="ActualWidth"/>
            <Binding RelativeSource="{RelativeSource Self}"
                    Path="ActualWidth"/>
            <Binding RelativeSource="{RelativeSource Self}"
                    Path="ActualHeight"/>
        </MultiBinding>
    </Border.OpacityMask>
</Border>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
```

## See also

- [GroupBox](#)
- [How to: Create a GroupBox](#)

# Image

2 minutes to read • [Edit Online](#)

The [Image](#) element is used to display bitmap images in Windows Presentation Foundation (WPF) applications.

## In This Section

[How-to Topics](#)

## Reference

[Image](#)

[BitmapImage](#)

[BitmapSource](#)

## See also

- [Imaging Overview](#)
- [How-to Topics](#)

# Image How-to Topics

2 minutes to read • [Edit Online](#)

The topics in this section describe how to use the [Image](#) element.

## In This Section

[Use the Image Element](#)

[Convert an Image to Greyscale](#)

[Crop an Image](#)

[Rotate an Image](#)

## Reference

[Image](#)

[BitmapImage](#)

[BitmapSource](#)

## See also

- [Imaging Overview](#)
- [How-to Topics](#)

# How to: Use the Image Element

3 minutes to read • [Edit Online](#)

This example shows how to include images in an application by using the [Image](#) element.

## Example

The following example shows how to render an image 200 pixels wide. In this Extensible Application Markup Language (XAML) example, both attribute syntax and property tag syntax are used to define the image. For more information on attribute syntax and property syntax, see [Dependency Properties Overview](#). A [BitmapImage](#) is used to define the image's source data and is explicitly defined for the property tag syntax example. In addition, the [DecodePixelWidth](#) of the [BitmapImage](#) is set to the same width as the [Width](#) of the [Image](#). This is done to ensure that the minimum amount of memory is used rendering the image.

### NOTE

In general, if you want to specify the size of a rendered image, specify only the [Width](#) or the [Height](#) but not both. If you only specify one, the image's aspect ratio is preserved. Otherwise, the image may unexpectedly appear stretched or warped. To control the image's stretching behavior, use the [Stretch](#) and [StretchDirection](#) properties.

### NOTE

When you specify the size of an image with either [Width](#) or [Height](#), you should also set either [DecodePixelWidth](#) or [DecodePixelHeight](#) to the same respective size.

The [Stretch](#) property determines how the image source is stretched to fill the image element. For more information, see the [Stretch](#) enumeration.

```
<!-- Simple image rendering. However, rendering an image this way may not
     result in the best use of application memory. See markup below which
     creates the same end result but using less memory. -->
<Image Width="200"
       Source="C:\Documents and Settings\All Users\Documents\My Pictures\Sample Pictures\Water Lilies.jpg"/>

<Image Width="200">
    <Image.Source>
        <!-- To save significant application memory, set the DecodePixelWidth or
            DecodePixelHeight of the BitmapImage value of the image source to the desired
            height and width of the rendered image. If you don't do this, the application will
            cache the image as though it were rendered as its normal size rather than just
            the size that is displayed. -->
        <!-- Note: In order to preserve aspect ratio, only set either DecodePixelWidth
            or DecodePixelHeight but not both. -->
        <BitmapImage DecodePixelWidth="200"
                     UriSource="C:\Documents and Settings\All Users\Documents\My Pictures\Sample Pictures\Water Lilies.jpg" />
    </Image.Source>
</Image>
```

## Example

The following example shows how to render an image 200 pixels wide using code.

**NOTE**

Setting [BitmapImage](#) properties must be done within a [BeginInit](#) and [EndInit](#) block.

```
// Create Image Element
Image myImage = new Image();
myImage.Width = 200;

// Create source
BitmapImage myBitmapImage = new BitmapImage();

// BitmapImage.UriSource must be in a BeginInit/EndInit block
myBitmapImage.BeginInit();
myBitmapImage.UriSource = new Uri(@"C:\Documents and Settings\All Users\Documents\My Pictures\Sample
Pictures\Water Lilies.jpg");

// To save significant application memory, set the DecodePixelWidth or
// DecodePixelHeight of the BitmapImage value of the image source to the desired
// height or width of the rendered image. If you don't do this, the application will
// cache the image as though it were rendered as its normal size rather than just
// the size that is displayed.
// Note: In order to preserve aspect ratio, set DecodePixelWidth
// or DecodePixelHeight but not both.
myBitmapImage.DecodePixelWidth = 200;
myBitmapImage.EndInit();
//set image source
myImage.Source = myBitmapImage;
```

```
' Create Image Element
Dim myImage As New Image()
myImage.Width = 200

' Create source
Dim myBitmapImage As New BitmapImage()

' BitmapImage.UriSource must be in a BeginInit/EndInit block
myBitmapImage.BeginInit()
myBitmapImage.UriSource = New Uri("C:\Documents and Settings\All Users\Documents\My Pictures\Sample
Pictures\Water Lilies.jpg")

' To save significant application memory, set the DecodePixelWidth or
' DecodePixelHeight of the BitmapImage value of the image source to the desired
' height or width of the rendered image. If you don't do this, the application will
' cache the image as though it were rendered as its normal size rather than just
' the size that is displayed.
' Note: In order to preserve aspect ratio, set DecodePixelWidth
' or DecodePixelHeight but not both.
myBitmapImage.DecodePixelWidth = 200
myBitmapImage.EndInit()
'set image source
myImage.Source = myBitmapImage
```

## See also

- [Imaging Overview](#)

# How to: Convert an Image to Greyscale

3 minutes to read • [Edit Online](#)

This example shows how to convert an image to grayscale using [FormatConvertedBitmap](#).

## Example

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <Page.Resources>

        <!-- This resource defines a BitmapImage with a source and a
            DecodePixelWidth of 200. This property is set to the same value
            as the desired width of the image to save on memory use. This
            BitmapImage is used as the base for the other BitmapSource resources. -->
        <BitmapImage x:Key="masterImage" DecodePixelWidth="200"
                     UriSource="C:\Documents and Settings\All Users\Documents\My Pictures\Sample Pictures\Forest.jpg"/>

        <!-- This FormatConvertedBitmap uses the BitmapImage defined above and
            changes the format to Gray32Float (grayscale). -->
        <FormatConvertedBitmap x:Key="convertFormatImage"
                              Source="{StaticResource masterImage}"
                              DestinationFormat="Gray32Float" />

    </Page.Resources>

    <StackPanel>

        <!-- Apply the "convertFormatImage" resource defined above to this image. -->
        <Image Width="200" Source="{StaticResource convertFormatImage}" />
    </StackPanel>
</Page>
```

```

using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
using System.Windows.Media.Imaging;

namespace SDKSample
{
    public partial class FormatConvertedBitmapExample : Page
    {
        public FormatConvertedBitmapExample()
        {

            ///// Create a BitmapImage and set it's DecodePixelWidth to 200. Use /////
            ///// this BitmapImage as a source for other BitmapSource objects. /////

            BitmapImage myBitmapImage = new BitmapImage();

            // BitmapSource objects like BitmapImage can only have their properties
            // changed within a BeginInit/EndInit block.
            myBitmapImage.BeginInit();
            myBitmapImage.UriSource = new Uri(@"sampleImages/WaterLilies.jpg", UriKind.Relative);

            // To save significant application memory, set the DecodePixelWidth or
            // DecodePixelHeight of the BitmapImage value of the image source to the desired
            // height or width of the rendered image. If you don't do this, the application will
            // cache the image as though it were rendered as its normal size rather than just
            // the size that is displayed.
            // Note: In order to preserve aspect ratio, set DecodePixelWidth
            // or DecodePixelHeight but not both.
            myBitmapImage.DecodePixelWidth = 200;
            myBitmapImage.EndInit();

            ////////////// Convert the BitmapSource to a new format /////////////
            // Use the BitmapImage created above as the source for a new BitmapSource object
            // which is set to a gray scale format using the FormatConvertedBitmap BitmapSource.
            // Note: New BitmapSource does not cache. It is always pulled when required.

            FormatConvertedBitmap newFormatedBitmapSource = new FormatConvertedBitmap();

            // BitmapSource objects like FormatConvertedBitmap can only have their properties
            // changed within a BeginInit/EndInit block.
            newFormatedBitmapSource.BeginInit();

            // Use the BitmapSource object defined above as the source for this new
            // BitmapSource (chain the BitmapSource objects together).
            newFormatedBitmapSource.Source = myBitmapImage;

            // Set the new format to Gray32Float (grayscale).
            newFormatedBitmapSource.DestinationFormat = PixelFormats.Gray32Float;
            newFormatedBitmapSource.EndInit();

            // Create Image Element
            Image myImage = new Image();
            myImage.Width = 200;
            //set image source
            myImage.Source = newFormatedBitmapSource;

            // Add Image to the UI
            StackPanel myStackPanel = new StackPanel();
            myStackPanel.Children.Add(myImage);
            this.Content = myStackPanel;
        }
    }
}

```

```

Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Media
Imports System.Windows.Media.Imaging

Namespace SDKSample

    Class FormatConvertedBitmapExample
        Inherits Page

        Public Sub New()

            '/// Create a BitmapImage and set it's DecodePixelWidth to 200. Use /////
            '/// this BitmapImage as a source for other BitmapSource objects.     /////
            Dim myBitmapImage As New BitmapImage()

            ' BitmapSource objects like BitmapImage can only have their properties
            ' changed within a BeginInit/EndInit block.
            myBitmapImage.BeginInit()
            myBitmapImage.UriSource = New Uri("sampleImages/WaterLilies.jpg", UriKind.Relative)

            ' To save significant application memory, set the DecodePixelWidth or
            ' DecodePixelHeight of the BitmapImage value of the image source to the desired
            ' height or width of the rendered image. If you don't do this, the application will
            ' cache the image as though it were rendered as its normal size rather than just
            ' the size that is displayed.
            ' Note: In order to preserve aspect ratio, set DecodePixelWidth
            ' or DecodePixelHeight but not both.
            myBitmapImage.DecodePixelWidth = 200
            myBitmapImage.EndInit()

            '//////// Convert the BitmapSource to a new format ///////////
            ' Use the BitmapImage created above as the source for a new BitmapSource object
            ' which is set to a gray scale format using the FormatConvertedBitmap BitmapSource.
            ' Note: New BitmapSource does not cache. It is always pulled when required.
            Dim newFormatedBitmapSource As New FormatConvertedBitmap()

            ' BitmapSource objects like FormatConvertedBitmap can only have their properties
            ' changed within a BeginInit/EndInit block.
            newFormatedBitmapSource.BeginInit()

            ' Use the BitmapSource object defined above as the source for this new
            ' BitmapSource (chain the BitmapSource objects together).
            newFormatedBitmapSource.Source = myBitmapImage

            ' Set the new format to Gray32Float (grayscale).
            newFormatedBitmapSource.DestinationFormat = PixelFormats.Gray32Float
            newFormatedBitmapSource.EndInit()

            ' Create Image Element
            Dim myImage As New Image()
            myImage.Width = 200
            'set image source
            myImage.Source = newFormatedBitmapSource

            ' Add Image to the UI
            Dim myStackPanel As New StackPanel()
            myStackPanel.Children.Add(myImage)
            Me.Content = myStackPanel

        End Sub
    End Class
End Namespace 'ImagingSnippetGallery

```

## See also

- [Use the Image Element](#)
- [Crop an Image](#)
- [Rotate an Image](#)

# How to: Crop an Image

2 minutes to read • [Edit Online](#)

This example shows how to crop an image using [CroppedBitmap](#).

[CroppedBitmap](#) is primarily used when encoding a cropped version of an image to save out to a file. To crop an image for display purposes see the [How to: Create a Clip Region](#) topic.

## Example

The following Extensible Application Markup Language (XAML) defines resources used within the samples below.

```
<Page.Resources>
    <!-- Define some image resources, for use as the image element source. -->
    <BitmapImage x:Key="masterImage" UriSource="/sampleImages/gecko.jpg" />
    <CroppedBitmap x:Key="croppedImage"
        Source="{StaticResource masterImage}" SourceRect="30 20 105 50"/>
</Page.Resources>
```

The following example creates an image using a [CroppedBitmap](#) as its source.

```
<!-- Use the cropped image resource as the images source -->
<Image Width="200" Source="{StaticResource croppedImage}"
    Margin="5" Grid.Column="0" Grid.Row="1" />
```

```
// Create an Image element.
Image croppedImage = new Image();
croppedImage.Width = 200;
croppedImage.Margin = new Thickness(5);

// Create a CroppedBitmap based off of a xaml defined resource.
CroppedBitmap cb = new CroppedBitmap(
    (BitmapSource)this.Resources["masterImage"],
    new Int32Rect(30, 20, 105, 50));           //select region rect
croppedImage.Source = cb;                      //set image source to cropped
```

```
' Create an Image element.
Dim croppedImage As New Image()
croppedImage.Width = 200
croppedImage.Margin = New Thickness(5)

' Create a CroppedBitmap based off of a xaml defined resource.
Dim cb As New CroppedBitmap(CType(Me.Resources("masterImage"), BitmapSource), New Int32Rect(30, 20, 105, 50))
'select region rect
croppedImage.Source = cb 'set image source to cropped
```

The [CroppedBitmap](#) can also be used as the source of another [CroppedBitmap](#), chaining the cropping. Note that the [SourceRect](#) uses values that are relative to the source cropped bitmap and not the initial image.

```
<!-- Chain a cropped bitmap off a previously defined cropped image -->
<Image Width="200" Grid.Column="0" Grid.Row="3" Margin="5">
    <Image.Source>
        <CroppedBitmap Source="{StaticResource croppedImage}"
            SourceRect="30 0 75 50"/>
    </Image.Source>
</Image>
```

```
// Create an Image element.
Image chainImage = new Image();
chainImage.Width = 200;
chainImage.Margin = new Thickness(5);

// Create the cropped image based on previous CroppedBitmap.
CroppedBitmap chained = new CroppedBitmap(cb,
    new Int32Rect(30, 0, (int)cb.Width-30, (int)cb.Height));
// Set the image's source.
chainImage.Source = chained;
```

```
' Create an Image element.
Dim chainImage As New Image()
chainImage.Width = 200
chainImage.Margin = New Thickness(5)

' Create the cropped image based on previous CroppedBitmap.
Dim chained As New CroppedBitmap(cb, New Int32Rect(30, 0, CType(cb.Width, Integer) - 30, CType(cb.Height,
Integer)))
' Set the image's source.
chainImage.Source = chained
```

## See also

- [How to: Create a Clip Region](#)

# How to: Rotate an Image

2 minutes to read • [Edit Online](#)

This example shows how to rotate an image 90 degrees by using a [Rotation](#) property of a [BitmapImage](#).

## Example

```
<Image Width="150" Margin="5" Grid.Column="0" Grid.Row="1">
    <Image.Source>
        <BitmapImage UriSource="/sampleImages/watermelon.jpg" Rotation="Rotate90" />
    </Image.Source>
</Image>
```

```
//Create Image element
Image rotated270 = new Image();
rotated270.Width = 150;

//Create source
BitmapImage bi = new BitmapImage();
//BitmapImage properties must be in a BeginInit/EndInit block
bi.BeginInit();
bi.UriSource = new Uri(@"pack://application:,,/sampleImages/watermelon.jpg");
//Set image rotation
bi.Rotation = Rotation.Rotate270;
bi.EndInit();
//set image source
rotated270.Source = bi;
```

```
'Create Image element
Dim rotated270 As New Image()
rotated270.Width = 150

'Create source
Dim bi As New BitmapImage()
'BitmapImage properties must be in a BeginInit/EndInit block
bi.BeginInit()
bi.UriSource = New Uri("pack://application:,,/sampleImages/watermelon.jpg")
'Set image rotation
bi.Rotation = Rotation.Rotate270
bi.EndInit()
'set image source
rotated270.Source = bi
```

# Label

2 minutes to read • [Edit Online](#)

**Label** controls usually provide information in the user interface (UI). Historically, a **Label** has contained only text, but because the **Label** that ships with Windows Presentation Foundation (WPF) is a **ContentControl**, it can contain either text or a **UIElement**.

A **Label** provides both functional and visual support for access keys. It is frequently used to enable quick keyboard access to controls such as a **TextBox**. To assign a **Label** to a **Control**, set the **Label.Target** property to the control that should get focus when the user presses the access key.

The following image shows a **Label** "Themes" that targets a **ComboBox**. When the user presses , the **ComboBox** receives focus. For more information, see [How to: Set the Target Property of a Label](#).



## In This Section

[How to: Create a Control That Has an Access Key and Text Wrapping](#)

## Reference

[Label](#)

# How to: Create a Control That Has an Access Key and Text Wrapping

2 minutes to read • [Edit Online](#)

This example shows how to create a control that has an access key and supports text wrapping. The example uses a [Label](#) control to illustrate these concepts.

## Example

### Add Text Wrapping to Your Label

The [Label](#) control does not support text wrapping. If you need a label that wraps across multiple lines, you can nest another element that does support text wrapping and put the element inside the label. The following example shows how to use a [TextBlock](#) to make a label that wraps several lines of text.

```
<Label Width="200" HorizontalAlignment="Left">
    <TextBlock TextWrapping="WrapWithOverflow">
        A long piece of text that requires text wrapping
        goes here.
    </TextBlock>
</Label>
```

### Add an Access Key and Text Wrapping to Your Label

If you need a [Label](#) that has an access key (mnemonic), use the [AccessText](#) element that is inside the [Label](#).

Controls such as [Label](#), [Button](#), [RadioButton](#), [CheckBox](#), [MenuItem](#), [TabItem](#), [Expander](#), and [GroupBox](#) have default control templates. These templates contain a [ContentPresenter](#). One of the properties that you can set on the [ContentPresenter](#) is [RecognizesAccessKey="true"](#), which you can use to specify an access key for the control.

The following example shows how to create a [Label](#) that has an access key and supports text wrapping. To enable text wrapping, the example sets the [TextWrapping](#) property and uses an underline character to specify the access key. (The character that immediately follows the underline character is the access key.)

```
<TextBox Name="textBox1" Width="50" Height="20"/>
<Label Width="200" HorizontalAlignment="Left"
      Target="{Binding ElementName(textBox1)}">
    <AccessText TextWrapping="WrapWithOverflow">
        _Another long piece of text that requires text wrapping
        goes here.
    </AccessText>
</Label>
```

## See also

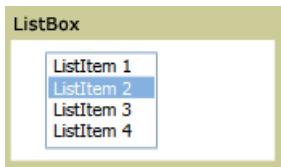
- [How to: Set the Target Property of a Label](#)

# ListBox

2 minutes to read • [Edit Online](#)

A [ListBox](#) control provides users with a list of selectable items.

The following figure illustrates a typical [ListBox](#).



Typical ListBox

## In This Section

[How-to Topics](#)

## Reference

[ListBox](#)

[ListBoxItem](#)

## Related Sections

# ListBox How-to Topics

2 minutes to read • [Edit Online](#)

The topics in this section describe how to use the [ListBox](#) control to display selectable lists of items.

## In This Section

[Bind a ListBox to Data](#)

[Get a ListBoxItem](#)

[How to: Add Data to an ItemsControl](#)

[Improve the Scrolling Performance of a ListBox](#)

## Reference

[ListBox](#)

[ListBoxItem](#)

## Related Sections

# How to: Bind a ListBox to Data

2 minutes to read • [Edit Online](#)

An application developer can create [ListBox](#) controls without specifying the contents of each [ListBoxItem](#) separately. You can use data binding to bind data to the individual items.

The following example shows how to create a [ListBox](#) that populates the [ListBoxItem](#) elements by data binding to a data source called *Colors*. In this case it is not necessary to use [ListBoxItem](#) tags to specify the content of each item.

## Example

```
<Canvas.Resources>
    <src:myColors x:Key="Colors"/>
</Canvas.Resources>
```

```
<ListBox Name="myListBox" HorizontalAlignment="Left" SelectionMode="Extended"
        Width="265" Height="55" Background="HoneyDew" SelectionChanged="myListBox_SelectionChanged"
        ItemsSource="{Binding Source={StaticResource Colors}}" IsSynchronizedWithCurrentItem="true">
</ListBox>
```

## See also

- [ListBox](#)
- [ListBoxItem](#)
- [Controls](#)

# How to: Get a ListBoxItem

2 minutes to read • [Edit Online](#)

If you need to get a specific [ListBoxItem](#) at a particular index in a [ListBox](#), you can use an [ItemContainerGenerator](#).

## Example

The following example shows a [ListBox](#) and its items.

```
<ListBox Margin="10,0,0,5" Name="lb" VerticalAlignment="Top" Grid.Column="0" Grid.Row="2">
    <ListBoxItem>Item 0</ListBoxItem>
    <ListBoxItem>Item 1</ListBoxItem>
    <ListBoxItem>Item 2</ListBoxItem>
    <ListBoxItem>Item 3</ListBoxItem>
</ListBox>
```

The following example shows how to retrieve the item by specifying the index of the item in the [ContainerFromIndex](#) property of the [ItemContainerGenerator](#).

```
private void GetIndex0(object sender, RoutedEventArgs e)
{
    ListBoxItem lbi = (ListBoxItem)
        (lb.ItemContainerGenerator.ContainerFromIndex(0));
    Item.Content = "The contents of the item at index 0 are: " +
        (lbi.Content.ToString()) + ".";
}
```

```
Private Sub GetIndex0(ByVal Sender As Object, ByVal e As RoutedEventArgs)

    Dim lbi As ListBoxItem = CType(
        lb.ItemContainerGenerator.ContainerFromIndex(0), ListBoxItem)
    Item.Content = "The contents of the item at index 0 are: " +
        (lbi.Content.ToString()) + "."
End Sub
```

After you have retrieved the list box item, you can display the contents of the item, as shown in the following example.

```
Item.Content = "The contents of the item at index 0 are: " +
    (lbi.Content.ToString()) + ".";
```

```
Item.Content = "The contents of the item at index 0 are: " +
    (lbi.Content.ToString()) + ".";
```

# How to: Improve the Scrolling Performance of a ListBox

2 minutes to read • [Edit Online](#)

If a [ListBox](#) contains many items, the user interface response can be slow when a user scrolls the [ListBox](#) by using the mouse wheel or dragging the thumb of a scrollbar. You can improve the performance of the [ListBox](#) when the user scrolls by setting the `VirtualizingStackPanel.VirtualizationMode` attached property to [VirtualizationMode.Recycling](#).

## Example

### Description

The following example creates a [ListBox](#) and sets the `VirtualizingStackPanel.VirtualizationMode` attached property to [VirtualizationMode.Recycling](#) to improve performance during scrolling.

### Code

```
<StackPanel>

    <StackPanel.Resources>
        <src:LotsOfItems x:Key="data"/>
    </StackPanel.Resources>

    <ListBox Height="150" ItemsSource="{StaticResource data}"
             VirtualizingStackPanel.VirtualizationMode="Recycling" />

</StackPanel>
```

The following example shows the data that the previous example uses.

```
public class LotsOfItems : ObservableCollection<String>
{
    public LotsOfItems()
    {
        for (int i = 0; i < 1000; ++i)
        {
            Add("item " + i.ToString());
        }
    }
}
```

```
Public Class LotsOfItems
    Inherits ObservableCollection(Of String)
    Public Sub New()
        For i As Integer = 0 To 999
            Add("item " & i.ToString())
        Next
    End Sub
End Class
```

# ListView

2 minutes to read • [Edit Online](#)

The [ListView](#) control provides the infrastructure to display a set of data items in different layouts or views.

The following illustration shows a [ListView](#).

First Name	Last Name	Employee No.
Jesper	Aaberg	12345
Dominik	Paiha	98765
Yale	Li	23875
Muru	Subramani	49392

## In This Section

[Overviews](#)

[How-to Topics](#)

## Reference

[ListView](#)

[ListViewItem](#)

[GridView](#)

## Related Sections

[Data Binding Overview](#)

[Data Templating Overview](#)

# ListView Overviews

2 minutes to read • [Edit Online](#)

The topics in this section show you how to use a *ListView* control.

## In This Section

[ListView Overview](#)

[GridView Overview](#)

[GridView Column Header Styles and Templates Overview](#)

## Reference

[ListView](#)

[GridView](#)

## Related Sections

[How-to Topics](#)

# ListView Overview

3 minutes to read • [Edit Online](#)

The [ListView](#) control provides the infrastructure to display a set of data items in different layouts or views. For example, a user may want to display data items in a table and also to sort its columns.

## What Is a ListView?

The [ListView](#) control is an [ItemsControl](#) that is derived from [ListBox](#). Typically, its items are members of a data collection and are represented as [ListViewItem](#) objects. A [ListViewItem](#) is a [ContentControl](#) and can contain only a single child element. However, that child element can be any visual element.

## Defining a View Mode for a ListView

To specify a view mode for the content of a [ListView](#) control, you set the [View](#) property. One view mode that Windows Presentation Foundation (WPF) provides is [GridView](#), which displays a collection of data items in a table that has customizable columns.

The following example shows how to define a [GridView](#) for a [ListView](#) control that displays employee information.

```
<ListView ItemsSource="{Binding Source=
    {StaticResource EmployeeInfoDataSource}}">

    <ListView.View>

        <GridView AllowsColumnReorder="true"
            ColumnHeaderToolTip="Employee Information">

            <GridViewColumn DisplayMemberBinding=
                "{Binding Path=FirstName}"
                Header="First Name" Width="100"/>

            <GridViewColumn DisplayMemberBinding=
                "{Binding Path=LastName}"
                Width="100">
                <GridViewColumnHeader>Last Name
                    <GridViewColumnHeader.ContextMenu>
                        <ContextMenu MenuItem.Click="LastNameCM_Click"
                            Name="LastNameCM">
                            <MenuItem Header="Ascending" />
                            <MenuItem Header="Descending" />
                        </ContextMenu>
                    </GridViewColumnHeader.ContextMenu>
                </GridViewColumnHeader>
            </GridViewColumn>

            <GridViewColumn DisplayMemberBinding=
                "{Binding Path=EmployeeNumber}"
                Header="Employee No." Width="100"/>
        </GridView>
    </ListView.View>
</ListView>
```

The following illustration shows how the data appears for the previous example.

First Name	Last Name	Employee No.
Jesper	Aaberg	12345
Dominik	Paiha	98765
Yale	Li	23875
Muru	Subramani	49392

You can create a custom view mode by defining a class that inherits from the [ViewBase](#) class. The [ViewBase](#) class provides the infrastructure that you need to create a custom view. For more information about how to create a custom view, see [Create a Custom View Mode for a ListView](#).

## Binding Data to a ListView

Use the [Items](#) and [ItemsSource](#) properties to specify items for a [ListView](#) control. The following example sets the [ItemsSource](#) property to a data collection that is called [EmployeeInfoDataSource](#).

```
<ListView ItemsSource="{Binding Source=
    {StaticResource EmployeeInfoDataSource}}">
```

In a [GridView](#), [GridViewColumn](#) objects bind to specified data fields. The following example binds a [GridViewColumn](#) object to a data field by specifying a [Binding](#) for the [DisplayMemberBinding](#) property.

```
GridViewColumn gvc1 = new GridViewColumn();
gvc1.DisplayMemberBinding = new Binding("FirstName");
gvc1.Header = "FirstName";
gvc1.Width = 100;
```

```
Dim gvc1 As New GridViewColumn()
gvc1.DisplayMemberBinding = New Binding("FirstName")
gvc1.Header = "FirstName"
gvc1.Width = 100
```

```
<GridViewColumn DisplayMemberBinding=
    "{Binding Path=FirstName}"
    Header="First Name" Width="100"/>
```

You can also specify a [Binding](#) as part of a [DataTemplate](#) definition that you use to style the cells in a column. In the following example, the [DataTemplate](#) that is identified with a [ResourceKey](#) sets the [Binding](#) for a [GridViewColumn](#). Note that this example does not define the [DisplayMemberBinding](#) because doing so overrides the binding that is specified by [DataTemplate](#).

```
<DataTemplate x:Key="myCellTemplateMonth">
    <DockPanel>
        <TextBlock Foreground="DarkBlue" HorizontalAlignment="Center">
            <TextBlock.Text>
                <Binding Path="Month"/>
            </TextBlock.Text>
        </TextBlock>
    </DockPanel>
</DataTemplate>
```

```
<GridViewColumn Header="Month" Width="80"
    CellTemplate="{StaticResource myCellTemplateMonth}"/>
```

## Styling a ListView That Implements a GridView

The [ListView](#) control contains [ListViewItem](#) objects, which represent the data items that are displayed. You can use the following properties to define the content and style of data items:

- On the [ListView](#) control, use the [ItemTemplate](#), [ItemTemplateSelector](#), and [ItemContainerStyle](#) properties.
- On the [ListViewItem](#) control, use the [ContentTemplate](#) and [ContentTemplateSelector](#) properties.

To avoid alignment issues between cells in a [GridView](#), do not use the [ItemContainerStyle](#) to set properties or add content that affects the width of an item in a [ListView](#). For example, an alignment issue can occur when you set the [Margin](#) property in the [ItemContainerStyle](#). To specify properties or define content that affects the width of items in a [GridView](#), use the properties of the [GridView](#) class and its related classes, such as [GridViewColumn](#).

For more information about how to use [GridView](#) and its supporting classes, see [GridView Overview](#).

If you define an [ItemContainerStyle](#) for a [ListView](#) control and also define an [ItemTemplate](#), you must include a [ContentPresenter](#) in the style in order for the [ItemTemplate](#) to work correctly.

Do not use the [HorizontalContentAlignment](#) and [VerticalContentAlignment](#) properties for [ListView](#) content that is displayed by using a [GridView](#). To specify the alignment of content in a column of a [GridView](#), define a [CellTemplate](#).

## Sharing the Same View Mode

Two [ListView](#) controls cannot share the same view mode at the same time. If you try to use the same view mode with more than one [ListView](#) control, an exception occurs.

To specify a view mode that can be simultaneously used by more than one [ListView](#), use templates or styles.

## Creating a Custom View Mode

Customized views like [GridView](#) are derived from the [ViewBase](#) abstract class, which provides the tools to display data items that are represented as [ListViewItem](#) objects.

## See also

- [GridView](#)
- [ListView](#)
- [ListViewItem](#)
- [Binding](#)
- [GridView Overview](#)
- [How-to Topics](#)
- [Controls](#)

# GridView Overview

5 minutes to read • [Edit Online](#)

**GridView** view mode is one of the view modes for a [ListView](#) control. The **GridView** class and its supporting classes enable you and your users to view item collections in a table that typically uses buttons as interactive column headers. This topic introduces the **GridView** class and outlines its use.

## What Is a GridView View?

The **GridView** view mode displays a list of data items by binding data fields to columns and by displaying a column header to identify the field. The default **GridView** style implements buttons as column headers. By using buttons for column headers, you can implement important user interaction capabilities; for example, users can click the column header to sort **GridView** data according to the contents of a specific column.

### NOTE

The button controls that **GridView** uses for column headers are derived from [ButtonBase](#).

The following illustration shows a **GridView** view of [ListView](#) content.

Name	Time	Artist	Disk
Song1	3:54	Singer1	Disk1
Song2	4:31	Singer2	Disk3
<b>Recommended</b>			
Song3	5:06	Singer3	Disk1
<b>Strongly Recommended</b>			
Song4	4:18	Singer3	Disk2
Song5	6:15	Singer1	Disk3
<b>Strongly Recommended</b>			

**GridView** columns are represented by [GridViewColumn](#) objects, which can automatically size to their content. Optionally, you can explicitly set a [GridViewColumn](#) to a specific width. You can resize columns by dragging the gripper between column headers. You can also dynamically add, remove, replace, and reorder columns because this functionality is built into **GridView**. However, **GridView** cannot directly update the data that it displays.

The following example shows how to define a **GridView** that displays employee data. In this example, [ListView](#) defines the `EmployeeInfoDataSource` as the [ItemsSource](#). The property definitions of [DisplayMemberBinding](#) bind [GridViewColumn](#) content to `EmployeeInfoDataSource` data categories.

```

<ListView ItemsSource="{Binding Source=
    {StaticResource EmployeeInfoDataSource}}">

    <ListView.View>

        <GridView AllowsColumnReorder="true"
            ColumnHeaderToolTip="Employee Information">

            <GridViewColumn DisplayMemberBinding=
                "{Binding Path=FirstName}"
                Header="First Name" Width="100"/>

            <GridViewColumn DisplayMemberBinding=
                "{Binding Path=LastName}"
                Width="100">
                <GridViewColumnHeader>Last Name
                    <GridViewColumnHeader.ContextMenu>
                        <ContextMenu MenuItem.Click="LastNameCM_Click"
                            Name="LastNameCM">
                            <MenuItem Header="Ascending" />
                            <MenuItem Header="Descending" />
                        </ContextMenu>
                    </GridViewColumnHeader.ContextMenu>
                </GridViewColumnHeader>
            </GridViewColumn>

            <GridViewColumn DisplayMemberBinding=
                "{Binding Path=EmployeeNumber}"
                Header="Employee No." Width="100"/>
        </GridView>

    </ListView.View>
</ListView>

```

The following illustration shows the table that the previous example creates. The **GridView** control displays data from an **ItemsSource** object:

First Name	Last Name	Employee No.
Jesper	Aaberg	12345
Dominik	Paiha	98765
Yale	Li	23875
Muru	Subramani	49392

## GridView Layout and Style

The column cells and the column header of a **GridViewColumn** have the same width. By default, each column sizes its width to fit its content. Optionally, you can set a column to a fixed width.

Related data content displays in horizontal rows. For example, in the previous illustration, each employee's last name, first name, and ID number are displayed as a set because they appear in a horizontal row.

### Defining and Styling Columns in a GridView

When defining the data field to display in a **GridViewColumn**, use the **DisplayMemberBinding**, **CellTemplate**, or **CellTemplateSelector** properties. The **DisplayMemberBinding** property takes precedence over either of the template properties.

To specify the alignment of content in a column of a **GridView**, define a **CellTemplate**. Do not use the **HorizontalContentAlignment** and **VerticalContentAlignment** properties for **ListView** content that is displayed by using a **GridView**.

To specify template and style properties for column headers, use the **GridView**, **GridViewColumn**, and

[GridViewColumnHeader](#) classes. For more information, see [GridView Column Header Styles and Templates Overview](#).

### Adding Visual Elements to a GridView

To add visual elements, such as [CheckBox](#) and [Button](#) controls, to a [GridView](#) view mode, use templates or styles.

If you explicitly define a visual element as a data item, it can appear only one time in a [GridView](#). This limitation exists because an element can have only one parent and therefore, can appear only one time in the visual tree.

### Styling Rows in a GridView

Use the [GridViewRowPresenter](#) and [GridViewHeaderRowPresenter](#) classes to format and display the rows of a [GridView](#). For an example of how to style rows in a [GridView](#) view mode, see [Style a Row in a ListView That Implements a GridView](#).

### Alignment Issues When You Use ItemContainerStyle

To prevent alignment issues between column headers and cells, do not set a property or specify a template that affects the width of an item in an [ItemContainerStyle](#). For example, do not set the [Margin](#) property or specify a [ControlTemplate](#) that adds a [CheckBox](#) to an [ItemContainerStyle](#) that is defined on a [ListView](#) control. Instead, specify the properties and templates that affect column width directly on classes that define a [GridView](#) view mode.

For example, to add a [CheckBox](#) to the rows in [GridView](#) view mode, add the [CheckBox](#) to a [DataTemplate](#), and then set the [CellTemplate](#) property to that [DataTemplate](#).

## User Interactions with a GridView

When you use a [GridView](#) in your application, users can interact with and modify the formatting of the [GridView](#). For example, users can reorder columns, resize a column, select items in a table, and scroll through content. You can also define an event handler that responds when a user clicks the column header button. The event handler can perform operations like sorting the data that is displayed in the [GridView](#) according to the contents of a column.

The following list discusses in more detail the capabilities of using [GridView](#) for user interaction:

- **Reorder columns by using the drag-and-drop method.**

Users can reorder columns in a [GridView](#) by pressing the left mouse button while it is over a column header and then dragging that column to a new position. While the user drags the column header, a floating version of the header is displayed as well as a solid black line that shows where to insert the column.

If you want to modify the default style for the floating version of a header, specify a [ControlTemplate](#) for a [GridViewColumnHeader](#) type that is triggered when the [Role](#) property is set to [Floating](#). For more information, see [Create a Style for a Dragged GridView Column Header](#).

- **Resize a column to its content.**

Users can double-click the gripper to the right of a column header in order to resize a column to fit its content.

**NOTE**

You can set the [Width](#) property to `Double.NaN` to produce the same effect.

- **Select row items.**

Users can select one or more items in a [GridView](#).

If you want to change the [Style](#) of a selected item, see [Use Triggers to Style Selected Items in a ListView](#).

- **Scroll to view content that is not initially visible on the screen.**

If the size of the [GridView](#) is not large enough to display all the items, users can scroll horizontally or vertically by using scrollbars, which are provided by a [ScrollViewer](#) control. A [ScrollBar](#) is hidden if all the content is visible in a specific direction. Column headers do not scroll with a vertical scroll bar, but do scroll horizontally.

- **Interact with columns by clicking the column header buttons.**

When users click a column header button, they can sort the data that is displayed in the column if you have provided a sorting algorithm.

You can handle the [Click](#) event for column header buttons in order to provide functionality like a sorting algorithm. To handle the [Click](#) event for a single column header, set an event handler on the [GridViewColumnHeader](#). To set an event handler that handles the [Click](#) event for all column headers, set the handler on the [ListView](#) control.

## Obtaining Other Custom Views

The [GridView](#) class, which is derived from the [ViewBase](#) abstract class, is just one of the possible view modes for the [ListView](#) class. You can create other custom views for [ListView](#) by deriving from the [ViewBase](#) class. For an example of a custom view mode, see [Create a Custom View Mode for a ListView](#).

## GridView Supporting Classes

The following classes support the [GridView](#) view mode.

- [GridViewColumn](#)
- [GridViewColumnHeader](#)
- [GridViewRowPresenter](#)
- [GridViewHeaderRowPresenter](#)
- [GridViewColumnCollection](#)
- [GridViewColumnHeaderRole](#)

## See also

- [ListView](#)
- [ListViewItem](#)
- [GridViewColumn](#)
- [GridViewColumnHeader](#)
- [GridViewRowPresenter](#)
- [GridViewHeaderRowPresenter](#)
- [ViewBase](#)
- [ListView Overview](#)
- [Sort a GridView Column When a Header Is Clicked](#)
- [How-to Topics](#)

# GridView Column Header Styles and Templates Overview

2 minutes to read • [Edit Online](#)

This overview discusses the order of precedence for properties that you use to customize a column header in the [GridView](#) view mode of a [ListView](#) control.

## Customizing a Column Header in a GridView

The properties that define the content, layout, and style of a column header in a [GridView](#) are found on many related classes. Some of these properties have functionality that is similar or the same.

The rows in the following table show groups of properties that perform the same function. You can use these properties to customize the column headers in a [GridView](#). The order of precedence for related properties is from right to left where the property in the farthest right column has the highest precedence. For example, if a [ContentTemplate](#) is set on the [GridViewColumnHeader](#) object and the [HeaderTemplateSelector](#) is set on the associated [GridViewColumn](#), the [ContentTemplate](#) takes precedence. In this scenario, the [HeaderTemplateSelector](#) has no effect.

### Related properties for column headers in a GridView

Classes	GridView	GridViewColumn	GridViewColumnHeader
<b>Context Menu Properties</b>	<a href="#">ColumnHeaderContextMenu</a>	Not applicable	<a href="#">ContextMenu</a>
<b>ToolTip</b>	<a href="#">ColumnHeaderToolTip</a>	Not applicable	<a href="#">ToolTip</a>
<b>Properties</b>			
<b>Header Template</b>	<a href="#">ColumnHeaderTemplate</a> <sup>1/</sup>	<a href="#">HeaderTemplate</a> <sup>1/</sup>	<a href="#">ContentTemplate</a> <sup>1/</sup>
<b>Properties</b>	<a href="#">ColumnHeaderTemplateSelector</a>	<a href="#">HeaderTemplateSelector</a>	<a href="#">ContentTemplateSelector</a>
<b>Style Properties</b>	<a href="#">ColumnHeaderContainerStyle</a>	<a href="#">HeaderContainerStyle</a>	<a href="#">Style</a>

<sup>1</sup>For **Header Template Properties**, if you set both the template and template selector properties, the template property takes precedence. For example, if you set both the [ContentTemplate](#) and [ContentTemplateSelector](#) properties, the [ContentTemplate](#) property takes precedence.

## See also

- [How-to Topics](#)
- [ListView Overview](#)
- [GridView Overview](#)

# ListView How-to Topics

2 minutes to read • [Edit Online](#)

The topics in this section describe how to use the [ListView](#) control to display a set of data items.

## In This Section

- [Sort a GridView Column When a Header Is Clicked](#)
- [Create a Custom View Mode for a ListView](#)
- [Use Templates to Style a ListView That Uses GridView](#)
- [Create a Style for a Dragged GridView Column Header](#)
- [Display ListView Contents by Using a GridView](#)
- [Use Triggers to Style Selected Items in a ListView](#)
- [Create ListViewItems with a CheckBox](#)
- [Display Data by Using GridViewRowPresenter](#)
- [Group Items in a ListView That Implements a GridView](#)
- [Style a Row in a ListView That Implements a GridView](#)
- [Change the Horizontal Alignment of a Column in a ListView](#)
- [Handle the MouseDoubleClick Event for Each Item in a ListView](#)

## Reference

[ListView](#)

[ListViewItem](#)

[GridView](#)

## Related Sections

[ListView Overview](#)

# How to: Sort a GridView Column When a Header Is Clicked

2 minutes to read • [Edit Online](#)

This example shows how to create a [ListView](#) control that implements a [GridView](#) view mode and sorts the data content when a user clicks a column header.

## Example

The following example defines a [GridView](#) with three columns that bind to the [Year](#), [Month](#), and [Day](#), properties of the [DateTime](#) structure.

```
<GridView>
    <GridViewColumn DisplayMemberBinding="{Binding Path=Year}"
        Header="Year"
        Width="100"/>
    <GridViewColumn DisplayMemberBinding="{Binding Path=Month}"
        Header="Month"
        Width="100"/>
    <GridViewColumn DisplayMemberBinding="{Binding Path=Day}"
        Header="Day"
        Width="100"/>
</GridView>
```

The following example shows the data items that are defined as an [ArrayList](#) of [DateTime](#) objects. The [ArrayList](#) is defined as the [ItemsSource](#) for the [ListView](#) control.

```
<ListView.ItemsSource>
    <s:ArrayList>
        <p:DateTime>1993/1/1 12:22:02</p:DateTime>
        <p:DateTime>1993/1/2 13:2:01</p:DateTime>
        <p:DateTime>1997/1/3 2:1:6</p:DateTime>
        <p:DateTime>1997/1/4 13:6:55</p:DateTime>
        <p:DateTime>1999/2/1 12:22:02</p:DateTime>
        <p:DateTime>1998/2/2 13:2:01</p:DateTime>
        <p:DateTime>2000/2/3 2:1:6</p:DateTime>
        <p:DateTime>2002/2/4 13:6:55</p:DateTime>
        <p:DateTime>2001/3/1 12:22:02</p:DateTime>
        <p:DateTime>2006/3/2 13:2:01</p:DateTime>
        <p:DateTime>2004/3/3 2:1:6</p:DateTime>
        <p:DateTime>2004/3/4 13:6:55</p:DateTime>
    </s:ArrayList>
</ListView.ItemsSource>
```

The `s` and `p` identifiers in the XAML tags refer to namespace mappings that are defined in the metadata of the XAML page. The following example shows the metadata definition.

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="ListViewSort.Window1"
    xmlns:s="clr-namespace:System.Collections;assembly=mscorlib"
    xmlns:p="clr-namespace:System;assembly=mscorlib">
```

To sort the data according to the contents of a column, the example defines an event handler to handle the [Click](#) event that occurs when you press the column header button. The following example shows how to specify an event handler for the [GridViewColumnHeader](#) control.

```
<ListView x:Name='lv' Height="150" HorizontalAlignment="Center"  
VerticalAlignment="Center"  
GridViewColumnHeader.Click="GridViewColumnHeaderClickedHandler"  
>
```

The example defines the event handler so that the sort direction changes between ascending order and descending order each time you press the column header button. The following example shows the event handler.

```

public partial class Window1 : Window
{
    public Window1()
    {
        InitializeComponent();
    }

    GridViewColumnHeader _lastHeaderClicked = null;
    ListSortDirection _lastDirection = ListSortDirection.Ascending;

    void GridViewColumnHeaderClickedHandler(object sender,
                                              RoutedEventArgs e)
    {
        var headerClicked = e.OriginalSource as GridViewColumnHeader;
        ListSortDirection direction;

        if (headerClicked != null)
        {
            if (headerClicked.Role != GridViewColumnHeaderRole.Padding)
            {
                if (headerClicked != _lastHeaderClicked)
                {
                    direction = ListSortDirection.Ascending;
                }
                else
                {
                    if (_lastDirection == ListSortDirection.Ascending)
                    {
                        direction = ListSortDirection.Descending;
                    }
                    else
                    {
                        direction = ListSortDirection.Ascending;
                    }
                }
            }

            var columnBinding = headerClicked.Column.DisplayMemberBinding as Binding;
            var sortBy = columnBinding?.Path.Path ?? headerClicked.Column.Header as string;

            Sort(sortBy, direction);

            if (direction == ListSortDirection.Ascending)
            {
                headerClicked.Column.HeaderTemplate =
                    Resources["HeaderTemplateArrowUp"] as DataTemplate;
            }
            else
            {
                headerClicked.Column.HeaderTemplate =
                    Resources["HeaderTemplateArrowDown"] as DataTemplate;
            }

            // Remove arrow from previously sorted header
            if (_lastHeaderClicked != null && _lastHeaderClicked != headerClicked)
            {
                _lastHeaderClicked.Column.HeaderTemplate = null;
            }

            _lastHeaderClicked = headerClicked;
            _lastDirection = direction;
        }
    }
}

```

```

Partial Public Class Window1
    Inherits Window
    Public Sub New()
        InitializeComponent()
    End Sub

    Private _lastHeaderClicked As GridViewColumnHeader = Nothing
    Private _lastDirection As ListSortDirection = ListSortDirection.Ascending

    Private Sub GridViewColumnHeaderClickedHandler(ByVal sender As Object, ByVal e As RoutedEventArgs)
        Dim headerClicked = TryCast(e.OriginalSource, GridViewColumnHeader)
        Dim direction As ListSortDirection

        If headerClicked IsNot Nothing Then
            If headerClicked.Role <> GridViewColumnHeaderRole.Padding Then
                If headerClicked IsNot _lastHeaderClicked Then
                    direction = ListSortDirection.Ascending
                Else
                    If _lastDirection = ListSortDirection.Ascending Then
                        direction = ListSortDirection.Descending
                    Else
                        direction = ListSortDirection.Ascending
                    End If
                End If

                Dim columnBinding = TryCast(headerClicked.Column.DisplayMemberBinding, Binding)
                Dim sortBy = If(columnBinding?.Path.Path, TryCast(headerClicked.Column.Header, String))

                Sort(sortBy, direction)

                If direction = ListSortDirection.Ascending Then
                    headerClicked.Column.HeaderTemplate = TryCast(Resources("HeaderTemplateArrowUp"),
DataTemplate)
                Else
                    headerClicked.Column.HeaderTemplate = TryCast(Resources("HeaderTemplateArrowDown"),
DataTemplate)
                End If

                ' Remove arrow from previously sorted header
                If _lastHeaderClicked IsNot Nothing AndAlso _lastHeaderClicked IsNot headerClicked Then
                    _lastHeaderClicked.Column.HeaderTemplate = Nothing
                End If

                _lastHeaderClicked = headerClicked
                _lastDirection = direction
            End If
        End If
    End Sub
End Class

```

The following example shows the sorting algorithm that is called by the event handler to sort the data. The sort is performed by creating a new [SortDescription](#) structure.

```

private void Sort(string sortBy, ListSortDirection direction)
{
    ICollectionView dataView =
        CollectionViewSource.GetDefaultView(lv.ItemsSource);

    dataView.SortDescriptions.Clear();
    SortDescription sd = new SortDescription(sortBy, direction);
    dataView.SortDescriptions.Add(sd);
    dataView.Refresh();
}

```

```
Private Sub Sort(ByVal sortBy As String, ByVal direction As ListSortDirection)
    Dim dataView As ICollectionView = CollectionViewSource.GetDefaultView(lv.ItemsSource)

    dataView.SortDescriptions.Clear()
    Dim sd As New SortDescription(sortBy, direction)
    dataView.SortDescriptions.Add(sd)
    dataView.Refresh()
End Sub
```

## See also

- [ListView](#)
- [GridView](#)
- [ListView Overview](#)
- [GridView Overview](#)
- [How-to Topics](#)

# How to: Create a Custom View Mode for a ListView

2 minutes to read • [Edit Online](#)

This example shows how to create a custom [View](#) mode for a [ListView](#) control.

## Example

You must use the [ViewBase](#) class when you create a custom view for the [ListView](#) control. The following example shows a view mode that is called `PlainView`, which is derived from the [ViewBase](#) class.

```
public class PlainView : ViewBase
{
    public static readonly DependencyProperty
        ItemContainerStyleProperty =
        ItemsControl.ItemContainerStyleProperty.AddOwner(typeof(PlainView));

    public Style ItemContainerStyle
    {
        get { return (Style)GetValue(ItemContainerStyleProperty); }
        set { SetValue(ItemContainerStyleProperty, value); }
    }

    public static readonly DependencyProperty ItemTemplateProperty =
        ItemsControl.ItemTemplateProperty.AddOwner(typeof(PlainView));

    public DataTemplate ItemTemplate
    {
        get { return (DataTemplate)GetValue(ItemTemplateProperty); }
        set { SetValue(ItemTemplateProperty, value); }
    }

    public static readonly DependencyProperty ItemWidthProperty =
        WrapPanel.ItemWidthProperty.AddOwner(typeof(PlainView));

    public double ItemWidth
    {
        get { return (double)GetValue(ItemWidthProperty); }
        set { SetValue(ItemWidthProperty, value); }
    }

    public static readonly DependencyProperty ItemHeightProperty =
        WrapPanel.ItemHeightProperty.AddOwner(typeof(PlainView));

    public double ItemHeight
    {
        get { return (double)GetValue(ItemHeightProperty); }
        set { SetValue(ItemHeightProperty, value); }
    }

    protected override object DefaultStyleKey
    {
        get
        {
            return new ComponentResourceKey(GetType(), "myPlainViewDSK");
        }
    }
}
```

```

Public Class PlainView
    Inherits ViewBase

    Public Shared ReadOnly ItemContainerStyleProperty As DependencyProperty =
        ItemsControl.ItemContainerStyleProperty.AddOwner(GetType(PlainView))

    Public Property ItemContainerStyle() As Style
        Get
            Return CType(GetValue(ItemContainerStyleProperty), Style)
        End Get
        Set(ByVal value As Style)
            SetValue(ItemContainerStyleProperty, value)
        End Set
    End Property

    Public Shared ReadOnly ItemTemplateProperty As DependencyProperty =
        ItemsControl.ItemTemplateProperty.AddOwner(GetType(PlainView))

    Public Property ItemTemplate() As DataTemplate
        Get
            Return CType(GetValue(ItemTemplateProperty), DataTemplate)
        End Get
        Set(ByVal value As DataTemplate)
            SetValue(ItemTemplateProperty, value)
        End Set
    End Property

    Public Shared ReadOnly ItemWidthProperty As DependencyProperty =
        WrapPanel.ItemWidthProperty.AddOwner(GetType(PlainView))

    Public Property ItemWidth() As Double
        Get
            Return CDbl(GetValue(ItemWidthProperty))
        End Get
        Set(ByVal value As Double)
            SetValue(ItemWidthProperty, value)
        End Set
    End Property

    Public Shared ReadOnly ItemHeightProperty As DependencyProperty =
        WrapPanel.ItemHeightProperty.AddOwner(GetType(PlainView))

    Public Property ItemHeight() As Double
        Get
            Return CDbl(GetValue(ItemHeightProperty))
        End Get
        Set(ByVal value As Double)
            SetValue(ItemHeightProperty, value)
        End Set
    End Property

    Protected Overrides ReadOnly Property DefaultStyleKey() As Object
        Get
            Return New ComponentResourceKey(Me.GetType(), "myPlainViewDSK")
        End Get
    End Property

End Class

```

To apply a style to the custom view, use the [Style](#) class. The following example defines a [Style](#) for the [PlainView](#) view mode. In the previous example, this style is set as the value of the [DefaultStyleKey](#) property that is defined for [PlainView](#).

```

<Style x:Key="{ComponentResourceKey
    TypeInTargetAssembly={x:Type l:PlainView},
    ResourceId=myPlainViewDSK}"
    TargetType="{x:Type ListView}"
    BasedOn="{StaticResource {x:Type ListBox}}"
>
<Setter Property="HorizontalContentAlignment"
    Value="Center"/>
<Setter Property="ItemContainerStyle"
    Value="{Binding (ListView.View).ItemContainerStyle,
    RelativeSource={RelativeSource Self}}"/>
<Setter Property="ItemTemplate"
    Value="{Binding (ListView.View).ItemTemplate,
    RelativeSource={RelativeSource Self}}"/>
<Setter Property="ItemsPanel">
<Setter.Value>
    <ItemsPanelTemplate>
        <WrapPanel Width="{Binding (FrameworkElement.ActualWidth),
            RelativeSource={RelativeSource
                AncestorType=ScrollContentPresenter}}"
            ItemWidth="{Binding (ListView.View).ItemWidth,
            RelativeSource={RelativeSource AncestorType=ListView}}"
            MinWidth="{Binding (ListView.View).ItemWidth,
            RelativeSource={RelativeSource AncestorType=ListView}}"
            ItemHeight="{Binding (ListView.View).ItemHeight,
            RelativeSource={RelativeSource AncestorType=ListView}}"/>
    </ItemsPanelTemplate>
</Setter.Value>
</Setter>
</Style>

```

To define the layout of data in a custom view mode, define a [DataTemplate](#) object. The following example defines a [DataTemplate](#) that can be used to display data in the [PlainView](#) view mode.

```

<DataTemplate x:Key="centralTile">
    <StackPanel Height="100" Width="90">
        <Grid Width="70" Height="70" HorizontalAlignment="Center">
            <Image Source="{Binding XPath=@Image}" Margin="6,6,6,9"/>
        </Grid>
        <TextBlock Text="{Binding XPath=@Name}" FontSize="13"
            HorizontalAlignment="Center" Margin="0,0,0,1" />
        <TextBlock Text="{Binding XPath=@Type}" FontSize="9"
            HorizontalAlignment="Center" Margin="0,0,0,1" />
    </StackPanel>
</DataTemplate>

```

The following example shows how to define a [ResourceKey](#) for the [PlainView](#) view mode that uses the [DataTemplate](#) that is defined in the previous example.

```

<l:PlainView x:Key="tileView"
    ItemTemplate="{StaticResource centralTile}"
    ItemWidth="100"/>

```

A [ListView](#) control can use a custom view if you set the [View](#) property to the resource key. The following example shows how to specify [PlainView](#) as the view mode for a [ListView](#).

```

//Set the ListView View property to the tileView custom view
lv.View = lv.FindResource("tileView") as ViewBase;

```

```
'Set the ListView View property to the tileView custom view  
lv.View = TryCast(lv.FindResource("tileView"), ViewBase)
```

For the complete sample, see [ListView with Multiple Views\(C#\)](#) or [ListView with Multiple Views\(Visual Basic\)](#).

## See also

- [ListView](#)
- [GridView](#)
- [How-to Topics](#)
- [ListView Overview](#)
- [GridView Overview](#)

# How to: Use Templates to Style a ListView That Uses GridView

2 minutes to read • [Edit Online](#)

This example shows how to use the [DataTemplate](#) and [Style](#) objects to specify the appearance of a [ListView](#) control that uses a [GridView](#) view mode.

## Example

The following examples show [Style](#) and [DataTemplate](#) objects that customize the appearance of a column header for a [GridViewColumn](#).

```
<Style x:Key="myHeaderStyle" TargetType="{x:Type GridViewColumnHeader}">
    <Setter Property="Background" Value="LightBlue"/>
</Style>
```

```
<DataTemplate x:Key="myHeaderTemplate">
    <DockPanel>
        <CheckBox/>
        <TextBlock FontSize="16" Foreground="DarkBlue">
            <TextBlock.Text>
                <Binding/>
            </TextBlock.Text>
        </TextBlock>
    </DockPanel>
</DataTemplate>
```

The following example shows how to use these [Style](#) and [DataTemplate](#) objects to set the [HeaderContainerStyle](#) and [HeaderTemplate](#) properties of a [GridViewColumn](#). The [DisplayMemberBinding](#) property defines the content of the column cells.

```
<GridViewColumn Header="Month" Width="80"
    HeaderContainerStyle="{StaticResource myHeaderStyle}"
    HeaderTemplate="{StaticResource myHeaderTemplate}"
    DisplayMemberBinding="{Binding Path=Month}"/>
```

The [HeaderContainerStyle](#) and [HeaderTemplate](#) are only two of several properties that you can use to customize column header appearance for a [GridView](#) control. For more information, see [GridView Column Header Styles and Templates Overview](#).

The following example shows how to define a [DataTemplate](#) that customizes the appearance of the cells in a [GridViewColumn](#).

```
<DataTemplate x:Key="myCellTemplateMonth">
  <DockPanel>
    <TextBlock Foreground="DarkBlue" HorizontalAlignment="Center">
      <TextBlock.Text>
        <Binding Path="Month"/>
      </TextBlock.Text>
    </TextBlock>
  </DockPanel>
</DataTemplate>
```

The following example shows how to use this [DataTemplate](#) to define the content of a [GridViewColumn](#) cell. This template is used instead of the [DisplayMemberBinding](#) property that is shown in the previous [GridViewColumn](#) example.

```
<GridViewColumn Header="Month" Width="80"
  CellTemplate="{StaticResource myCellTemplateMonth}" />
```

## See also

- [ListView](#)
- [GridView](#)
- [GridView Overview](#)
- [How-to Topics](#)
- [ListView Overview](#)

# How to: Create a Style for a Dragged GridView Column Header

2 minutes to read • [Edit Online](#)

This example shows how to change the appearance of a dragged `GridViewColumnHeader` when the user changes the position of a column.

## Example

When you drag a column header to another location in a `ListView` that uses `GridView` for its view mode, the column moves to the new position. While you are dragging the column header, a floating copy of the header appears in addition to the original header. A column header in a `GridView` is represented by a `GridViewColumnHeader` object.

To customize the appearance of both the floating and original headers, you can set `Triggers` to modify the `GridViewColumnHeader` `Style`. These `Triggers` are applied when the `IsPressed` property value is `true` and the `Role` property value is `Floating`.

When the user presses the mouse button and holds it down while the mouse pauses on the `GridViewColumnHeader`, the `IsPressed` property value changes to `true`. Likewise, when the user begins the drag operation, the `Role` property changes to `Floating`.

The following example shows how to set `Triggers` to change the `Foreground` and `Background` colors of the original and floating headers when the user drags a column to a new position.

```
<ControlTemplate TargetType="{x:Type GridViewColumnHeader}">
```

```
<ControlTemplate.Triggers>
```

```
<Trigger Property="IsPressed"
    Value="true">
    <Setter TargetName="HighlightBorder"
        Property="Visibility"
        Value="Hidden"/>
    <Setter TargetName="PART_HeaderGripper"
        Property="Visibility"
        Value="Hidden"/>
    <Setter Property="Background"
        Value="SkyBlue"/>
    <Setter Property="Foreground"
        Value="Yellow"/>
</Trigger>
```

```
<Trigger Property="Role"  
        Value="Floating">  
    <Setter TargetName="PART_HeaderGripper"  
            Property="Visibility"  
            Value="Collapsed"/>  
    <Setter Property="Background"  
            Value="Yellow"/>  
    <Setter Property="Foreground"  
            Value="SkyBlue"/>  
</Trigger>
```

```
</ControlTemplate.Triggers>
```

```
</ControlTemplate>
```

## See also

- [GridViewColumnHeader](#)
- [GridViewColumnHeaderRole](#)
- [ListView](#)
- [GridView](#)
- [How-to Topics](#)
- [ListView Overview](#)
- [GridView Overview](#)

# How to: Display ListView Contents by Using a GridView

2 minutes to read • [Edit Online](#)

This example shows how to define a **GridView** view mode for a **ListView** control.

## Example

You can define the view mode of a **GridView** by specifying **GridViewColumn** objects. The following example shows how to define **GridViewColumn** objects that bind to the data content that is specified for the **ListView** control. This **GridView** example specifies three **GridViewColumn** objects that map to the `FirstName`, `LastName`, and `EmployeeNumber` fields of the `EmployeeInfoDataSource` that is set as the `ItemsSource` of the **ListView** control.

```
<ListView ItemsSource="{Binding Source=
    {StaticResource EmployeeInfoDataSource}}">

    <ListView.View>

        <GridView AllowsColumnReorder="true"
            ColumnHeaderToolTip="Employee Information">

            <GridViewColumn DisplayMemberBinding=
                "{Binding Path=FirstName}"
                Header="First Name" Width="100"/>

            <GridViewColumn DisplayMemberBinding=
                "{Binding Path=LastName}"
                Width="100">
                <GridViewColumnHeader>Last Name
                    <GridViewColumnHeader.ContextMenu>
                        <ContextMenu MenuItem.Click="LastNameCM_Click"
                            Name="LastNameCM">
                            <MenuItem Header="Ascending" />
                            <MenuItem Header="Descending" />
                        </ContextMenu>
                    </GridViewColumnHeader.ContextMenu>
                </GridViewColumnHeader>
            </GridViewColumn>

            <GridViewColumn DisplayMemberBinding=
                "{Binding Path=EmployeeNumber}"
                Header="Employee No." Width="100"/>
        </GridView>

    </ListView.View>
</ListView>
```

The following illustration shows how this example appears.

First Name	Last Name	Employee No.
Jesper	Aaberg	12345
Dominik	Paiha	98765
Yale	Li	23875
Muru	Subramani	49392

## See also

- [ListView](#)
- [GridView](#)
- [ListView Overview](#)
- [GridView Overview](#)
- [How-to Topics](#)

# How to: Use Triggers to Style Selected Items in a ListView

2 minutes to read • [Edit Online](#)

This example shows how to define [Triggers](#) for a [ListViewItem](#) control so that when a property value of a [ListViewItem](#) changes, the [Style](#) of the [ListViewItem](#) changes in response.

## Example

If you want the [Style](#) of a [ListViewItem](#) to change in response to property changes, define [Triggers](#) for the [Style](#) change.

The following example defines a [Trigger](#) that sets the [Foreground](#) property to [Blue](#) and changes the [Cursor](#) to display a [Hand](#) when the [IsMouseOver](#) property changes to `true`.

```
<Style x:Key="MyContainer" TargetType="{x:Type ListViewItem}">

    <Setter Property="Margin" Value="0,1,0,0"/>
    <Setter Property="Height" Value="21"/>

    <Style.Triggers>

        <Trigger Property="IsMouseOver" Value="true">
            <Setter Property="Foreground" Value="Blue" />
            <Setter Property="Cursor" Value="Hand"/>
        </Trigger>

    </Style.Triggers>
</Style>
```

The following example defines a [MultiTrigger](#) that sets the [Foreground](#) property of a [ListViewItem](#) to [Yellow](#) when the [ListViewItem](#) is the selected item and has keyboard focus.

```
<Style x:Key="MyContainer" TargetType="{x:Type ListViewItem}">

    <Setter Property="Margin" Value="0,1,0,0"/>
    <Setter Property="Height" Value="21"/>

    <Style.Triggers>

        <MultiTrigger>
            <MultiTrigger.Conditions>
                <Condition Property="IsSelected" Value="true" />
                <Condition Property="Selector.IsSelectionActive" Value="true" />
            </MultiTrigger.Conditions>
            <Setter Property="Foreground" Value="Yellow" />
        </MultiTrigger>
    </Style.Triggers>
</Style>
```

```
</Style.Triggers>  
</Style>
```

## See also

- [Control](#)
- [ListView](#)
- [GridView](#)
- [How-to Topics](#)
- [ListView Overview](#)
- [GridView Overview](#)

# How to: Create ListViewItems with a CheckBox

2 minutes to read • [Edit Online](#)

This example shows how to display a column of **CheckBox** controls in a **ListView** control that uses a **GridView**.

## Example

To create a column that contains **CheckBox** controls in a **ListView**, create a **DataTemplate** that contains a **CheckBox**. Then set the **CellTemplate** of a **GridViewColumn** to the **DataTemplate**.

The following example shows a **DataTemplate** that contains a **CheckBox**. The example binds the **.IsChecked** property of the **CheckBox** to the **IsSelected** property value of the **ListViewItem** that contains it. Therefore, when the **ListViewItem** that contains the **CheckBox** is selected, the **CheckBox** is checked.

```
<DataTemplate x:Key="FirstCell">
    <StackPanel Orientation="Horizontal">
        <CheckBox IsChecked="{Binding Path=IsSelected,
            RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type ListViewItem}}}" />
    </StackPanel>
</DataTemplate>
```

The following example shows how to create a column of **CheckBox** controls. To make the column, the example sets the **CellTemplate** property of the **GridViewColumn** to the **DataTemplate**.

```
<GridViewColumn CellTemplate="{StaticResource FirstCell}"
    Width="30"/>
```

## See also

- [Control](#)
- [ListView](#)
- [GridView](#)
- [ListView Overview](#)
- [How-to Topics](#)
- [GridView Overview](#)

# How to: Display Data by Using GridViewRowPresenter

2 minutes to read • [Edit Online](#)

This example shows how to use the [GridViewRowPresenter](#) and [GridViewHeaderRowPresenter](#) objects to display data in columns.

## Example

The following example shows how to specify a [GridViewColumnCollection](#) that displays the [DayOfWeek](#) and [Year](#) of a [DateTime](#) object by using [GridViewRowPresenter](#) and [GridViewHeaderRowPresenter](#) objects. The example also defines a [Style](#) for the [Header](#) of a [GridViewColumn](#).

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x='http://schemas.microsoft.com/winfx/2006/xaml'
    xmlns:sys="clr-namespace:System;assembly=mscorlib">

    <Window.Resources>

        <Style x:Key="MyHeaderStyle" TargetType="{x:Type GridViewColumnHeader}">
            <Setter Property="Background" Value="BurlyWood"/>
        </Style>

        <GridViewColumnCollection x:Key="gvcc">
            <GridViewColumn Header="Year"
                DisplayMemberBinding="{Binding Year}"
                Width="80"/>
            <GridViewColumn Header="Day"
                DisplayMemberBinding="{Binding DayOfWeek}"
                Width="80" />
        </GridViewColumnCollection>
    </Window.Resources>

    <StackPanel>
        <GridViewHeaderRowPresenter Name="hrp" Columns="{StaticResource gvcc}"
            ColumnHeaderContainerStyle=
            "{StaticResource MyHeaderStyle}" />

        <GridViewRowPresenter Columns="{StaticResource gvcc}" >
            <GridViewRowPresenter.Content>
                <sys:DateTime>2005/2/1</sys:DateTime>
            </GridViewRowPresenter.Content>
        </GridViewRowPresenter>
        <GridViewRowPresenter Columns="{StaticResource gvcc}" >
            <GridViewRowPresenter.Content>
                <sys:DateTime>2006/10/12</sys:DateTime>
            </GridViewRowPresenter.Content>
        </GridViewRowPresenter>
    </StackPanel>
</Window>
```

## See also

- [GridViewHeaderRowPresenter](#)
- [GridViewRowPresenter](#)
- [GridViewColumnCollection](#)
- [GridView Overview](#)

# How to: Group Items in a ListView That Implements a GridView

2 minutes to read • [Edit Online](#)

This example shows how to display groups of items in the [GridView](#) view mode of a [ListView](#) control.

## Example

To display groups of items in a [ListView](#), define a [CollectionViewSource](#). The following example shows a [CollectionViewSource](#) that groups data items according to the value of the `Catalog` data field.

```
<CollectionViewSource x:Key='src'
    Source="{Binding Source={StaticResource MyData},
        XPath=Item}">
<CollectionViewSource.GroupDescriptions>
    <PropertyGroupDescription PropertyName="@Catalog" />
</CollectionViewSource.GroupDescriptions>
</CollectionViewSource>
```

The following example sets the [ItemsSource](#) for the [ListView](#) to the [CollectionViewSource](#) that the previous example defines. The example also defines a [GroupStyle](#) that implements an [Expander](#) control.

```
<ListView ItemsSource='{Binding Source={StaticResource src}}'
    BorderThickness="0">
<ListView.GroupStyle>
    <GroupStyle>
        <GroupStyle.ContainerStyle>
            <Style TargetType="{x:Type GroupItem}">
                <Setter Property="Margin" Value="0,0,0,5"/>
                <Setter Property="Template">
                    <Setter.Value>
                        <ControlTemplate TargetType="{x:Type GroupItem}">
                            <Expander IsExpanded="True" BorderBrush="#FFA4B97F"
                                BorderThickness="0,0,0,1">
                                <Expander.Header>
                                    <DockPanel>
                                        <TextBlock FontWeight="Bold" Text="{Binding Path=Name}"
                                            Margin="5,0,0,0" Width="100"/>
                                        <TextBlock FontWeight="Bold"
                                            Text="{Binding Path=ItemCount}"/>
                                    </DockPanel>
                                </Expander.Header>
                                <Expander.Content>
                                    <ItemsPresenter />
                                </Expander.Content>
                            </Expander>
                        </ControlTemplate>
                    </Setter.Value>
                </Setter>
            </Style>
        </GroupStyle.ContainerStyle>
    </GroupStyle>
</ListView.GroupStyle>
```

```
</ListView>
```

## See also

- [ListView](#)
- [GridView](#)
- [How-to Topics](#)
- [ListView Overview](#)
- [GridView Overview](#)

# How to: Style a Row in a ListView That Implements a GridView

2 minutes to read • [Edit Online](#)

This example shows how to style a row in a [ListView](#) control that implements a [GridViewView](#) mode.

## Example

You can style a row in a [ListView](#) control by setting an [ItemContainerStyle](#) on the [ListView](#) control. Set the style for its items that are represented as [ListViewItem](#) objects. The [ItemContainerStyle](#) references the [ControlTemplate](#) objects that are used to display the row content.

The complete sample, which the following examples are extracted from, displays a collection of song information that is stored in an XML database. Each song in the database has a rating field and the value of this field specifies how to display a row of song information.

The following example shows how to define [ItemContainerStyle](#) for the [ListViewItem](#) objects that represent the songs in the song collection. The [ItemContainerStyle](#) references [ControlTemplate](#) objects that specify how to display a row of song information.

```
<ListView.ItemContainerStyle>
  <Style TargetType="{x:Type ListViewItem}" >
    <Setter Property="Template"
      Value="{StaticResource Default}"/>
    <Style.Triggers>
      <DataTrigger Binding="{Binding XPath=@Rating}" Value="5">
        <Setter Property="Template"
          Value="{StaticResource StronglyRecommended}"/>
      </DataTrigger>
      <DataTrigger Binding="{Binding XPath=@Rating}" Value="4">
        <Setter Property="Template"
          Value="{StaticResource Recommended}"/>
      </DataTrigger>
    </Style.Triggers>
  </Style>
</ListView.ItemContainerStyle>
```

The following example shows a [ControlTemplate](#) that adds the text string "Strongly Recommended" to the row. This template is referenced in the [ItemContainerStyle](#) and displays when the song's rating has a value of 5 (five). The [ControlTemplate](#) includes a [GridViewRowPresenter](#) object that lays out the contents of the row in columns as defined by the [GridView](#) view mode.

```
<ControlTemplate x:Key="StronglyRecommended"
  TargetType='{x:Type ListViewItem}'>
  <StackPanel Background="Beige">
    <GridViewRowPresenter Content="{TemplateBinding Content}"
      Columns="{TemplateBinding GridView.ColumnCollection}"/>
    <TextBlock Background="LightBlue" Text="Strongly Recommended" />
  </StackPanel>
</ControlTemplate>
```

The following example defines [GridView](#).

```
<ListView.View>
  <GridView ColumnHeaderContainerStyle="{StaticResource MyHeaderStyle}">
    <GridViewColumn Header="Name"
      DisplayMemberBinding="{Binding XPath=@Name}"
      Width="100"/>
    <GridViewColumn Header="Time"
      DisplayMemberBinding="{Binding XPath=@Time}"
      Width="80"/>
    <GridViewColumn Header="Artist"
      DisplayMemberBinding="{Binding XPath=@Artist}"
      Width="80" />
    <GridViewColumn Header="Disk"
      DisplayMemberBinding="{Binding XPath=@Disk}"
      Width="100"/>
  </GridView>
</ListView.View>
```

## See also

- [ListView](#)
- [GridView](#)
- [How-to Topics](#)
- [ListView Overview](#)
- [Styling and Templating](#)

# How to: Change the Horizontal Alignment of a Column in a ListView

2 minutes to read • [Edit Online](#)

By default, the content of each column in a [ListViewItem](#) is left-aligned. You can change the alignment of each column by providing a [DataTemplate](#) and setting the [HorizontalAlignment](#) property on the element within the [DataTemplate](#). This topic shows how a [ListView](#) aligns its content by default and how to change the alignment of one column in a [ListView](#).

## Example

In the following example, the data in the `Title` and `ISBN` columns is left-aligned.

```
<!--XmlDataProvider is defined in a ResourceDictionary,  
such as Window.Resources-->  
<XmlDataProvider x:Key="InventoryData" XPath="Books">  
    <x:XData>  
        <Books xmlns="">  
            <Book ISBN="0-7356-0562-9" Stock="in" Number="9">  
                <Title>XML in Action</Title>  
                <Summary>XML Web Technology</Summary>  
            </Book>  
            <Book ISBN="0-7356-1370-2" Stock="in" Number="8">  
                <Title>Programming Microsoft Windows With C#</Title>  
                <Summary>C# Programming using the .NET Framework</Summary>  
            </Book>  
            <Book ISBN="0-7356-1288-9" Stock="out" Number="7">  
                <Title>Inside C#</Title>  
                <Summary>C# Language Programming</Summary>  
            </Book>  
            <Book ISBN="0-7356-1377-X" Stock="in" Number="5">  
                <Title>Introducing Microsoft .NET</Title>  
                <Summary>Overview of .NET Technology</Summary>  
            </Book>  
            <Book ISBN="0-7356-1448-2" Stock="out" Number="4">  
                <Title>Microsoft C# Language Specifications</Title>  
                <Summary>The C# language definition</Summary>  
            </Book>  
        </Books>  
    </x:XData>  
</XmlDataProvider>
```

```
<ListView ItemsSource="{Binding Source={StaticResource InventoryData}, XPath=Book}">  
    <ListView.View>  
        <GridView>  
            <GridViewColumn Width="300" Header="Title"  
                DisplayMemberBinding="{Binding XPath=Title}"/>  
            <GridViewColumn Width="150" Header="ISBN"  
                DisplayMemberBinding="{Binding XPath=@ISBN}"/>  
        </GridView>  
    </ListView.View>  
</ListView>
```

To change the alignment of the `ISBN` column, you need to specify that the [HorizontalContentAlignment](#) property of each [ListViewItem](#) is [Stretch](#), so that the elements in each [ListViewItem](#) can span or be positioned along the

entire width of each column. Because the [ListView](#) is bound to a data source, you need to create a style that sets the [HorizontalContentAlignment](#). Next, you need to use a [DataTemplate](#) to display the content instead of using the [DisplayMemberBinding](#) property. To display the [ISBN](#) of each template, the [DataTemplate](#) can just contain a [TextBlock](#) that has its [HorizontalAlignment](#) property set to [Right](#).

The following example defines the style and [DataTemplate](#) necessary to make the [ISBN](#) column right-aligned, and changes the [GridViewColumn](#) to reference the [DataTemplate](#).

```
<!--The Style and DataTemplate are defined in a ResourceDictionary,  
such as Window.Resources-->  
<Style TargetType="ListViewItem">  
    <Setter Property="HorizontalContentAlignment" Value="Stretch"/>  
</Style>  
  
<DataTemplate x:Key="ISBNTemplate">  
    <TextBlock HorizontalAlignment="Right"  
        Text="{Binding XPath=@ISBN}"/>  
</DataTemplate>
```

```
<ListView ItemsSource="{Binding Source={StaticResource InventoryData}, XPath=Book}">  
    <ListView.View>  
        <GridView>  
            <GridViewColumn Width="300" Header="Title"  
                DisplayMemberBinding="{Binding XPath=Title}"/>  
            <GridViewColumn Width="150" Header="ISBN"  
                CellTemplate="{StaticResource ISBNTemplate}"/>  
        </GridView>  
    </ListView.View>  
</ListView>
```

## See also

- [Data Binding Overview](#)
- [Data Templating Overview](#)
- [Bind to XML Data Using an XMLDataProvider and XPath Queries](#)
- [ListView Overview](#)

# How to: Handle the MouseDoubleClick Event for Each Item in a ListView

2 minutes to read • [Edit Online](#)

To handle an event for an item in a [ListView](#), you need to add an event handler to each [ListViewItem](#). When a [ListView](#) is bound to a data source, you don't explicitly create a [ListViewItem](#), but you can handle the event for each item by adding an [EventSetter](#) to a style of a [ListViewItem](#).

## Example

The following example creates a data-bound [ListView](#) and creates a [Style](#) to add an event handler to each [ListViewItem](#).

```
<!--XmlDataProvider is defined in a ResourceDictionary,  
such as Window.Resources-->  
<XmlDataProvider x:Key="InventoryData" XPath="Books">  
    <x:XData>  
        <Books xmlns="">  
            <Book ISBN="0-7356-0562-9" Stock="in" Number="9">  
                <Title>XML in Action</Title>  
                <Summary>XML Web Technology</Summary>  
            </Book>  
            <Book ISBN="0-7356-1370-2" Stock="in" Number="8">  
                <Title>Programming Microsoft Windows With C#</Title>  
                <Summary>C# Programming using the .NET Framework</Summary>  
            </Book>  
            <Book ISBN="0-7356-1288-9" Stock="out" Number="7">  
                <Title>Inside C#</Title>  
                <Summary>C# Language Programming</Summary>  
            </Book>  
            <Book ISBN="0-7356-1377-X" Stock="in" Number="5">  
                <Title>Introducing Microsoft .NET</Title>  
                <Summary>Overview of .NET Technology</Summary>  
            </Book>  
            <Book ISBN="0-7356-1448-2" Stock="out" Number="4">  
                <Title>Microsoft C# Language Specifications</Title>  
                <Summary>The C# language definition</Summary>  
            </Book>  
        </Books>  
    </x:XData>  
</XmlDataProvider>
```

```
<!--The Style is defined in a ResourceDictionary,  
such as Window.Resources-->  
<Style TargetType="ListViewItem">  
    <EventSetter Event="MouseDoubleClick" Handler="ListViewItem_MouseDoubleClick"/>  
</Style>
```

```

<ListView ItemsSource="{Binding Source={StaticResource InventoryData}, XPath=Book}">
    <ListView.View>
        <GridView>
            <GridViewColumn Width="300" Header="Title"
                DisplayMemberBinding="{Binding XPath=Title}"/>
            <GridViewColumn Width="150" Header="ISBN"
                DisplayMemberBinding="{Binding XPath=@ISBN}"/>
        </GridView>
    </ListView.View>
</ListView>

```

The following example handles the [MouseDoubleClick](#) event.

```

void ListViewItem_MouseDoubleClick(object sender, MouseButtonEventArgs e)
{
    XmlElement book = ((ListViewItem) sender).Content as XmlElement;

    if (book == null)
    {
        return;
    }

    if (book.GetAttribute("Stock") == "out")
    {
        MessageBox.Show("Time to order more copies of " + book["Title"].InnerText);
    }
    else
    {
        MessageBox.Show(book["Title"].InnerText + " is available.");
    }
}

```

```

Private Sub ListViewItem_MouseDoubleClick(ByVal sender As Object, _
                                         ByVal e As MouseButtonEventArgs)

    Dim lvi As ListViewItem = CType(sender, ListViewItem)
    Dim book As XmlElement = CType(lvi.Content, XmlElement)

    If book.GetAttribute("Stock") = "out" Then
        MessageBox.Show("Time to order more copies of " + book("Title").InnerText)
    Else
        MessageBox.Show(book("Title").InnerText + " is available.")
    End If
End Sub

```

#### NOTE

Although it is most common to bind a [ListView](#) to a data source, you can use a style to add an event handler to each [ListViewItem](#) in a non-data-bound [ListView](#) regardless of whether you explicitly create a [ListViewItem](#). For more information about explicitly and implicitly created [ListViewItem](#) controls, see [ItemsControl](#).

## See also

- [XmlElement](#)
- [Data Binding Overview](#)
- [Styling and Templating](#)
- [Bind to XML Data Using an XMLDataProvider and XPath Queries](#)

- [ListView Overview](#)

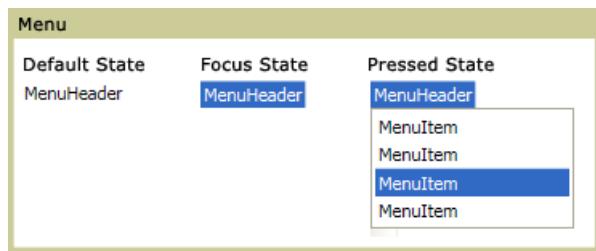
# Menu

2 minutes to read • [Edit Online](#)

A [Menu](#) is a control that allows hierarchical organization of elements associated with commands or event handlers.

Each [Menu](#) can contain multiple [MenuItem](#) controls. Each [MenuItem](#) can invoke a command or invoke a [click](#) event handler. A [MenuItem](#) can also have multiple [MenuItem](#) elements as children, forming a submenu.

The following illustration shows the three different states of a menu control. The default state is when no device such as a mouse pointer is resting on the [Menu](#). The focus state occurs when the mouse pointer is hovering over the [Menu](#) and pressed state occurs when a mouse button is clicked over the [Menu](#).



Menus in different states

## In This Section

[Menu Overview](#)

## Reference

[Menu](#)

[MenuItem](#)

[MenuBase](#)

[ContextMenu](#)

## Related Sections

# Menu Overview

3 minutes to read • [Edit Online](#)

The [Menu](#) class enables you to organize elements associated with commands and event handlers in a hierarchical order. Each [Menu](#) element contains a collection of [MenuItem](#) elements.

## Menu Control

The [Menu](#) control presents a list of items that specify commands or options for an application. Typically, clicking a [MenuItem](#) opens a submenu or causes an application to carry out a command.

## Creating Menus

The following example creates a [Menu](#) to manipulate text in a [TextBox](#). The [Menu](#) contains [MenuItem](#) objects that use the [Command](#), [IsCheckable](#), and [Header](#) properties and the [Checked](#), [Unchecked](#), and [Click](#) events.

```
<Menu>
  <MenuItem Header="_Edit">
    <MenuItem Command="ApplicationCommands.Copy"/>
    <MenuItem Command="ApplicationCommands.Cut"/>
    <MenuItem Command="ApplicationCommands.Paste"/>
  </MenuItem>
  <MenuItem Header="_Font">
    <MenuItem Header="_Bold" IsCheckable="True"
      Checked="Bold_Checked"
      Unchecked="Bold_Unchecked"/>
    <MenuItem Header="_Italic" IsCheckable="True"
      Checked="Italic_Checked"
      Unchecked="Italic_Unchecked"/>
    <Separator/>
    <MenuItem Header="I_ncrease Font Size"
      Click="IncreaseFont_Click"/>
    <MenuItem Header="_Decrease Font Size"
      Click="DecreaseFont_Click"/>
  </MenuItem>
</Menu>
<TextBox Name="textBox1" TextWrapping="Wrap"
  Margin="2">
  The quick brown fox jumps over the lazy dog.
</TextBox>
```

```

private void Bold_Checked(object sender, RoutedEventArgs e)
{
    textBox1.FontWeight = FontWeights.Bold;
}

private void Bold_Unchecked(object sender, RoutedEventArgs e)
{
    textBox1.FontWeight = FontWeights.Normal;
}

private void Italic_Checked(object sender, RoutedEventArgs e)
{
    textBox1.FontStyle = FontStyles.Italic;
}

private void Italic_Unchecked(object sender, RoutedEventArgs e)
{
    textBox1.FontStyle = FontStyles.Normal;
}

private void IncreaseFont_Click(object sender, RoutedEventArgs e)
{
    if (textBox1.FontSize < 18)
    {
        textBox1.FontSize += 2;
    }
}

private void DecreaseFont_Click(object sender, RoutedEventArgs e)
{
    if (textBox1.FontSize > 10)
    {
        textBox1.FontSize -= 2;
    }
}

```

```

Private Sub Bold_Checked(ByVal sender As Object, ByVal e As RoutedEventArgs)
    textBox1.FontWeight = FontWeights.Bold
End Sub

Private Sub Bold_Unchecked(ByVal sender As Object, ByVal e As RoutedEventArgs)
    textBox1.FontWeight = FontWeights.Normal
End Sub

Private Sub Italic_Checked(ByVal sender As Object, ByVal e As RoutedEventArgs)
    textBox1.FontStyle = FontStyles.Italic
End Sub

Private Sub Italic_Unchecked(ByVal sender As Object, ByVal e As RoutedEventArgs)
    textBox1.FontStyle = FontStyles.Normal
End Sub

Private Sub IncreaseFont_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    If textBox1.FontSize < 18 Then
        textBox1.FontSize += 2
    End If
End Sub

Private Sub DecreaseFont_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    If textBox1.FontSize > 10 Then
        textBox1.FontSize -= 2
    End If
End Sub

```

# MenuItems with Keyboard Shortcuts

Keyboard shortcuts are character combinations that can be entered with the keyboard to invoke [Menu](#) commands. For example, the shortcut for **Copy** is CTRL+C. There are two properties to use with keyboard shortcuts and menu items —[InputGestureText](#) or [Command](#).

## InputGestureText

The following example shows how to use the [InputGestureText](#) property to assign keyboard shortcut text to [MenuItem](#) controls. This only places the keyboard shortcut in the menu item. It does not associate the command with the [MenuItem](#). The application must handle the user's input to carry out the action.

```
<MenuItem Header="_Cut" InputGestureText="Ctrl+X"/>
<MenuItem Header="_Find" InputGestureText="Ctrl+F"/>
```

## Command

The following example shows how to use the [Command](#) property to associate the **Open** and **Save** commands with [MenuItem](#) controls. Not only does the command property associate a command with a [MenuItem](#), but it also supplies the input gesture text to use as a shortcut.

```
<MenuItem Header="_Open" Command="ApplicationCommands.Open"/>
<MenuItem Header="_Save" Command="ApplicationCommands.Save"/>
```

The [MenuItem](#) class also has a [CommandTarget](#) property, which specifies the element where the command occurs. If [CommandTarget](#) is not set, the element that has keyboard focus receives the command. For more information about commands, see [Commanding Overview](#).

# Menu Styling

With control styling, you can dramatically change the appearance and behavior of [Menu](#) controls without having to write a custom control. In addition to setting visual properties, you can also apply a [Style](#) to individual parts of a control, change the behavior of parts of the control through properties, or add additional parts or change the layout of a control. The following examples demonstrate several ways to add a [Style](#) to a [Menu](#) control.

The first code example defines a [Style](#) called `Simple` that shows how to use the current system settings in your style. The code assigns the color of the `MenuItemHighlightBrush` as the menu's background color and the `MenuItemTextBrush` as the menu's foreground color. Notice that you use resource keys to assign the brushes.

```
<Style x:Key="Simple" TargetType="{x:Type MenuItem}">
  <Setter Property = "Background" Value= "{DynamicResource {x:Static SystemColors.MenuHighlightBrushKey}}"/>
  <Setter Property = "Foreground" Value= "{DynamicResource {x:Static SystemColors.MenuTextBrushKey}}"/>
  <Setter Property = "Height" Value= "{DynamicResource {x:Static SystemParameters.CaptionHeightKey}}"/>
</Style>
```

The following sample uses [Trigger](#) elements that enable you to change the appearance of a [MenuItem](#) in response to events that occur on the [Menu](#). When you move the mouse over the [Menu](#), the foreground color and the font characteristics of the menu items change.

```
<Style x:Key="Triggers" TargetType="{x:Type MenuItem}">
  <Style.Triggers>
    <Trigger Property="MenuItem.IsMouseOver" Value="true">
      <Setter Property = "Foreground" Value="Red"/>
      <Setter Property = "FontSize" Value="16"/>
      <Setter Property = "FontStyle" Value="Italic"/>
    </Trigger>
  </Style.Triggers>
</Style>
```

## See also

- [WPF Controls Gallery Sample](#)

# Panel

2 minutes to read • [Edit Online](#)

[Panel](#) is the base class for all elements that support application layout in Windows Presentation Foundation (WPF).

## In This Section

[Panels Overview](#)

[How-to Topics](#)

## Reference

[Panel](#)

[Canvas](#)

[DockPanel](#)

[Grid](#)

[StackPanel](#)

[VirtualizingStackPanel](#)

[WrapPanel](#)

## Related Sections

[Layout](#)

[Walkthrough: My first WPF desktop application](#)

[ScrollViewer Overview](#)

# Panels Overview

32 minutes to read • [Edit Online](#)

[Panel](#) elements are components that control the rendering of elements—their size and dimensions, their position, and the arrangement of their child content. The Windows Presentation Foundation (WPF) provides a number of predefined [Panel](#) elements as well as the ability to construct custom [Panel](#) elements.

This topic contains the following sections.

- [The Panel Class](#)
- [Panel Element Common Members](#)
- [Derived Panel Elements](#)
- [User Interface Panels](#)
- [Nested Panel Elements](#)
- [Custom Panel Elements](#)
- [Localization/Globalization Support](#)

## The Panel Class

[Panel](#) is the base class for all elements that provide layout support in Windows Presentation Foundation (WPF). Derived [Panel](#) elements are used to position and arrange elements in Extensible Application Markup Language (XAML) and code.

The WPF includes a comprehensive suite of derived panel implementations that enable many complex layouts. These derived classes expose properties and methods that enable most standard user interface (UI) scenarios. Developers who are unable to find a child arrangement behavior that meets their needs can create new layouts by overriding the [ArrangeOverride](#) and [MeasureOverride](#) methods. For more information on custom layout behaviors, see [Custom Panel Elements](#).

## Panel Common Members

All [Panel](#) elements support the base sizing and positioning properties defined by [FrameworkElement](#), including [Height](#), [Width](#), [HorizontalAlignment](#), [VerticalAlignment](#), [Margin](#), and [LayoutTransform](#). For additional information on positioning properties defined by [FrameworkElement](#), see [Alignment](#), [Margins](#), and [Padding Overview](#).

[Panel](#) exposes additional properties that are of critical importance in understanding and using layout. The [Background](#) property is used to fill the area between the boundaries of a derived panel element with a [Brush](#). [Children](#) represents the child collection of elements that the [Panel](#) is comprised of. [InternalChildren](#) represents the content of the [Children](#) collection plus those members generated by data binding. Both consist of a [UIElementCollection](#) of child elements hosted within the parent [Panel](#).

[Panel](#) also exposes a [Panel.ZIndex](#) attached property that can be used to achieve layered order in a derived [Panel](#). Members of a panel's [Children](#) collection with a higher [Panel.ZIndex](#) value appear in front of those with a lower [Panel.ZIndex](#) value. This is particularly useful for panels such as [Canvas](#) and [Grid](#) which allow children to share the same coordinate space.

[Panel](#) also defines the [OnRender](#) method, which can be used to override the default presentation behavior of

a [Panel](#).

#### Attached Properties

Derived panel elements make extensive use of attached properties. An attached property is a specialized form of dependency property that does not have the conventional common language runtime (CLR) property "wrapper". Attached properties have a specialized syntax in Extensible Application Markup Language (XAML), which can be seen in several of the examples that follow.

One purpose of an attached property is to allow child elements to store unique values of a property that is actually defined by a parent element. An application of this functionality is having child elements inform the parent how they wish to be presented in the user interface (UI), which is extremely useful for application layout. For more information, see [Attached Properties Overview](#).

## Derived Panel Elements

Many objects derive from [Panel](#), but not all of them are intended for use as root layout providers. There are six defined panel classes ([Canvas](#), [DockPanel](#), [Grid](#), [StackPanel](#), [VirtualizingStackPanel](#), and [WrapPanel](#)) that are designed specifically for creating application UI.

Each panel element encapsulates its own special functionality, as seen in the following table.

ELEMENT NAME	UI PANEL?	DESCRIPTION
<a href="#">Canvas</a>	Yes	Defines an area within which you can explicitly position child elements by coordinates relative to the <a href="#">Canvas</a> area.
<a href="#">DockPanel</a>	Yes	Defines an area within which you can arrange child elements either horizontally or vertically, relative to each other.
<a href="#">Grid</a>	Yes	Defines a flexible grid area consisting of columns and rows. Child elements of a <a href="#">Grid</a> can be positioned precisely using the <a href="#">Margin</a> property.
<a href="#">StackPanel</a>	Yes	Arranges child elements into a single line that can be oriented horizontally or vertically.
<a href="#">TabPanel</a>	No	Handles the layout of tab buttons in a <a href="#">TabControl</a> .
<a href="#">ToolBarOverflowPanel</a>	No	Arranges content within a <a href="#">ToolBar</a> control.
<a href="#">UniformGrid</a>	No	<a href="#">UniformGrid</a> is used to arrange children in a grid with all equal cell sizes.
<a href="#">VirtualizingPanel</a>	No	Provides a base class for panels that can "virtualize" their children collection.

ELEMENT NAME	UI PANEL?	DESCRIPTION
<a href="#">VirtualizingStackPanel</a>	Yes	Arranges and virtualizes content on a single line oriented horizontally or vertically.
<a href="#">WrapPanel</a>	Yes	<a href="#">WrapPanel</a> positions child elements in sequential position from left to right, breaking content to the next line at the edge of the containing box. Subsequent ordering happens sequentially from top to bottom or right to left, depending on the value of the <a href="#">Orientation</a> property.

## User Interface Panels

There are six panel classes available in WPF that are optimized to support UI scenarios: [Canvas](#), [DockPanel](#), [Grid](#), [StackPanel](#), [VirtualizingStackPanel](#), and [WrapPanel](#). These panel elements are easy to use, versatile, and extensible enough for most applications.

Each derived [Panel](#) element treats sizing constraints differently. Understanding how a [Panel](#) handles constraints in either the horizontal or vertical direction can make layout more predictable.

PANEL NAME	X-DIMENSION	Y-DIMENSION
<a href="#">Canvas</a>	Constrained to content	Constrained to content
<a href="#">DockPanel</a>	Constrained	Constrained
<a href="#">StackPanel</a> (Vertical Orientation)	Constrained	Constrained to content
<a href="#">StackPanel</a> (Horizontal Orientation)	Constrained to content	Constrained
<a href="#">Grid</a>	Constrained	Constrained, except in cases where <a href="#">Auto</a> apply to rows and columns
<a href="#">WrapPanel</a>	Constrained to content	Constrained to content

More detailed descriptions and usage examples of each of these elements can be found below.

### [Canvas](#)

The [Canvas](#) element enables positioning of content according to absolute x- and y- coordinates. Elements can be drawn in a unique location; or, if elements occupy the same coordinates, the order in which they appear in markup determines the order in which the elements are drawn.

[Canvas](#) provides the most flexible layout support of any [Panel](#). Height and Width properties are used to define the area of the canvas, and elements inside are assigned absolute coordinates relative to the area of the parent [Canvas](#). Four attached properties, [Canvas.Left](#), [Canvas.Top](#), [Canvas.Right](#) and [Canvas.Bottom](#), allow fine control of object placement within a [Canvas](#), allowing the developer to position and arrange elements precisely on the screen.

#### **ClipToBounds Within a Canvas**

[Canvas](#) can position child elements at any position on the screen, even at coordinates that are outside of its own defined [Height](#) and [Width](#). Furthermore, [Canvas](#) is not affected by the size of its children. As a result, it is possible for a child element to overdraw other elements outside the bounding rectangle of the parent [Canvas](#).

The default behavior of a [Canvas](#) is to allow children to be drawn outside the bounds of the parent [Canvas](#). If this behavior is undesirable, the [ClipToBounds](#) property can be set to `true`. This causes [Canvas](#) to clip to its own size. [Canvas](#) is the only layout element that allows children to be drawn outside its bounds.

This behavior is graphically illustrated in the [Width Properties Comparison Sample](#).

#### Defining and Using a Canvas

A [Canvas](#) can be instantiated simply by using Extensible Application Markup Language (XAML) or code. The following example demonstrates how to use [Canvas](#) to absolutely position content. This code produces three 100-pixel squares. The first square is red, and its top-left (*x*, *y*) position is specified as (0, 0). The second square is green, and its top-left position is (100, 100), just below and to the right of the first square. The third square is blue, and its top-left position is (50, 50), thus encompassing the lower-right quadrant of the first square and the upper-left quadrant of the second. Because the third square is laid out last, it appears to be on top of the other two squares—that is, the overlapping portions assume the color of the third box.

```
// Create the application's main window
mainWindow = new Window();
mainWindow.Title = "Canvas Sample";

// Create the Canvas
myParentCanvas = new Canvas();
myParentCanvas.Width = 400;
myParentCanvas.Height = 400;

// Define child Canvas elements
myCanvas1 = new Canvas();
myCanvas1.Background = Brushes.Red;
myCanvas1.Height = 100;
myCanvas1.Width = 100;
Canvas.SetTop(myCanvas1, 0);
Canvas.SetLeft(myCanvas1, 0);

myCanvas2 = new Canvas();
myCanvas2.Background = Brushes.Green;
myCanvas2.Height = 100;
myCanvas2.Width = 100;
Canvas.SetTop(myCanvas2, 100);
Canvas.SetLeft(myCanvas2, 100);

myCanvas3 = new Canvas();
myCanvas3.Background = Brushes.Blue;
myCanvas3.Height = 100;
myCanvas3.Width = 100;
Canvas.SetTop(myCanvas3, 50);
Canvas.SetLeft(myCanvas3, 50);

// Add child elements to the Canvas' Children collection
myParentCanvas.Children.Add(myCanvas1);
myParentCanvas.Children.Add(myCanvas2);
myParentCanvas.Children.Add(myCanvas3);

// Add the parent Canvas as the Content of the Window Object
mainWindow.Content = myParentCanvas;
mainWindow.Show();
```

```

WindowTitle = "Canvas Sample"
'Create a Canvas as the root Panel
Dim myParentCanvas As New Canvas()
myParentCanvas.Width = 400
myParentCanvas.Height = 400

' Define child Canvas elements
Dim myCanvas1 As New Canvas()
myCanvas1.Background = Brushes.Red
myCanvas1.Height = 100
myCanvas1.Width = 100
Canvas.SetTop(myCanvas1, 0)
Canvas.SetLeft(myCanvas1, 0)

Dim myCanvas2 As New Canvas()
myCanvas2.Background = Brushes.Green
myCanvas2.Height = 100
myCanvas2.Width = 100
Canvas.SetTop(myCanvas2, 100)
Canvas.SetLeft(myCanvas2, 100)

Dim myCanvas3 As New Canvas()
myCanvas3.Background = Brushes.Blue
myCanvas3.Height = 100
myCanvas3.Width = 100
Canvas.SetTop(myCanvas3, 50)
Canvas.SetLeft(myCanvas3, 50)

' Add child elements to the Canvas' Children collection
myParentCanvas.Children.Add(myCanvas1)
myParentCanvas.Children.Add(myCanvas2)
myParentCanvas.Children.Add(myCanvas3)

' Add the parent Canvas as the Content of the Window Object
Me.Content = myParentCanvas

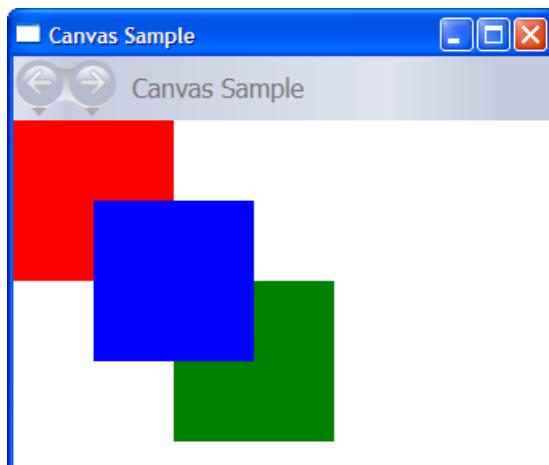
```

```

<Page WindowTitle="Canvas Sample" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
    <Canvas Height="400" Width="400">
        <Canvas Height="100" Width="100" Top="0" Left="0" Background="Red"/>
        <Canvas Height="100" Width="100" Top="100" Left="100" Background="Green"/>
        <Canvas Height="100" Width="100" Top="50" Left="50" Background="Blue"/>
    </Canvas>
</Page>

```

The compiled application yields a new UI that looks like this.



## DockPanel

The [DockPanel](#) element uses the [DockPanel.Dock](#) attached property as set in child content elements to

position content along the edges of a container. When [DockPanel.Dock](#) is set to [Top](#) or [Bottom](#), it positions child elements above or below each other. When [DockPanel.Dock](#) is set to [Left](#) or [Right](#), it positions child elements to the left or right of each other. The [LastChildFill](#) property determines the position of the final element added as a child of a [DockPanel](#).

You can use [DockPanel](#) to position a group of related controls, such as a set of buttons. Alternately, you can use it to create a "paned" UI, similar to that found in Microsoft Outlook.

#### Sizing to Content

If its [Height](#) and [Width](#) properties are not specified, [DockPanel](#) sizes to its content. The size can increase or decrease to accommodate the size of its child elements. However, when these properties are specified and there is no longer room for the next specified child element, [DockPanel](#) does not display that child element or subsequent child elements and does not measure subsequent child elements.

#### LastChildFill

By default, the last child of a [DockPanel](#) element will "fill" the remaining, unallocated space. If this behavior is not desired, set the [LastChildFill](#) property to `false`.

#### Defining and Using a DockPanel

The following example demonstrates how to partition space using a [DockPanel](#). Five [Border](#) elements are added as children of a parent [DockPanel](#). Each uses a different positioning property of a [DockPanel](#) to partition space. The final element "fills" the remaining, unallocated space.

```
// Create the application's main window
mainWindow = gcnew Window();
mainWindow->Title = "DockPanel Sample";

// Create the DockPanel
DockPanel^ myDockPanel = gcnew DockPanel();
myDockPanel->LastChildFill = true;

// Define the child content
Border^ myBorder1 = gcnew Border();
myBorder1->Height = 25;
myBorder1->Background = Brushes::SkyBlue;
myBorder1->BorderBrush = Brushes::Black;
myBorder1->BorderThickness = Thickness(1);
DockPanel::SetDock(myBorder1, Dock::Top);
TextBlock^ myTextBlock1 = gcnew TextBlock();
myTextBlock1->Foreground = Brushes::Black;
myTextBlock1->Text = "Dock = Top";
myBorder1->Child = myTextBlock1;

Border^ myBorder2 = gcnew Border();
myBorder2->Height = 25;
myBorder2->Background = Brushes::SkyBlue;
myBorder2->BorderBrush = Brushes::Black;
myBorder2->BorderThickness = Thickness(1);
DockPanel::SetDock(myBorder2, Dock::Top);
TextBlock^ myTextBlock2 = gcnew TextBlock();
myTextBlock2->Foreground = Brushes::Black;
myTextBlock2->Text = "Dock = Top";
myBorder2->Child = myTextBlock2;

Border^ myBorder3 = gcnew Border();
myBorder3->Height = 25;
myBorder3->Background = Brushes::LemonChiffon;
myBorder3->BorderBrush = Brushes::Black;
myBorder3->BorderThickness = Thickness(1);
DockPanel::SetDock(myBorder3, Dock::Bottom);
TextBlock^ myTextBlock3 = gcnew TextBlock();
myTextBlock3->Foreground = Brushes::Black;
myTextBlock3->Text = "Dock = Bottom";
```

```

myBorder3->Child = myTextBlock3;

Border^ myBorder4 = gcnew Border();
myBorder4->Width = 200;
myBorder4->Background = Brushes::PaleGreen;
myBorder4->BorderBrush = Brushes::Black;
myBorder4->BorderThickness = Thickness(1);
DockPanel::SetDock(myBorder4, Dock::Left);
TextBlock^ myTextBlock4 = gcnew TextBlock();
myTextBlock4->Foreground = Brushes::Black;
myTextBlock4->Text = "Dock = Left";
myBorder4->Child = myTextBlock4;

Border^ myBorder5 = gcnew Border();
myBorder5->Background = Brushes::White;
myBorder5->BorderBrush = Brushes::Black;
myBorder5->BorderThickness = Thickness(1);
TextBlock^ myTextBlock5 = gcnew TextBlock();
myTextBlock5->Foreground = Brushes::Black;
myTextBlock5->Text = "This content will Fill the remaining space";
myBorder5->Child = myTextBlock5;

// Add child elements to the DockPanel Children collection
myDockPanel->Children->Add(myBorder1);
myDockPanel->Children->Add(myBorder2);
myDockPanel->Children->Add(myBorder3);
myDockPanel->Children->Add(myBorder4);
myDockPanel->Children->Add(myBorder5);

// Add the parent Canvas as the Content of the Window Object
mainWindow->Content = myDockPanel;
mainWindow->Show();

```

```

// Create the application's main window
mainWindow = new Window ();
mainWindow.Title = "DockPanel Sample";

// Create the DockPanel
DockPanel myDockPanel = new DockPanel();
myDockPanel.LastChildFill = true;

// Define the child content
Border myBorder1 = new Border();
myBorder1.Height = 25;
myBorder1.Background = Brushes.SkyBlue;
myBorder1.BorderBrush = Brushes.Black;
myBorder1.BorderThickness = new Thickness(1);
DockPanel.SetDock(myBorder1, Dock.Top);
TextBlock myTextBlock1 = new TextBlock();
myTextBlock1.Foreground = Brushes.Black;
myTextBlock1.Text = "Dock = Top";
myBorder1.Child = myTextBlock1;

Border myBorder2 = new Border();
myBorder2.Height = 25;
myBorder2.Background = Brushes.SkyBlue;
myBorder2.BorderBrush = Brushes.Black;
myBorder2.BorderThickness = new Thickness(1);
DockPanel.SetDock(myBorder2, Dock.Top);
TextBlock myTextBlock2 = new TextBlock();
myTextBlock2.Foreground = Brushes.Black;
myTextBlock2.Text = "Dock = Top";
myBorder2.Child = myTextBlock2;

Border myBorder3 = new Border();

```

```
myBorder3.Height = 25;
myBorder3.Background = Brushes.LemonChiffon;
myBorder3.BorderBrush = Brushes.Black;
myBorder3.BorderThickness = new Thickness(1);
DockPanel.SetDock(myBorder3, Dock.Bottom);
TextBlock myTextBlock3 = new TextBlock();
myTextBlock3.Foreground = Brushes.Black;
myTextBlock3.Text = "Dock = Bottom";
myBorder3.Child = myTextBlock3;

Border myBorder4 = new Border();
myBorder4.Width = 200;
myBorder4.Background = Brushes.PaleGreen;
myBorder4.BorderBrush = Brushes.Black;
myBorder4.BorderThickness = new Thickness(1);
DockPanel.SetDock(myBorder4, Dock.Left);
TextBlock myTextBlock4 = new TextBlock();
myTextBlock4.Foreground = Brushes.Black;
myTextBlock4.Text = "Dock = Left";
myBorder4.Child = myTextBlock4;

Border myBorder5 = new Border();
myBorder5.Background = Brushes.White;
myBorder5.BorderBrush = Brushes.Black;
myBorder5.BorderThickness = new Thickness(1);
TextBlock myTextBlock5 = new TextBlock();
myTextBlock5.Foreground = Brushes.Black;
myTextBlock5.Text = "This content will Fill the remaining space";
myBorder5.Child = myTextBlock5;

// Add child elements to the DockPanel Children collection
myDockPanel.Children.Add(myBorder1);
myDockPanel.Children.Add(myBorder2);
myDockPanel.Children.Add(myBorder3);
myDockPanel.Children.Add(myBorder4);
myDockPanel.Children.Add(myBorder5);

// Add the parent Canvas as the Content of the Window Object
mainWindow.Content = myDockPanel;
mainWindow.Show();
```

```

WindowTitle = "DockPanel Sample"
'Create a DockPanel as the root Panel
Dim myDockPanel As New DockPanel()
myDockPanel.LastChildFill = True

' Define the child content
Dim myBorder1 As New Border()
myBorder1.Height = 25
myBorder1.Background = Brushes.SkyBlue
myBorder1.BorderBrush = Brushes.Black
myBorder1.BorderThickness = New Thickness(1)
DockPanel.SetDock(myBorder1, Dock.Top)
Dim myTextBlock1 As New TextBlock()
myTextBlock1.Foreground = Brushes.Black
myTextBlock1.Text = "Dock = Top"
myBorder1.Child = myTextBlock1

Dim myBorder2 As New Border()
myBorder2.Height = 25
myBorder2.Background = Brushes.SkyBlue
myBorder2.BorderBrush = Brushes.Black
myBorder2.BorderThickness = New Thickness(1)
DockPanel.SetDock(myBorder2, Dock.Top)
Dim myTextBlock2 As New TextBlock()
myTextBlock2.Foreground = Brushes.Black
myTextBlock2.Text = "Dock = Top"
myBorder2.Child = myTextBlock2

Dim myBorder3 As New Border()
myBorder3.Height = 25
myBorder3.Background = Brushes.LemonChiffon
myBorder3.BorderBrush = Brushes.Black
myBorder3.BorderThickness = New Thickness(1)
DockPanel.SetDock(myBorder3, Dock.Bottom)
Dim myTextBlock3 As New TextBlock()
myTextBlock3.Foreground = Brushes.Black
myTextBlock3.Text = "Dock = Bottom"
myBorder3.Child = myTextBlock3

Dim myBorder4 As New Border()
myBorder4.Width = 200
myBorder4.Background = Brushes.PaleGreen
myBorder4.BorderBrush = Brushes.Black
myBorder4.BorderThickness = New Thickness(1)
DockPanel.SetDock(myBorder4, Dock.Left)
Dim myTextBlock4 As New TextBlock()
myTextBlock4.Foreground = Brushes.Black
myTextBlock4.Text = "Dock = Left"
myBorder4.Child = myTextBlock4

Dim myBorder5 As New Border()
myBorder5.Background = Brushes.White
myBorder5.BorderBrush = Brushes.Black
myBorder5.BorderThickness = New Thickness(1)
Dim myTextBlock5 As New TextBlock()
myTextBlock5.Foreground = Brushes.Black
myTextBlock5.Text = "This content will Fill the remaining space"
myBorder5.Child = myTextBlock5

' Add child elements to the DockPanel Children collection
myDockPanel.Children.Add(myBorder1)
myDockPanel.Children.Add(myBorder2)
myDockPanel.Children.Add(myBorder3)
myDockPanel.Children.Add(myBorder4)
myDockPanel.Children.Add(myBorder5)
Me.Content = myDockPanel

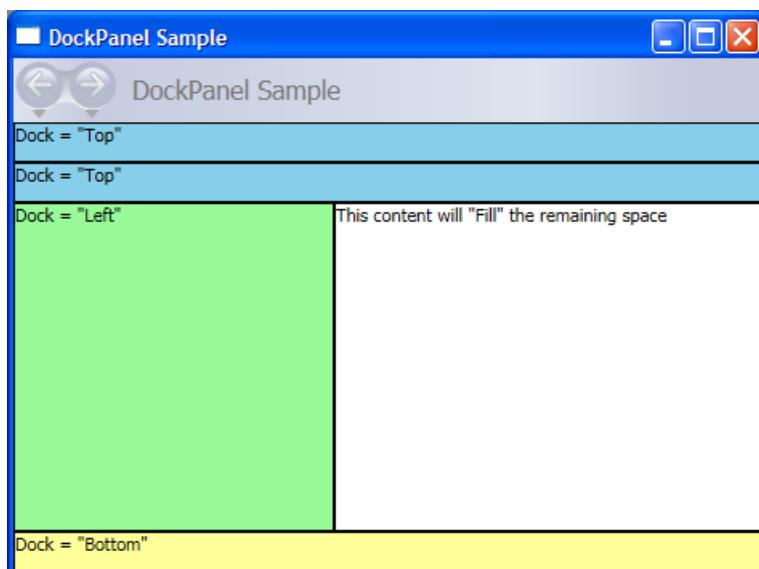
```

```

<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" WindowTitle="DockPanel Sample">
    <DockPanel LastChildFill="True">
        <Border Height="25" Background="SkyBlue" BorderBrush="Black" BorderThickness="1" DockPanel.Dock="Top">
            <TextBlock Foreground="Black">Dock = "Top"</TextBlock>
        </Border>
        <Border Height="25" Background="SkyBlue" BorderBrush="Black" BorderThickness="1" DockPanel.Dock="Top">
            <TextBlock Foreground="Black">Dock = "Top"</TextBlock>
        </Border>
        <Border Height="25" Background="LemonChiffon" BorderBrush="Black" BorderThickness="1" DockPanel.Dock="Bottom">
            <TextBlock Foreground="Black">Dock = "Bottom"</TextBlock>
        </Border>
        <Border Width="200" Background="PaleGreen" BorderBrush="Black" BorderThickness="1" DockPanel.Dock="Left">
            <TextBlock Foreground="Black">Dock = "Left"</TextBlock>
        </Border>
        <Border Background="White" BorderBrush="Black" BorderThickness="1">
            <TextBlock Foreground="Black">This content will "Fill" the remaining space</TextBlock>
        </Border>
    </DockPanel>
</Page>

```

The compiled application yields a new UI that looks like this.



## Grid

The [Grid](#) element merges the functionality of an absolute positioning and tabular data control. A [Grid](#) enables you to easily position and style elements. [Grid](#) allows you to define flexible row and column groupings, and even provides a mechanism to share sizing information between multiple [Grid](#) elements.

### How is Grid Different from Table?

[Table](#) and [Grid](#) share some common functionality, but each is best suited for different scenarios. A [Table](#) is designed for use within flow content (see [Flow Document Overview](#) for more information on flow content). Grids are best used inside of forms (basically anywhere outside of flow content). Within a [FlowDocument](#), [Table](#) supports flow content behaviors like pagination, column reflow, and content selection while a [Grid](#) does not. A [Grid](#) on the other hand is best used outside of a [FlowDocument](#) for many reasons including [Grid](#) adds elements based on a row and column index, [Table](#) does not. The [Grid](#) element allows layering of child content, allowing more than one element to exist within a single "cell." [Table](#) does not support layering. Child elements of a [Grid](#) can be absolutely positioned relative to the area of their "cell" boundaries. [Table](#) does not support this feature. Finally, a [Grid](#) is lighter weight than a [Table](#).

### Sizing Behavior of Columns and Rows

Columns and rows defined within a [Grid](#) can take advantage of [Star](#) sizing in order to distribute remaining space proportionally. When [Star](#) is selected as the Height or Width of a row or column, that column or row receives a weighted proportion of remaining available space. This is in contrast to [Auto](#), which will distribute space evenly based on the size of the content within a column or row. This value is expressed as `*` or `2*` when using Extensible Application Markup Language (XAML). In the first case, the row or column would receive one times the available space, in the second case, two times, and so on. By combining this technique to proportionally distribute space with a [HorizontalAlignment](#) and [VerticalAlignment](#) value of [Stretch](#) it is possible to partition layout space by percentage of screen space. [Grid](#) is the only layout panel that can distribute space in this manner.

### Defining and Using a Grid

The following example demonstrates how to build a UI similar to that found on the Run dialog available on the Windows Start menu.

```
// Create the Grid.
grid1 = new Grid ();
grid1.Background = Brushes.Gainsboro;
grid1.HorizontalAlignment = HorizontalAlignment.Left;
grid1.VerticalAlignment = VerticalAlignment.Top;
grid1.ShowGridLines = true;
grid1.Width = 425;
grid1.Height = 165;

// Define the Columns.
colDef1 = new ColumnDefinition();
colDef1.Width = new GridLength(1, GridUnitType.Auto);
colDef2 = new ColumnDefinition();
colDef2.Width = new GridLength(1, GridUnitType.Star);
colDef3 = new ColumnDefinition();
colDef3.Width = new GridLength(1, GridUnitType.Star);
colDef4 = new ColumnDefinition();
colDef4.Width = new GridLength(1, GridUnitType.Star);
colDef5 = new ColumnDefinition();
colDef5.Width = new GridLength(1, GridUnitType.Star);
grid1.ColumnDefinitions.Add(colDef1);
grid1.ColumnDefinitions.Add(colDef2);
grid1.ColumnDefinitions.Add(colDef3);
grid1.ColumnDefinitions.Add(colDef4);
grid1.ColumnDefinitions.Add(colDef5);

// Define the Rows.
rowDef1 = new RowDefinition();
rowDef1.Height = new GridLength(1, GridUnitType.Auto);
rowDef2 = new RowDefinition();
rowDef2.Height = new GridLength(1, GridUnitType.Auto);
rowDef3 = new RowDefinition();
rowDef3.Height = new GridLength(1, GridUnitType.Star);
rowDef4 = new RowDefinition();
rowDef4.Height = new GridLength(1, GridUnitType.Auto);
grid1.RowDefinitions.Add(rowDef1);
grid1.RowDefinitions.Add(rowDef2);
grid1.RowDefinitions.Add(rowDef3);
grid1.RowDefinitions.Add(rowDef4);

// Add the Image.
img1 = new Image();
img1.Source = new System.Windows.Media.Imaging.BitmapImage(new Uri("runicon.png", UriKind.Relative));
Grid.SetRow(img1, 0);
Grid.SetColumn(img1, 0);

// Add the main application dialog.
txt1 = new TextBlock();
txt1.Text = "Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.";
```

```

    for you. ,
txt1.TextWrapping = TextWrapping.Wrap;
Grid.SetColumnSpan(txt1, 4);
Grid.SetRow(txt1, 0);
Grid.SetColumn(txt1, 1);

// Add the second text cell to the Grid.
txt2 = new TextBlock();
txt2.Text = "Open:" ;
Grid.SetRow(txt2, 1);
Grid.SetColumn(txt2, 0);

// Add the TextBox control.
tb1 = new TextBox();
Grid.SetRow(tb1, 1);
Grid.SetColumn(tb1, 1);
Grid.SetColumnSpan(tb1, 5);

// Add the buttons.
button1 = new Button();
button2 = new Button();
button3 = new Button();
button1.Content = "OK";
button2.Content = "Cancel";
button3.Content = "Browse ...";
Grid.SetRow(button1, 3);
Grid.SetColumn(button1, 2);
button1.Margin = new Thickness(10, 0, 10, 15);
button2.Margin = new Thickness(10, 0, 10, 15);
button3.Margin = new Thickness(10, 0, 10, 15);
Grid.SetRow(button2, 3);
Grid.SetColumn(button2, 3);
Grid.SetRow(button3, 3);
Grid.SetColumn(button3, 4);

grid1.Children.Add(img1);
grid1.Children.Add(txt1);
grid1.Children.Add(txt2);
grid1.Children.Add(tb1);
grid1.Children.Add(button1);
grid1.Children.Add(button2);
grid1.Children.Add(button3);

mainWindow.Content = grid1;

```

```

'Create a Grid as the root Panel element.
Dim myGrid As New Grid()
myGrid.Height = 165
myGrid.Width = 425
myGrid.Background = Brushes.Gainsboro
myGrid.ShowGridLines = True
myGrid.HorizontalAlignment = Windows.HorizontalAlignment.Left
myGrid.VerticalAlignment = Windows.VerticalAlignment.Top

' Define and Add the Rows and Columns.
Dim colDef1 As New ColumnDefinition
colDef1.Width = New GridLength(1, GridUnitType.Auto)
Dim colDef2 As New ColumnDefinition
colDef2.Width = New GridLength(1, GridUnitType.Star)
Dim colDef3 As New ColumnDefinition
colDef3.Width = New GridLength(1, GridUnitType.Star)
Dim colDef4 As New ColumnDefinition
colDef4.Width = New GridLength(1, GridUnitType.Star)
Dim colDef5 As New ColumnDefinition
colDef5.Width = New GridLength(1, GridUnitType.Star)
myGrid.ColumnDefinitions.Add(colDef1)
myGrid.ColumnDefinitions.Add(colDef2)

```

```
...,
myGrid.ColumnDefinitions.Add(colDef1)
myGrid.ColumnDefinitions.Add(colDef2)
myGrid.ColumnDefinitions.Add(colDef3)
myGrid.ColumnDefinitions.Add(colDef4)
myGrid.ColumnDefinitions.Add(colDef5)

Dim rowDef1 As New RowDefinition
rowDef1.Height = New GridLength(1, GridUnitType.Auto)
Dim rowDef2 As New RowDefinition
rowDef2.Height = New GridLength(1, GridUnitType.Auto)
Dim rowDef3 As New Controls.RowDefinition
rowDef3.Height = New GridLength(1, GridUnitType.Star)
Dim rowDef4 As New RowDefinition
rowDef4.Height = New GridLength(1, GridUnitType.Auto)
myGrid.RowDefinitions.Add(rowDef1)
myGrid.RowDefinitions.Add(rowDef2)
myGrid.RowDefinitions.Add(rowDef3)
myGrid.RowDefinitions.Add(rowDef4)

' Add the Image.
Dim img1 As New Image
img1.Source = New System.Windows.Media.Imaging.BitmapImage(New Uri("runicon.png", UriKind.Relative))
Grid.SetRow(img1, 0)
Grid.SetColumn(img1, 0)
myGrid.Children.Add(img1)

' Add the main application dialog.
Dim txt1 As New TextBlock
txt1.Text = "Type the name of a program, document, or Internet resource, and Windows will open it for you."
txt1.TextWrapping = TextWrapping.Wrap
Grid.SetColumnSpan(txt1, 4)
Grid.SetRow(txt1, 0)
Grid.SetColumn(txt1, 1)
myGrid.Children.Add(txt1)

' Add the second TextBlock Cell to the Grid.
Dim txt2 As New TextBlock
txt2.Text = "Open:"
Grid.SetRow(txt2, 1)
Grid.SetColumn(txt2, 0)
myGrid.Children.Add(txt2)

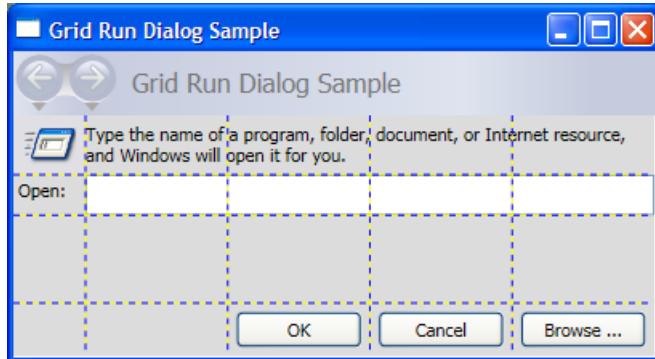
' Add the TextBox control.
Dim tb1 As New TextBox
Grid.SetRow(tb1, 1)
Grid.SetColumn(tb1, 1)
Grid.SetColumnSpan(tb1, 5)
myGrid.Children.Add(tb1)

' Add the Button controls.
Dim button1 As New Button
Dim button2 As New Button
Dim button3 As New Button
button1.Content = "OK"
button1.Margin = New Thickness(10, 0, 10, 15)
button2.Content = "Cancel"
button2.Margin = New Thickness(10, 0, 10, 15)
button3.Content = "Browse ..."
button3.Margin = New Thickness(10, 0, 10, 15)

Grid.SetRow(button1, 3)
Grid.SetColumn(button1, 2)
Grid.SetRow(button2, 3)
Grid.SetColumn(button2, 3)
Grid.SetRow(button3, 3)
Grid.SetColumn(button3, 4)
myGrid.Children.Add(button1)
myGrid.Children.Add(button2)
myGrid.Children.Add(button3)
```

```
Me.Content = myGrid
```

The compiled application yields a new UI that looks like this.



## StackPanel

A [StackPanel](#) enables you to "stack" elements in an assigned direction. The default stack direction is vertical. The [Orientation](#) property can be used to control content flow.

### StackPanel vs. DockPanel

Although [DockPanel](#) can also "stack" child elements, [DockPanel](#) and [StackPanel](#) do not produce analogous results in some usage scenarios. For example, the order of child elements can affect their size in a [DockPanel](#) but not in a [StackPanel](#). This is because [StackPanel](#) measures in the direction of stacking at [PositiveInfinity](#), whereas [DockPanel](#) measures only the available size.

The following example demonstrates this key difference.

```
// Create the application's main window
mainWindow = gcnew Window();
mainWindow->Title = "StackPanel vs. DockPanel";

// Add root Grid
myGrid = gcnew Grid();
myGrid->Width = 175;
myGrid->Height = 150;
RowDefinition^ myRowDef1 = gcnew RowDefinition();
RowDefinition^ myRowDef2 = gcnew RowDefinition();
myGrid->RowDefinitions->Add(myRowDef1);
myGrid->RowDefinitions->Add(myRowDef2);

// Define the DockPanel
myDockPanel = gcnew DockPanel();
Grid::SetRow(myDockPanel, 0);

//Define an Image and Source
Image^ myImage = gcnew Image();
BitmapImage^ bi = gcnew BitmapImage();
bi->BeginInit();
bi->UriSource = gcnew System::Uri("smiley_stackpanel.png", UriKind::Relative);
bi->EndInit();
myImage->Source = bi;

Image^ myImage2 = gcnew Image();
BitmapImage^ bi2 = gcnew BitmapImage();
bi2->BeginInit();
bi2->UriSource = gcnew System::Uri("smiley_stackpanel.png", UriKind::Relative);
bi2->EndInit();
myImage2->Source = bi2;

Image^ myImage3 = gcnew Image();
BitmapImage^ bi3 = gcnew BitmapImage();
bi3->BeginInit();
```

```

bi3->BeginInit();
bi3->UriSource = gcnew System::Uri("smiley_stackpanel.PNG", UriKind::Relative);
bi3->EndInit();
myImage3->Stretch = Stretch::Fill;
myImage3->Source = bi3;

// Add the images to the parent DockPanel
myDockPanel->Children->Add(myImage);
myDockPanel->Children->Add(myImage2);
myDockPanel->Children->Add(myImage3);

//Define a StackPanel
myStackPanel = gcnew StackPanel();
myStackPanel->Orientation = Orientation::Horizontal;
Grid::SetRow(myStackPanel, 1);

Image^ myImage4 = gcnew Image();
BitmapImage^ bi4 = gcnew BitmapImage();
bi4->BeginInit();
bi4->UriSource = gcnew System::Uri("smiley_stackpanel.png", UriKind::Relative);
bi4->EndInit();
myImage4->Source = bi4;

Image^ myImage5 = gcnew Image();
BitmapImage^ bi5 = gcnew BitmapImage();
bi5->BeginInit();
bi5->UriSource = gcnew System::Uri("smiley_stackpanel.png", UriKind::Relative);
bi5->EndInit();
myImage5->Source = bi5;

Image^ myImage6 = gcnew Image();
BitmapImage^ bi6 = gcnew BitmapImage();
bi6->BeginInit();
bi6->UriSource = gcnew System::Uri("smiley_stackpanel.PNG", UriKind::Relative);
bi6->EndInit();
myImage6->Stretch = Stretch::Fill;
myImage6->Source = bi6;

// Add the images to the parent StackPanel
myStackPanel->Children->Add(myImage4);
myStackPanel->Children->Add(myImage5);
myStackPanel->Children->Add(myImage6);

// Add the layout panels as children of the Grid
myGrid->Children->Add(myDockPanel);
myGrid->Children->Add(myStackPanel);

// Add the Grid as the Content of the Parent Window Object
mainWindow->Content = myGrid;
mainWindow->Show();

```

```

// Create the application's main window
mainWindow = new Window ();
mainWindow.Title = "StackPanel vs. DockPanel";

// Add root Grid
myGrid = new Grid();
myGrid.Width = 175;
myGrid.Height = 150;
RowDefinition myRowDef1 = new RowDefinition();
RowDefinition myRowDef2 = new RowDefinition();
myGrid.RowDefinitions.Add(myRowDef1);
myGrid.RowDefinitions.Add(myRowDef2);

// Define the DockPanel
myDockPanel = new DockPanel();

```

```

myDockPanel = new DockPanel();
Grid.SetRow(myDockPanel, 0);

//Define an Image and Source
Image myImage = new Image();
BitmapImage bi = new BitmapImage();
bi.BeginInit();
bi.UriSource = new Uri("smiley_stackpanel.png", UriKind.Relative);
bi.EndInit();
myImage.Source = bi;

Image myImage2 = new Image();
BitmapImage bi2 = new BitmapImage();
bi2.BeginInit();
bi2.UriSource = new Uri("smiley_stackpanel.png", UriKind.Relative);
bi2.EndInit();
myImage2.Source = bi2;

Image myImage3 = new Image();
BitmapImage bi3 = new BitmapImage();
bi3.BeginInit();
bi3.UriSource = new Uri("smiley_stackpanel.PNG", UriKind.Relative);
bi3.EndInit();
myImage3.Stretch = Stretch.Fill;
myImage3.Source = bi3;

// Add the images to the parent DockPanel
myDockPanel.Children.Add(myImage);
myDockPanel.Children.Add(myImage2);
myDockPanel.Children.Add(myImage3);

//Define a StackPanel
myStackPanel = new StackPanel();
myStackPanel.Orientation = Orientation.Horizontal;
Grid.SetRow(myStackPanel, 1);

Image myImage4 = new Image();
BitmapImage bi4 = new BitmapImage();
bi4.BeginInit();
bi4.UriSource = new Uri("smiley_stackpanel.png", UriKind.Relative);
bi4.EndInit();
myImage4.Source = bi4;

Image myImage5 = new Image();
BitmapImage bi5 = new BitmapImage();
bi5.BeginInit();
bi5.UriSource = new Uri("smiley_stackpanel.png", UriKind.Relative);
bi5.EndInit();
myImage5.Source = bi5;

Image myImage6 = new Image();
BitmapImage bi6 = new BitmapImage();
bi6.BeginInit();
bi6.UriSource = new Uri("smiley_stackpanel.PNG", UriKind.Relative);
bi6.EndInit();
myImage6.Stretch = Stretch.Fill;
myImage6.Source = bi6;

// Add the images to the parent StackPanel
myStackPanel.Children.Add(myImage4);
myStackPanel.Children.Add(myImage5);
myStackPanel.Children.Add(myImage6);

// Add the layout panels as children of the Grid
myGrid.Children.Add(myDockPanel);
myGrid.Children.Add(myStackPanel);

// Add the Grid as the Content of the Parent Window Object
mainWindow.Content = myGrid;
mainWindow.Show();

```

```
    Window.Show();
```

```
'Add root Grid
Dim myGrid As New Grid
myGrid.Width = 175
myGrid.Height = 150
Dim myRowDef1 As New RowDefinition
Dim myRowDef2 As New RowDefinition
myGrid.RowDefinitions.Add(myRowDef1)
myGrid.RowDefinitions.Add(myRowDef2)

'Define the DockPanel
Dim myDockPanel As New DockPanel
Grid.SetRow(myDockPanel, 0)

'Define an Image and Source.
Dim myImage As New Image
Dim bi As New BitmapImage
bi.BeginInit()
bi.UriSource = New Uri("smiley_stackpanel.png", UriKind.Relative)
bi.EndInit()
myImage.Source = bi

Dim myImage2 As New Image
Dim bi2 As New BitmapImage
bi2.BeginInit()
bi2.UriSource = New Uri("smiley_stackpanel.png", UriKind.Relative)
bi2.EndInit()
myImage2.Source = bi2

Dim myImage3 As New Image
Dim bi3 As New BitmapImage
bi3.BeginInit()
bi3.UriSource = New Uri("smiley_stackpanel.PNG", UriKind.Relative)
bi3.EndInit()
myImage3.Stretch = Stretch.Fill
myImage3.Source = bi3

'Add the images to the parent DockPanel.
myDockPanel.Children.Add(myImage)
myDockPanel.Children.Add(myImage2)
myDockPanel.Children.Add(myImage3)

'Define a StackPanel.
Dim myStackPanel As New StackPanel
myStackPanel.Orientation = Orientation.Horizontal
Grid.SetRow(myStackPanel, 1)

Dim myImage4 As New Image
Dim bi4 As New BitmapImage
bi4.BeginInit()
bi4.UriSource = New Uri("smiley_stackpanel.png", UriKind.Relative)
bi4.EndInit()
myImage4.Source = bi4

Dim myImage5 As New Image
Dim bi5 As New BitmapImage
bi5.BeginInit()
bi5.UriSource = New Uri("smiley_stackpanel.png", UriKind.Relative)
bi5.EndInit()
myImage5.Source = bi5

Dim myImage6 As New Image
Dim bi6 As New BitmapImage
bi6.BeginInit()
bi6.UriSource = New Uri("smiley_stackpanel.PNG", UriKind.Relative)
```

```

        bi6.Source = New Uri("smiley_stackpanel.png", UriKind.Relative)
bi6.EndInit()
myImage6.Stretch = Stretch.Fill
myImage6.Source = bi6

'Add the images to the parent StackPanel.
myStackPanel.Children.Add(myImage4)
myStackPanel.Children.Add(myImage5)
myStackPanel.Children.Add(myImage6)

'Add the layout panels as children of the Grid
myGrid.Children.Add(myDockPanel)
myGrid.Children.Add(myStackPanel)

```

```

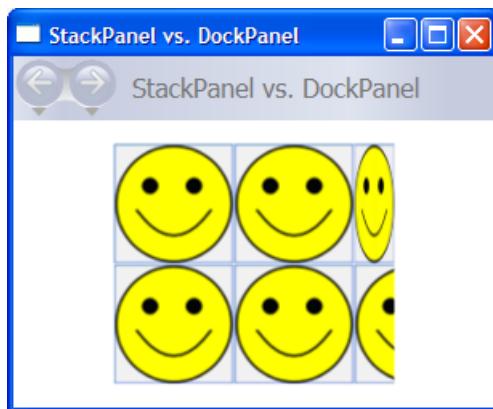
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      WindowTitle="StackPanel vs. DockPanel">
    <Grid Width="175" Height="150">
        <Grid.ColumnDefinitions>
            <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition />
        </Grid.RowDefinitions>

        <DockPanel Grid.Column="0" Grid.Row="0">
            <Image Source="smiley_stackpanel.png" />
            <Image Source="smiley_stackpanel.png" />
            <Image Source="smiley_stackpanel.png" Stretch="Fill"/>
        </DockPanel>

        <StackPanel Grid.Column="0" Grid.Row="1" Orientation="Horizontal">
            <Image Source="smiley_stackpanel.png" />
            <Image Source="smiley_stackpanel.png" />
            <Image Source="smiley_stackpanel.png" Stretch="Fill"/>
        </StackPanel>
    </Grid>
</Page>

```

The difference in rendering behavior can be seen in this image.



#### Defining and Using a StackPanel

The following example demonstrates how to use a [StackPanel](#) to create a set of vertically-positioned buttons. For horizontal positioning, set the [Orientation](#) property to [Horizontal](#).

```

// Create the application's main window
mainWindow = new Window ();
mainWindow.Title = "StackPanel Sample";

// Define the StackPanel
myStackPanel = new StackPanel();
myStackPanel.HorizontalAlignment = HorizontalAlignment.Left;
myStackPanel.VerticalAlignment = VerticalAlignment.Top;

// Define child content
Button myButton1 = new Button();
myButton1.Content = "Button 1";
Button myButton2 = new Button();
myButton2.Content = "Button 2";
Button myButton3 = new Button();
myButton3.Content = "Button 3";

// Add child elements to the parent StackPanel
myStackPanel.Children.Add(myButton1);
myStackPanel.Children.Add(myButton2);
myStackPanel.Children.Add(myButton3);

// Add the StackPanel as the Content of the Parent Window Object
mainWindow.Content = myStackPanel;
mainWindow.Show ();

```

```

WindowTitle = "StackPanel Sample"
' Define the StackPanel
Dim myStackPanel As New StackPanel()
myStackPanel.HorizontalAlignment = Windows.HorizontalAlignment.Left
myStackPanel.VerticalAlignment = Windows.VerticalAlignment.Top

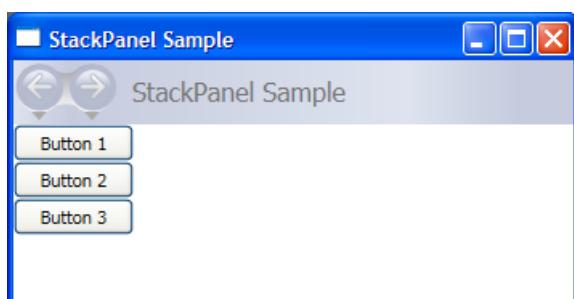
' Define child content
Dim myButton1 As New Button()
myButton1.Content = "Button 1"
Dim myButton2 As New Button()
myButton2.Content = "Button 2"
Dim myButton3 As New Button()
myButton3.Content = "Button 3"

' Add child elements to the parent StackPanel
myStackPanel.Children.Add(myButton1)
myStackPanel.Children.Add(myButton2)
myStackPanel.Children.Add(myButton3)

Me.Content = myStackPanel

```

The compiled application yields a new UI that looks like this.



#### VirtualizingStackPanel

WPF also provides a variation of the [StackPanel](#) element that automatically "virtualizes" data-bound child

content. In this context, the word virtualize refers to a technique by which a subset of elements are generated from a larger number of data items based upon which items are visible on-screen. It is intensive, both in terms of memory and processor, to generate a large number of UI elements when only a few may be on the screen at a given time. [VirtualizingStackPanel](#) (through functionality provided by [VirtualizingPanel](#)) calculates visible items and works with the [ItemContainerGenerator](#) from an [ItemsControl](#) (such as [ListBox](#) or [ListView](#)) to only create elements for visible items.

The [VirtualizingStackPanel](#) element is automatically set as the items host for controls such as the [ListBox](#). When hosting a data bound collection, content is automatically virtualized, as long as the content is within the bounds of a [ScrollViewer](#). This greatly improves performance when hosting many child items.

The following markup demonstrates how to use a [VirtualizingStackPanel](#) as an items host. The [VirtualizingStackPanel.IsVirtualizingProperty](#) attached property must be set to `true` (default) for virtualization to occur.

```
<StackPanel DataContext="{Binding Source={StaticResource Leagues}}">
    <TextBlock Text="{Binding XPath=@name}" FontFamily="Arial" FontSize="18" Foreground="Black"/>
    <ListBox VirtualizingStackPanel.IsVirtualizing="True"
        ItemsSource="{Binding XPath=Team}"
        ItemTemplate="{DynamicResource NameDataStyle}"/>
</StackPanel>
```

## WrapPanel

[WrapPanel](#) is used to position child elements in sequential position from left to right, breaking content to the next line when it reaches the edge of its parent container. Content can be oriented horizontally or vertically.

[WrapPanel](#) is useful for simple flowing user interface (UI) scenarios. It can also be used to apply uniform sizing to all of its child elements.

The following example demonstrates how to create a [WrapPanel](#) to display [Button](#) controls that wrap when they reach the edge of their container.

```
// Create the application's main window
mainWindow = gcnew System::Windows::Window();
mainWindow->Title = "WrapPanel Sample";

// Instantiate a new WrapPanel and set properties
myWrapPanel = gcnew WrapPanel();
myWrapPanel->Background = Brushes::Azure;
myWrapPanel->Orientation = Orientation::Horizontal;
myWrapPanel->ItemHeight = 25;

myWrapPanel->ItemWidth = 75;
myWrapPanel->Width = 150;
myWrapPanel->HorizontalAlignment = HorizontalAlignment::Left;
myWrapPanel->VerticalAlignment = VerticalAlignment::Top;

// Define 3 button elements. Each button is sized at width of 75, so the third button wraps to the next
line.
btn1 = gcnew Button();
btn1->Content = "Button 1";
btn2 = gcnew Button();
btn2->Content = "Button 2";
btn3 = gcnew Button();
btn3->Content = "Button 3";

// Add the buttons to the parent WrapPanel using the Children.Add method.
myWrapPanel->Children->Add(btn1);
myWrapPanel->Children->Add(btn2);
myWrapPanel->Children->Add(btn3);

// Add the WrapPanel to the MainWindow as Content
mainWindow->Content = myWrapPanel;
mainWindow->Show();
```

```
// Create the application's main window
mainWindow = new System.Windows.Window();
mainWindow.Title = "WrapPanel Sample";

// Instantiate a new WrapPanel and set properties
myWrapPanel = new WrapPanel();
myWrapPanel.Background = System.Windows.Media.Brushes.Azure;
myWrapPanel.Orientation = Orientation.Horizontal;
myWrapPanel.Width = 200;
myWrapPanel.HorizontalAlignment = HorizontalAlignment.Left;
myWrapPanel.VerticalAlignment = VerticalAlignment.Top;

// Define 3 button elements. The last three buttons are sized at width
// of 75, so the forth button wraps to the next line.
btn1 = new Button();
btn1.Content = "Button 1";
btn1.Width = 200;
btn2 = new Button();
btn2.Content = "Button 2";
btn2.Width = 75;
btn3 = new Button();
btn3.Content = "Button 3";
btn3.Width = 75;
btn4 = new Button();
btn4.Content = "Button 4";
btn4.Width = 75;

// Add the buttons to the parent WrapPanel using the Children.Add method.
myWrapPanel.Children.Add(btn1);
myWrapPanel.Children.Add(btn2);
myWrapPanel.Children.Add(btn3);
myWrapPanel.Children.Add(btn4);

// Add the WrapPanel to the MainWindow as Content
mainWindow.Content = myWrapPanel;
mainWindow.Show();
```

```

WindowTitle = "WrapPanel Sample"

' Instantiate a new WrapPanel and set properties
Dim myWrapPanel As New WrapPanel()
myWrapPanel.Background = Brushes.Azure
myWrapPanel.Orientation = Orientation.Horizontal

myWrapPanel.Width = 200
myWrapPanel.HorizontalAlignment = Windows.HorizontalAlignment.Left
myWrapPanel.VerticalAlignment = Windows.VerticalAlignment.Top

' Define 3 button elements. The last three buttons are sized at width
' of 75, so the forth button wraps to the next line.
Dim btn1 As New Button()
btn1.Content = "Button 1"
btn1.Width = 200
Dim btn2 As New Button()
btn2.Content = "Button 2"
btn2.Width = 75
Dim btn3 As New Button()
btn3.Content = "Button 3"
btn3.Width = 75
Dim btn4 As New Button()
btn4.Content = "Button 4"
btn4.Width = 75

' Add the buttons to the parent WrapPanel using the Children.Add method.
myWrapPanel.Children.Add(btn1)
myWrapPanel.Children.Add(btn2)
myWrapPanel.Children.Add(btn3)
myWrapPanel.Children.Add(btn4)

' Add the WrapPanel to the Page as Content
Me.Content = myWrapPanel

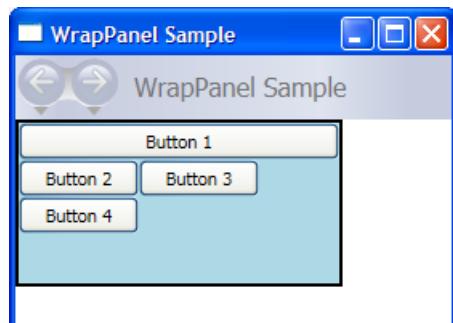
```

```

<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" WindowTitle="WrapPanel Sample">
    <Border HorizontalAlignment="Left" VerticalAlignment="Top" BorderBrush="Black" BorderThickness="2">
        <WrapPanel Background="LightBlue" Width="200" Height="100">
            <Button Width="200">Button 1</Button>
            <Button>Button 2</Button>
            <Button>Button 3</Button>
            <Button>Button 4</Button>
        </WrapPanel>
    </Border>
</Page>

```

The compiled application yields a new UI that looks like this.



## Nested Panel Elements

Panel elements can be nested within each other in order to produce complex layouts. This can prove very

useful in situations where one [Panel](#) is ideal for a portion of a UI, but may not meet the needs of a different portion of the UI.

There is no practical limit to the amount of nesting that your application can support, however, it is generally best to limit your application to only use those panels that are actually necessary for your desired layout. In many cases, a [Grid](#) element can be used instead of nested panels due to its flexibility as a layout container. This can increase performance in your application by keeping unnecessary elements out of the tree.

The following example demonstrates how to create a UI that takes advantage of nested [Panel](#) elements in order to achieve a specific layout. In this particular case, a [DockPanel](#) element is used to provide UI structure, and nested [StackPanel](#) elements, a [Grid](#), and a [Canvas](#) are used to position child elements precisely within the parent [DockPanel](#).

```
// Define the DockPanel.  
myDockPanel = new DockPanel();  
  
// Add the Left Docked StackPanel  
Border myBorder2 = new Border();  
myBorder2.BorderThickness = new Thickness(1);  
myBorder2.BorderBrush = Brushes.Black;  
DockPanel.SetDock(myBorder2, Dock.Left);  
StackPanel myStackPanel = new StackPanel();  
Button myButton1 = new Button();  
myButton1.Content = "Left Docked";  
myButton1.Margin = new Thickness(5);  
Button myButton2 = new Button();  
myButton2.Content = "StackPanel";  
myButton2.Margin = new Thickness(5);  
myStackPanel.Children.Add(myButton1);  
myStackPanel.Children.Add(myButton2);  
myBorder2.Child = myStackPanel;  
  
// Add the Top Docked Grid.  
Border myBorder3 = new Border();  
myBorder3.BorderThickness = new Thickness(1);  
myBorder3.BorderBrush = Brushes.Black;  
DockPanel.SetDock(myBorder3, Dock.Top);  
Grid myGrid = new Grid();  
myGrid.ShowGridLines = true;  
RowDefinition myRowDef1 = new RowDefinition();  
RowDefinition myRowDef2 = new RowDefinition();  
ColumnDefinition myColDef1 = new ColumnDefinition();  
ColumnDefinition myColDef2 = new ColumnDefinition();  
ColumnDefinition myColDef3 = new ColumnDefinition();  
myGrid.ColumnDefinitions.Add(myColDef1);  
myGrid.ColumnDefinitions.Add(myColDef2);  
myGrid.ColumnDefinitions.Add(myColDef3);  
myGrid.RowDefinitions.Add(myRowDef1);  
myGrid.RowDefinitions.Add(myRowDef2);  
TextBlock myTextBlock1 = new TextBlock();  
myTextBlock1.FontSize = 20;  
myTextBlock1.Margin = new Thickness(10);  
myTextBlock1.Text = "Grid Element Docked at the Top";  
Grid.SetRow(myTextBlock1, 0);  
Grid.SetColumnSpan(myTextBlock1, 3);  
Button myButton3 = new Button();  
myButton3.Margin = new Thickness(5);  
myButton3.Content = "A Row";  
Grid.SetColumn(myButton3, 0);  
Grid.SetRow(myButton3, 1);  
Button myButton4 = new Button();  
myButton4.Margin = new Thickness(5);  
myButton4.Content = "of Button";  
Grid.SetColumn(myButton4, 1);  
Grid.SetRow(myButton4, 1);  
Grid.SetColumn(myButton4, 1);
```

```

grid.SetRow(myButton4, 1);
Button myButton5 = new Button();
myButton5.Margin = new Thickness(5);
myButton5.Content = "Elements";
Grid.SetColumn(myButton5, 2);
Grid.SetRow(myButton5, 1);
myGrid.Children.Add(myTextBlock1);
myGrid.Children.Add(myButton3);
myGrid.Children.Add(myButton4);
myGrid.Children.Add(myButton5);
myBorder3.Child = myGrid;

// Add the Bottom Docked StackPanel.
Border myBorder4 = new Border();
myBorder4.BorderBrush = Brushes.Black;
myBorder4.BorderThickness = new Thickness(1);
DockPanel.SetDock(myBorder4, Dock.Bottom);
StackPanel myStackPanel2 = new StackPanel();
myStackPanel2.Orientation = Orientation.Horizontal;
TextBlock myTextBlock2 = new TextBlock();
myTextBlock2.Text = "This StackPanel is Docked to the Bottom";
myTextBlock2.Margin = new Thickness(5);
myStackPanel2.Children.Add(myTextBlock2);
myBorder4.Child = myStackPanel2;

// Add the Canvas, that fills remaining space.
Border myBorder5 = new Border();
myBorder4.BorderBrush = Brushes.Black;
myBorder5.BorderThickness = new Thickness(1);
Canvas myCanvas = new Canvas();
myCanvas.ClipToBounds = true;
TextBlock myTextBlock3 = new TextBlock();
myTextBlock3.Text = "Content in the Canvas will Fill the remaining space.";
Canvas.SetTop(myTextBlock3, 50);
Canvas.SetLeft(myTextBlock3, 50);
Ellipse myEllipse = new Ellipse();
myEllipse.Height = 100;
myEllipse.Width = 125;
myEllipse.Fill = Brushes.CornflowerBlue;
myEllipse.Stroke = Brushes.Aqua;
Canvas.SetTop(myEllipse, 100);
Canvas.SetLeft(myEllipse, 150);
myCanvas.Children.Add(myTextBlock3);
myCanvas.Children.Add(myEllipse);
myBorder5.Child = myCanvas;

// Add child elements to the parent DockPanel.
myDockPanel.Children.Add(myBorder2);
myDockPanel.Children.Add(myBorder3);
myDockPanel.Children.Add(myBorder4);
myDockPanel.Children.Add(myBorder5);

```

```

Dim myDockPanel As New DockPanel()

Dim myBorder2 As New Border()
myBorder2.BorderThickness = New Thickness(1)
myBorder2.BorderBrush = Brushes.Black
DockPanel.SetDock(myBorder2, Dock.Left)
Dim myStackPanel As New StackPanel()
Dim myButton1 As New Button()
myButton1.Content = "Left Docked"
myButton1.Margin = New Thickness(5)
Dim myButton2 As New Button()
myButton2.Content = "StackPanel"
myButton2.Margin = New Thickness(5)
myStackPanel.Children.Add(myButton1)
myStackPanel.Children.Add(myButton2)
myBorder2.Child = myStackPanel

```

```

Dim myBorder3 As New Border()
myBorder3.BorderThickness = New Thickness(1)
myBorder3.BorderBrush = Brushes.Black
DockPanel.SetDock(myBorder3, Dock.Top)
Dim myGrid As New Grid()
myGrid.ShowGridLines = True
Dim myRowDef1 As New RowDefinition()
Dim myRowDef2 As New RowDefinition()
Dim myColDef1 As New ColumnDefinition()
Dim myColDef2 As New ColumnDefinition()
Dim myColDef3 As New ColumnDefinition()
myGrid.ColumnDefinitions.Add(myColDef1)
myGrid.ColumnDefinitions.Add(myColDef2)
myGrid.ColumnDefinitions.Add(myColDef3)
myGrid.RowDefinitions.Add(myRowDef1)
myGrid.RowDefinitions.Add(myRowDef2)
Dim myTextBlock1 As New TextBlock()
myTextBlock1.FontSize = 20
myTextBlock1.Margin = New Thickness(10)
myTextBlock1.Text = "Grid Element Docked at the Top"
Grid.SetRow(myTextBlock1, 0)
Grid.SetColumnSpan(myTextBlock1, 3)
Dim myButton3 As New Button()
myButton3.Margin = New Thickness(5)
myButton3.Content = "A Row"
Grid.SetColumn(myButton3, 0)
Grid.SetRow(myButton3, 1)
Dim myButton4 As New Button()
myButton4.Margin = New Thickness(5)
myButton4.Content = "of Button"
Grid.SetColumn(myButton4, 1)
Grid.SetRow(myButton4, 1)
Dim myButton5 As New Button()
myButton5.Margin = New Thickness(5)
myButton5.Content = "Elements"
Grid.SetColumn(myButton5, 2)
Grid.SetRow(myButton5, 1)
myGrid.Children.Add(myTextBlock1)
myGrid.Children.Add(myButton3)
myGrid.Children.Add(myButton4)
myGrid.Children.Add(myButton5)
myBorder3.Child = myGrid

Dim myBorder4 As New Border()
myBorder4.BorderBrush = Brushes.Black
myBorder4.BorderThickness = New Thickness(1)
DockPanel.SetDock(myBorder4, Dock.Bottom)
Dim myStackPanel2 As New StackPanel()
myStackPanel2.Orientation = Orientation.Horizontal
Dim myTextBlock2 As New TextBlock()
myTextBlock2.Text = "This StackPanel is Docked to the Bottom"
myTextBlock2.Margin = New Thickness(5)
myStackPanel2.Children.Add(myTextBlock2)
myBorder4.Child = myStackPanel2

Dim myBorder5 As New Border()
myBorder5.BorderBrush = Brushes.Black
myBorder5.BorderThickness = New Thickness(1)
Dim myCanvas As New Canvas()
myCanvas.ClipToBounds = True
Dim myTextBlock3 As New TextBlock()
myTextBlock3.Text = "Content in the Canvas will Fill the remaining space."
Canvas.SetTop(myTextBlock3, 50)
Canvas.SetLeft(myTextBlock3, 50)
Dim myEllipse As New Ellipse()
myEllipse.Height = 100
myEllipse.Width = 125

```

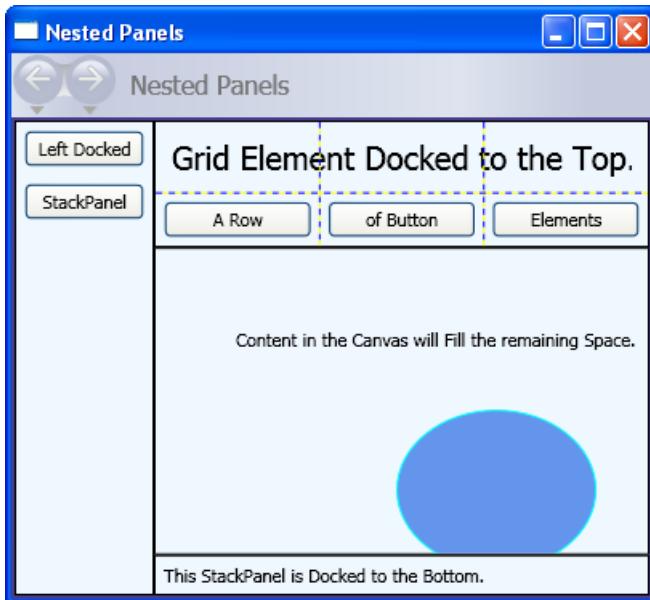
```

myEllipse.Fill = Brushes.CornflowerBlue
myEllipse.Stroke = Brushes.Aqua
Canvas.SetTop(myEllipse, 100)
Canvas.SetLeft(myEllipse, 150)
myCanvas.Children.Add(myTextBlock3)
myCanvas.Children.Add(myEllipse)
myBorder5.Child = myCanvas

myDockPanel.Children.Add(myBorder2)
myDockPanel.Children.Add(myBorder3)
myDockPanel.Children.Add(myBorder4)
myDockPanel.Children.Add(myBorder5)

```

The compiled application yields a new UI that looks like this.



## Custom Panel Elements

While WPF provides an array of flexible layout controls, custom layout behaviors can also be achieved by overriding the [ArrangeOverride](#) and [MeasureOverride](#) methods. Custom sizing and positioning can be accomplished by defining new positioning behaviors within these override methods.

Similarly, custom layout behaviors based on derived classes (such as [Canvas](#) or [Grid](#)) can be defined by overriding their [ArrangeOverride](#) and [MeasureOverride](#) methods.

The following markup demonstrates how to create a custom [Panel](#) element. This new [Panel](#), defined as [PlotPanel](#), supports the positioning of child elements through the use of hard-coded x- and y- coordinates. In this example, a [Rectangle](#) element (not shown) is positioned at plot point 50 (x), and 50 (y).

```
public:  
    ref class PlotPanel : Panel {  
  
public:  
    PlotPanel () {};  
  
protected:  
    // Override the default Measure method of Panel  
    virtual Size MeasureOverride(Size availableSize) override  
    {  
        Size^ panelDesiredSize = gcnew Size();  
  
        // In our example, we just have one child.  
        // Report that our panel requires just the size of its only child.  
        for each (UIElement^ child in InternalChildren)  
        {  
            child->Measure(availableSize);  
            panelDesiredSize = child->DesiredSize;  
        }  
        return *panelDesiredSize ;  
    }  
  
protected:  
    virtual System::Windows::Size ArrangeOverride (Size finalSize) override  
    {  
        for each (UIElement^ child in InternalChildren)  
        {  
            double x = 50;  
            double y = 50;  
            child->Arrange(Rect(Point(x, y), child->DesiredSize));  
        }  
        return finalSize;  
    };  
};
```

```

public class PlotPanel : Panel
{
    // Default public constructor
    public PlotPanel()
        : base()
    {

    }

    // Override the default Measure method of Panel
    protected override Size MeasureOverride(Size availableSize)
    {
        Size panelDesiredSize = new Size();

        // In our example, we just have one child.
        // Report that our panel requires just the size of its only child.
        foreach (UIElement child in InternalChildren)
        {
            child.Measure(availableSize);
            panelDesiredSize = child.DesiredSize;
        }

        return panelDesiredSize ;
    }
    protected override Size ArrangeOverride(Size finalSize)
    {
        foreach (UIElement child in InternalChildren)
        {
            double x = 50;
            double y = 50;

            child.Arrange(new Rect(new Point(x, y), child.DesiredSize));
        }
        return finalSize; // Returns the final Arranged size
    }
}

```

```

Public Class PlotPanel
    Inherits Panel
    'Override the default Measure method of Panel.

    Protected Overrides Function MeasureOverride(ByVal availableSize As System.Windows.Size) As
System.Windows.Size
        Dim panelDesiredSize As Size = New Size()
        ' In our example, we just have one child.
        ' Report that our panel requires just the size of its only child.
        For Each child As UIElement In InternalChildren
            child.Measure(availableSize)
            panelDesiredSize = child.DesiredSize
        Next
        Return panelDesiredSize
    End Function
    Protected Overrides Function ArrangeOverride(ByVal finalSize As System.Windows.Size) As
System.Windows.Size
        For Each child As UIElement In InternalChildren
            Dim x As Double = 50
            Dim y As Double = 50
            child.Arrange(New Rect(New System.Windows.Point(x, y), child.DesiredSize))
        Next
        Return finalSize
    End Function
End Class

```

To view a more complex custom panel implementation, see [Create a Custom Content-Wrapping Panel Sample](#).

# Localization/Globalization Support

WPF supports a number of features that assist in the creation of localizable UI.

All panel elements natively support the [FlowDirection](#) property, which can be used to dynamically re-flow content based on a user's locale or language settings. For more information, see [FlowDirection](#).

The [SizeToContent](#) property provides a mechanism that enables application developers to anticipate the needs of localized UI. Using the [WidthAndHeight](#) value of this property, a parent [Window](#) always sizes dynamically to fit content and is not constrained by artificial height or width restrictions.

[DockPanel](#), [Grid](#), and [StackPanel](#) are all good choices for localizable UI. [Canvas](#) is not a good choice, however, because it positions content absolutely, making it difficult to localize.

For additional information on creating WPF applications with localizable user interfaces (UIs)s, see the [Use Automatic Layout Overview](#).

## See also

- [Walkthrough: My first WPF desktop application](#)
- [WPF Layout Gallery Sample](#)
- [Layout](#)
- [WPF Controls Gallery Sample](#)
- [Alignment, Margins, and Padding Overview](#)
- [Create a Custom Content-Wrapping Panel Sample](#)
- [Attached Properties Overview](#)
- [Use Automatic Layout Overview](#)
- [Layout and Design](#)

# Panel How-to Topics

2 minutes to read • [Edit Online](#)

The topics in this section describe how to use the [Panel](#) element and related APIs.

## In This Section

- [Create a Custom Panel Element](#)
- [Override the Panel OnRender Method](#)
- [Set the Height Properties of an Element](#)
- [Set the Width Properties of an Element](#)

## Reference

- [Panel](#)
- [Canvas](#)
- [DockPanel](#)
- [Grid](#)
- [StackPanel](#)
- [VirtualizingStackPanel](#)
- [WrapPanel](#)

## Related Sections

- [Layout](#)
- [Walkthrough: My first WPF desktop application](#)
- [ScrollViewer Overview](#)

# How to: Create a Custom Panel Element

2 minutes to read • [Edit Online](#)

## Example

This example shows how to override the default layout behavior of the [Panel](#) element and create custom layout elements that are derived from [Panel](#).

The example defines a simple custom [Panel](#) element called `PlotPanel`, which positions child elements according to two hard-coded x- and y-coordinates. In this example, `x` and `y` are both set to `50`; therefore, all child elements are positioned at that location on the x and y axes.

To implement custom [Panel](#) behaviors, the example uses the [MeasureOverride](#) and [ArrangeOverride](#) methods. Each method returns the [Size](#) data that is necessary to position and render child elements.

```
public:  
    ref class PlotPanel : Panel {  
  
    public:  
        PlotPanel () {};  
  
    protected:  
        // Override the default Measure method of Panel  
        virtual Size MeasureOverride(Size availableSize) override  
        {  
            Size^ panelDesiredSize = gcnew Size();  
  
            // In our example, we just have one child.  
            // Report that our panel requires just the size of its only child.  
            for each (UIElement^ child in InternalChildren)  
            {  
                child->Measure(availableSize);  
                panelDesiredSize = child->DesiredSize;  
            }  
            return *panelDesiredSize ;  
        }  
  
    protected:  
        virtual System::Windows::Size ArrangeOverride (Size finalSize) override  
        {  
            for each (UIElement^ child in InternalChildren)  
            {  
                double x = 50;  
                double y = 50;  
                child->Arrange(Rect(Point(x, y), child->DesiredSize));  
            }  
            return finalSize;  
        };  
    };
```

```

public class PlotPanel : Panel
{
    // Default public constructor
    public PlotPanel()
        : base()
    {

    }

    // Override the default Measure method of Panel
    protected override Size MeasureOverride(Size availableSize)
    {
        Size panelDesiredSize = new Size();

        // In our example, we just have one child.
        // Report that our panel requires just the size of its only child.
        foreach (UIElement child in InternalChildren)
        {
            child.Measure(availableSize);
            panelDesiredSize = child.DesiredSize;
        }

        return panelDesiredSize ;
    }
    protected override Size ArrangeOverride(Size finalSize)
    {
        foreach (UIElement child in InternalChildren)
        {
            double x = 50;
            double y = 50;

            child.Arrange(new Rect(new Point(x, y), child.DesiredSize));
        }
        return finalSize; // Returns the final Arranged size
    }
}

```

```

Public Class PlotPanel
    Inherits Panel
    'Override the default Measure method of Panel.

    Protected Overrides Function MeasureOverride(ByVal availableSize As System.Windows.Size) As
System.Windows.Size
        Dim panelDesiredSize As Size = New Size()
        ' In our example, we just have one child.
        ' Report that our panel requires just the size of its only child.
        For Each child As UIElement In InternalChildren
            child.Measure(availableSize)
            panelDesiredSize = child.DesiredSize
        Next
        Return panelDesiredSize
    End Function
    Protected Overrides Function ArrangeOverride(ByVal finalSize As System.Windows.Size) As
System.Windows.Size
        For Each child As UIElement In InternalChildren
            Dim x As Double = 50
            Dim y As Double = 50
            child.Arrange(New Rect(New System.Windows.Point(x, y), child.DesiredSize))
        Next
        Return finalSize
    End Function
End Class

```

## See also

- [Panel](#)
- [Panels Overview](#)

# How to: Override the Panel OnRender Method

2 minutes to read • [Edit Online](#)

This example shows how to override the [OnRender](#) method of [Panel](#) in order to add custom graphical effects to a layout element.

## Example

Use the [OnRender](#) method in order to add graphical effects to a rendered panel element. For example, you can use this method to add custom border or background effects. A [DrawingContext](#) object is passed as an argument, which provides methods for drawing shapes, text, images, or videos. As a result, this method is useful for customization of a panel object.

```
// Override the OnRender call to add a Background and Border to the OffSetPanel
protected override void OnRender(DrawingContext dc)
{
    SolidColorBrush mySolidColorBrush = new SolidColorBrush();
    mySolidColorBrush.Color = Colors.LimeGreen;
    Pen myPen = new Pen(Brushes.Blue, 10);
    Rect myRect = new Rect(0, 0, 500, 500);
    dc.DrawRectangle(mySolidColorBrush, myPen, myRect);
}
```

```
' Override the OnRender call to add a Background and Border to the OffSetPanel
Protected Overrides Sub OnRender(ByVal dc As DrawingContext)
    Dim mySolidColorBrush As New SolidColorBrush()
    mySolidColorBrush.Color = Colors.LimeGreen
    Dim myPen As New Pen(Brushes.Blue, 10)
    Dim myRect As New Rect(0, 0, 500, 500)
    dc.DrawRectangle(mySolidColorBrush, myPen, myRect)
End Sub
```

## See also

- [Panel](#)
- [Panels Overview](#)
- [How-to Topics](#)

# How to: Set the Height Properties of an Element

3 minutes to read • [Edit Online](#)

## Example

This example visually shows the differences in rendering behavior among the four height-related properties in Windows Presentation Foundation (WPF).

The [FrameworkElement](#) class exposes four properties that describe the height characteristics of an element. These four properties can conflict, and when they do, the value that takes precedence is determined as follows: the [MinHeight](#) value takes precedence over the [MaxHeight](#) value, which in turn takes precedence over the [Height](#) value. A fourth property, [ActualHeight](#), is read-only, and reports the actual height as determined by interactions with the layout process.

The following Extensible Application Markup Language (XAML) examples draw a [Rectangle](#) element (`rect1`) as a child of [Canvas](#). You can change the height properties of a [Rectangle](#) by using a series of [ListBox](#) elements that represent the property values of [MinHeight](#), [MaxHeight](#), and [Height](#). In this manner, the precedence of each property is visually displayed.

```
<Canvas Height="200" MinWidth="200" Background="#b0c4de" VerticalAlignment="Top" HorizontalAlignment="Center" Name="myCanvas" Margin="0,0,0,50">
    <Rectangle HorizontalAlignment="Center" Canvas.Top="50" Canvas.Left="50" Name="rect1" Fill="#4682b4" Height="100" Width="100"/>
</Canvas>
```

```

<TextBlock Grid.Row="1" Grid.Column="0" Margin="10,0,0,0" TextWrapping="Wrap">Set the Rectangle Height:</TextBlock>
<ListBox Grid.Column="1" Grid.Row="1" Margin="10,0,0,0" Height="50" Width="50"
SelectionChanged="changeHeight">
<ListBoxItem>25</ListBoxItem>
<ListBoxItem>50</ListBoxItem>
<ListBoxItem>75</ListBoxItem>
<ListBoxItem>100</ListBoxItem>
<ListBoxItem>125</ListBoxItem>
<ListBoxItem>150</ListBoxItem>
<ListBoxItem>175</ListBoxItem>
<ListBoxItem>200</ListBoxItem>
</ListBox>

<TextBlock Grid.Row="1" Grid.Column="2" Margin="10,0,0,0" TextWrapping="Wrap">Set the Rectangle MinHeight:</TextBlock>
<ListBox Grid.Column="3" Grid.Row="1" Margin="10,0,0,0" Height="50" Width="50"
SelectionChanged="changeMinHeight">
<ListBoxItem>25</ListBoxItem>
<ListBoxItem>50</ListBoxItem>
<ListBoxItem>75</ListBoxItem>
<ListBoxItem>100</ListBoxItem>
<ListBoxItem>125</ListBoxItem>
<ListBoxItem>150</ListBoxItem>
<ListBoxItem>175</ListBoxItem>
<ListBoxItem>200</ListBoxItem>
</ListBox>

<TextBlock Grid.Row="1" Grid.Column="4" Margin="10,0,0,0" TextWrapping="Wrap">Set the Rectangle MaxHeight:</TextBlock>
<ListBox Grid.Column="5" Grid.Row="1" Margin="10,0,0,0" Height="50" Width="50"
SelectionChanged="changeMaxHeight">
<ListBoxItem>25</ListBoxItem>
<ListBoxItem>50</ListBoxItem>
<ListBoxItem>75</ListBoxItem>
<ListBoxItem>100</ListBoxItem>
<ListBoxItem>125</ListBoxItem>
<ListBoxItem>150</ListBoxItem>
<ListBoxItem>175</ListBoxItem>
<ListBoxItem>200</ListBoxItem>
</ListBox>

```

The following code-behind examples handle the events that the [SelectionChanged](#) event raises. Each handler takes the input from the [ListBox](#), parses the value as a [Double](#), and applies the value to the specified height-related property. The height values are also converted to a string and written to various [TextBlock](#) elements (definition of those elements is not shown in the selected XAML).

```
private void changeHeight(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    Double sz1 = Double.Parse(li.Content.ToString());
    rect1.Height = sz1;
    rect1.UpdateLayout();
    txt1.Text= "ActualHeight is set to " + rect1.ActualHeight;
    txt2.Text= "Height is set to " + rect1.Height;
    txt3.Text= "MinHeight is set to " + rect1.MinHeight;
    txt4.Text= "MaxHeight is set to " + rect1.MaxHeight;
}
private void changeMinHeight(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    Double sz1 = Double.Parse(li.Content.ToString());
    rect1.MinHeight = sz1;
    rect1.UpdateLayout();
    txt1.Text= "ActualHeight is set to " + rect1.ActualHeight;
    txt2.Text= "Height is set to " + rect1.Height;
    txt3.Text= "MinHeight is set to " + rect1.MinHeight;
    txt4.Text= "MaxHeight is set to " + rect1.MaxHeight;
}
private void changeMaxHeight(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    Double sz1 = Double.Parse(li.Content.ToString());
    rect1.MaxHeight = sz1;
    rect1.UpdateLayout();
    txt1.Text= "ActualHeight is set to " + rect1.ActualHeight;
    txt2.Text= "Height is set to " + rect1.Height;
    txt3.Text= "MinHeight is set to " + rect1.MinHeight;
    txt4.Text= "MaxHeight is set to " + rect1.MaxHeight;
}
```

```

Private Sub changeHeight(ByVal sender As Object, ByVal args As SelectionChangedEventArgs)
    Dim li As ListBoxItem = CType(CType(sender, ListBox).SelectedItem, ListBoxItem)
    Dim sz1 As Double = Double.Parse(li.Content.ToString())
    rect1.Height = sz1
    rect1.UpdateLayout()
    txt1.Text = "ActualHeight is set to " + rect1.ActualHeight.ToString()
    txt2.Text = "Height is set to " + rect1.Height.ToString()
    txt3.Text = "MinHeight is set to " + rect1.MinHeight.ToString()
    txt4.Text = "MaxHeight is set to " + rect1.MaxHeight.ToString()
End Sub
Private Sub changeMinHeight(ByVal sender As Object, ByVal args As SelectionChangedEventArgs)

    Dim li As ListBoxItem = CType(CType(sender, ListBox).SelectedItem, ListBoxItem)
    Dim sz1 As Double = Double.Parse(li.Content.ToString())
    rect1.MinHeight = sz1
    rect1.UpdateLayout()
    txt1.Text = "ActualHeight is set to " + rect1.ActualHeight.ToString()
    txt2.Text = "Height is set to " + rect1.Height.ToString()
    txt3.Text = "MinHeight is set to " + rect1.MinHeight.ToString()
    txt4.Text = "MaxHeight is set to " + rect1.MaxHeight.ToString()
End Sub
Private Sub changeMaxHeight(ByVal sender As Object, ByVal args As SelectionChangedEventArgs)

    Dim li As ListBoxItem = CType(CType(sender, ListBox).SelectedItem, ListBoxItem)
    Dim sz1 As Double = Double.Parse(li.Content.ToString())
    rect1.MaxHeight = sz1
    rect1.UpdateLayout()
    txt1.Text = "ActualHeight is set to " + rect1.ActualHeight.ToString()
    txt2.Text = "Height is set to " + rect1.Height.ToString()
    txt3.Text = "MinHeight is set to " + rect1.MinHeight.ToString()
    txt4.Text = "MaxHeight is set to " + rect1.MaxHeight.ToString()
End Sub

```

For the complete sample, see [Height Properties Sample](#).

## See also

- [FrameworkElement](#)
- [ListBox](#)
- [ActualHeight](#)
- [MaxHeight](#)
- [MinHeight](#)
- [Height](#)
- [Set the Width Properties of an Element](#)
- [Panels Overview](#)
- [Height Properties Sample](#)

# How to: Set the Width Properties of an Element

3 minutes to read • [Edit Online](#)

## Example

This example visually shows the differences in rendering behavior among the four width-related properties in Windows Presentation Foundation (WPF).

The [FrameworkElement](#) class exposes four properties that describe the width characteristics of an element. These four properties can conflict, and when they do, the value that takes precedence is determined as follows: the [MinWidth](#) value takes precedence over the [MaxWidth](#) value, which in turn takes precedence over the [Width](#) value. A fourth property, [ActualWidth](#), is read-only, and reports the actual width as determined by interactions with the layout process.

The following Extensible Application Markup Language (XAML) examples draw a [Rectangle](#) element (`rect1`) as a child of [Canvas](#). You can change the width properties of a [Rectangle](#) by using a series of [ListBox](#) elements that represent the property values of [MinWidth](#), [MaxWidth](#), and [Width](#). In this manner, the precedence of each property is visually displayed.

```
<Canvas Height="200" MinWidth="200" Background="#b0c4de" VerticalAlignment="Top" HorizontalAlignment="Center" Name="myCanvas">
    <Rectangle HorizontalAlignment="Center" Canvas.Top="50" Canvas.Left="50" Name="rect1" Fill="#4682b4" Width="100" Height="100"/>
</Canvas>
```

```

<TextBlock Grid.Row="1" Grid.Column="0" Margin="10,0,0,0" TextWrapping="Wrap">Set the Rectangle Width:</TextBlock>
<ListBox Grid.Column="1" Grid.Row="1" Margin="10,0,0,0" Width="50" Height="50"
SelectionChanged="changeWidth">
<ListBoxItem>25</ListBoxItem>
<ListBoxItem>50</ListBoxItem>
<ListBoxItem>75</ListBoxItem>
<ListBoxItem>100</ListBoxItem>
<ListBoxItem>125</ListBoxItem>
<ListBoxItem>150</ListBoxItem>
<ListBoxItem>175</ListBoxItem>
<ListBoxItem>200</ListBoxItem>
<ListBoxItem>225</ListBoxItem>
<ListBoxItem>250</ListBoxItem>
</ListBox>

<TextBlock Grid.Row="1" Grid.Column="2" Margin="10,0,0,0" TextWrapping="Wrap">Set the Rectangle MinWidth:</TextBlock>
<ListBox Grid.Column="3" Grid.Row="1" Margin="10,0,0,0" Width="50" Height="50"
SelectionChanged="changeMinWidth">
<ListBoxItem>25</ListBoxItem>
<ListBoxItem>50</ListBoxItem>
<ListBoxItem>75</ListBoxItem>
<ListBoxItem>100</ListBoxItem>
<ListBoxItem>125</ListBoxItem>
<ListBoxItem>150</ListBoxItem>
<ListBoxItem>175</ListBoxItem>
<ListBoxItem>200</ListBoxItem>
<ListBoxItem>225</ListBoxItem>
<ListBoxItem>250</ListBoxItem>
</ListBox>

<TextBlock Grid.Row="1" Grid.Column="4" Margin="10,0,0,0" TextWrapping="Wrap">Set the Rectangle MaxWidth:</TextBlock>
<ListBox Grid.Column="5" Grid.Row="1" Margin="10,0,0,0" Width="50" Height="50"
SelectionChanged="changeMaxWidth">
<ListBoxItem>25</ListBoxItem>
<ListBoxItem>50</ListBoxItem>
<ListBoxItem>75</ListBoxItem>
<ListBoxItem>100</ListBoxItem>
<ListBoxItem>125</ListBoxItem>
<ListBoxItem>150</ListBoxItem>
<ListBoxItem>175</ListBoxItem>
<ListBoxItem>200</ListBoxItem>
<ListBoxItem>225</ListBoxItem>
<ListBoxItem>250</ListBoxItem>
</ListBox>

```

The following code-behind examples handle the events that the [SelectionChanged](#) event raises. Each custom method takes the input from the [ListBox](#), parses the value as a [Double](#), and applies the value to the specified width-related property. The width values are also converted to a string and written to various [TextBlock](#) elements (definition of those elements is not shown in the selected XAML).

```
private void changeWidth(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    Double sz1 = Double.Parse(li.Content.ToString());
    rect1.Width = sz1;
    rect1.UpdateLayout();
    txt1.Text = "ActualWidth is set to " + rect1.ActualWidth;
    txt2.Text = "Width is set to " + rect1.Width;
    txt3.Text = "MinWidth is set to " + rect1.MinWidth;
    txt4.Text = "MaxWidth is set to " + rect1.MaxWidth;
}
private void changeMinWidth(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    Double sz1 = Double.Parse(li.Content.ToString());
    rect1.MinWidth = sz1;
    rect1.UpdateLayout();
    txt1.Text = "ActualWidth is set to " + rect1.ActualWidth;
    txt2.Text = "Width is set to " + rect1.Width;
    txt3.Text = "MinWidth is set to " + rect1.MinWidth;
    txt4.Text = "MaxWidth is set to " + rect1.MaxWidth;
}
private void changeMaxWidth(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    Double sz1 = Double.Parse(li.Content.ToString());
    rect1.MaxWidth = sz1;
    rect1.UpdateLayout();
    txt1.Text = "ActualWidth is set to " + rect1.ActualWidth;
    txt2.Text = "Width is set to " + rect1.Width;
    txt3.Text = "MinWidth is set to " + rect1.MinWidth;
    txt4.Text = "MaxWidth is set to " + rect1.MaxWidth;
}
```

```

Private Sub changeWidth(ByVal sender As Object, ByVal args As SelectionChangedEventArgs)
    Dim li As ListBoxItem = CType(CType(sender, ListBox).SelectedItem, ListBoxItem)
    Dim sz1 As Double = Double.Parse(li.Content.ToString())
    rect1.Width = sz1
    rect1.UpdateLayout()
    txt1.Text = "ActualWidth is set to " + rect1.ActualWidth.ToString()
    txt2.Text = "Width is set to " + rect1.Width.ToString()
    txt3.Text = "MinWidth is set to " + rect1.MinWidth.ToString()
    txt4.Text = "MaxWidth is set to " + rect1.MaxWidth.ToString()
End Sub
Private Sub changeMinWidth(ByVal sender As Object, ByVal args As SelectionChangedEventArgs)

    Dim li As ListBoxItem = CType(CType(sender, ListBox).SelectedItem, ListBoxItem)
    Dim sz1 As Double = Double.Parse(li.Content.ToString())
    rect1.MinWidth = sz1
    rect1.UpdateLayout()
    txt1.Text = "ActualWidth is set to " + rect1.ActualWidth.ToString()
    txt2.Text = "Width is set to " + rect1.Width.ToString()
    txt3.Text = "MinWidth is set to " + rect1.MinWidth.ToString()
    txt4.Text = "MaxWidth is set to " + rect1.MaxWidth.ToString()
End Sub
Private Sub changeMaxWidth(ByVal sender As Object, ByVal args As SelectionChangedEventArgs)

    Dim li As ListBoxItem = CType(CType(sender, ListBox).SelectedItem, ListBoxItem)
    Dim sz1 As Double = Double.Parse(li.Content.ToString())
    rect1.MaxWidth = sz1
    rect1.UpdateLayout()
    txt1.Text = "ActualWidth is set to " + rect1.ActualWidth.ToString()
    txt2.Text = "Width is set to " + rect1.Width.ToString()
    txt3.Text = "MinWidth is set to " + rect1.MinWidth.ToString()
    txt4.Text = "MaxWidth is set to " + rect1.MaxWidth.ToString()
End Sub

```

For the complete sample, see [Width Properties Comparison Sample](#).

## See also

- [ListBox](#)
- [FrameworkElement](#)
- [ActualWidth](#)
- [MaxWidth](#)
- [MinWidth](#)
- [Width](#)
- [Panels Overview](#)
- [Set the Height Properties of an Element](#)
- [Width Properties Comparison Sample](#)

# PasswordBox

2 minutes to read • [Edit Online](#)

The [PasswordBox](#) control is used to input sensitive or private information.

## See also

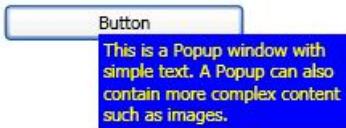
- [TextBox](#)
- [RichTextBox](#)
- [Control Library](#)

# Popup

2 minutes to read • [Edit Online](#)

The [Popup](#) control displays content in a separate window that floats over the current application window.

The following illustration shows a [Popup](#) control that is positioned with respect to a [Button](#) that is its parent:



## In This Section

[Popup Overview](#)

[Popup Placement Behavior](#)

[How-to Topics](#)

## Reference

[Popup](#)

## Related Sections

# Popup Overview

4 minutes to read • [Edit Online](#)

The [Popup](#) control provides a way to display content in a separate window that floats over the current application window relative to a designated element or screen coordinate. This topic introduces the [Popup](#) control and provides information about its use.

## What Is a Popup?

A [Popup](#) control displays content in a separate window relative to an element or point on the screen. When the [Popup](#) is visible, the [IsOpen](#) property is set to `true`.

### NOTE

A [Popup](#) does not automatically open when the mouse pointer moves over its parent object. If you want a [Popup](#) to automatically open, use the [ToolTip](#) or [ToolTipService](#) class. For more information, see [ToolTip Overview](#).

## Creating a Popup

The following example shows how to define a [Popup](#) control that is the child element of a [Button](#) control. Because a [Button](#) can have only one child element, this example places the text for the [Button](#) and the [Popup](#) controls in a [StackPanel](#). The content of the [Popup](#) appears in a [TextBlock](#) control, which displays its text in a separate window that floats over the application window near the related [Button](#) control.

```
<Button HorizontalAlignment="Left" Click="DisplayPopup"
    Width="150" Margin="20,10,0,0">
    <StackPanel>
        <TextBlock>Display Your Popup Text</TextBlock>
        <Popup Name="myPopup">
            <TextBlock Name="myPopupText"
                Background="LightBlue"
                Foreground="Blue">
                Popup Text
            </TextBlock>
        </Popup>
    </StackPanel>
</Button>
```

```
<Button Name="ButtonForPopup" HorizontalAlignment="Left"
    Click="CreatePopup"
    Width="150" Margin="20,10,0,0">
    <StackPanel Name="aStackPanel">
        <TextBlock>Create Popup</TextBlock>
    </StackPanel>
</Button>
```

## Controls That Implement a Popup

You can build [Popup](#) controls into other controls. The following controls implement the [Popup](#) control for specific uses:

- [ToolTip](#). If you want to create a tooltip for an element, use the [ToolTip](#) and [ToolTipService](#) classes. For more

information, see [ToolTip Overview](#).

- **ContextMenu**. If you want to create a context menu for an element, use the [ContextMenu](#) control. For more information, see [ContextMenu Overview](#).
- **ComboBox**. If you want to create a selection control that has a drop-down list box that can be shown or hidden, use the [ComboBox](#) control.
- **Expander**. If you want to create a control that displays a header with a collapsible area that displays content, use the [Expander](#) control. For more information, see [Expander Overview](#).

## Popup Behavior and Appearance

The [Popup](#) control provides functionality that enables you to customize its behavior and appearance. For example, you can set open and close behavior, animation, opacity and bitmap effects, and [Popup](#) size and position.

### Open and Close Behavior

A [Popup](#) control displays its content when the [IsOpen](#) property is set to `true`. By default, [Popup](#) stays open until the [IsOpen](#) property is set to `false`. However, you can change the default behavior by setting the [StaysOpen](#) property to `false`. When you set this property to `false`, the [Popup](#) content window has mouse capture. The [Popup](#) loses mouse capture and the window closes when a mouse event occurs outside the [Popup](#) window.

The [Opened](#) and [Closed](#) events are raised when the [Popup](#) content window is open or closed.

### Animation

The [Popup](#) control has built-in support for the animations that are typically associated with behaviors like fade-in and slide-in. You can turn on these animations by setting the [PopupAnimation](#) property to a [PopupAnimation](#) enumeration value. For [Popup](#) animations to work correctly, you must set the [AllowsTransparency](#) property to `true`.

You can also apply animations like [Storyboard](#) to the [Popup](#) control.

### Opacity and Bitmap Effects

The [Opacity](#) property for a [Popup](#) control has no effect on its content. By default, the [Popup](#) content window is opaque. To create a transparent [Popup](#), set the [AllowsTransparency](#) property to `true`.

The content of a [Popup](#) does not inherit bitmap effects, such as [DropShadowBitmapEffect](#), that you directly set on the [Popup](#) control or on any other element in the parent window. For bitmap effects to appear on the content of a [Popup](#), you must set the bitmap effect directly on its content. For example, if the child of a [Popup](#) is a [StackPanel](#), set the bitmap effect on the [StackPanel](#).

### Popup Size

By default, a [Popup](#) is automatically sized to its content. When auto-sizing occurs, some bitmap effects may be hidden because the default size of the screen area that is defined for the [Popup](#) content does not provide enough space for the bitmap effects to display.

[Popup](#) content can also be obscured when you set a [RenderTransform](#) on the content. In this scenario, some content might be hidden if the content of the transformed [Popup](#) extends beyond the area of the original [Popup](#). If a bitmap effect or transform requires more space, you can define a margin around the [Popup](#) content in order to provide more area for the control.

## Defining the Popup Position

You can position a [popup](#) by setting the [PlacementTarget](#), [PlacementRectangle](#), [Placement](#), [HorizontalOffset](#), and [VerticalOffsetProperty](#) properties. For more information, see [Popup Placement Behavior](#). When [Popup](#) is displayed on the screen, it does not reposition itself if its parent is repositioned.

## Customizing Popup Placement

You can customize the placement of a [Popup](#) control by specifying a set of coordinates that are relative to the [PlacementTarget](#) where you want the [Popup](#) to appear.

To customize placement, set the [Placement](#) property to [Custom](#). Then define a [CustomPopupPlacementCallback](#) delegate that returns a set of possible placement points and primary axes (in order of preference) for the [Popup](#). The point that shows the largest part of the [Popup](#) is automatically selected. For an example, see [Specify a Custom Popup Position](#).

## Popup and the Visual Tree

A [Popup](#) control does not have its own visual tree; it instead returns a size of 0 (zero) when the [MeasureOverride](#) method for [Popup](#) is called. However, when you set the [IsOpen](#) property of [Popup](#) to `true`, a new window with its own visual tree is created. The new window contains the [Child](#) content of [Popup](#). The width and height of the new window cannot be larger than 75 percent of the width or height of the screen.

The [Popup](#) control maintains a reference to its [Child](#) content as a logical child. When the new window is created, the content of [Popup](#) becomes a visual child of the window and remains the logical child of [Popup](#). Conversely, [Popup](#) remains the logical parent of its [Child](#) content.

## See also

- [Popup](#)
- [PopupPrimaryAxis](#)
- [PlacementMode](#)
- [CustomPopupPlacement](#)
- [CustomPopupPlacementCallback](#)
- [ToolTip](#)
- [ToolTipService](#)
- [How-to Topics](#)
- [How-to Topics](#)

# Popup Placement Behavior

11 minutes to read • [Edit Online](#)

A [Popup](#) control displays content in a separate window that floats over an application. You can specify the position of a [Popup](#) relative to a control, the mouse, or the screen by using the [PlacementTarget](#), [Placement](#), [PlacementRectangle](#), [HorizontalOffset](#), and [VerticalOffset](#) properties. These properties work together to give you flexibility in specifying the position of the [Popup](#).

## NOTE

The [ToolTip](#) and [ContextMenu](#) classes also define these five properties and behave similarly.

## Positioning the Popup

The placement of a [Popup](#) can be relative to a [UIElement](#) or to the entire screen. The following example creates four [Popup](#) controls that are relative to a [UIElement](#)—in this case, an image. All of the [Popup](#) controls have the [PlacementTarget](#) property set to `image1`, but each [Popup](#) has a different value for the [placement](#) property.

```
<Canvas Width="200" Height="150">
    <Image Name="image1"
        Canvas.Left="75"
        Source="Water_lilies.jpg" Height="200" Width="200"/>
    <Popup IsOpen="True" PlacementTarget="{Binding ElementName=image1}"
        Placement="Bottom">
        <TextBlock FontSize="14" Background="LightGreen">Placement=Bottom</TextBlock>
    </Popup>
    <Popup IsOpen="True" PlacementTarget="{Binding ElementName=image1}"
        Placement="Top">
        <TextBlock FontSize="14" Background="LightGreen">Placement=Top</TextBlock>
    </Popup>
    <Popup IsOpen="True" PlacementTarget="{Binding ElementName=image1}"
        Placement="Left">
        <TextBlock FontSize="14" Background="LightGreen">Placement=Left</TextBlock>
    </Popup>
    <Popup IsOpen="True" PlacementTarget="{Binding ElementName=image1}"
        Placement="Right">
        <TextBlock FontSize="14" Background="LightGreen">Placement=Right</TextBlock>
    </Popup>
</Canvas>
```

The following illustration shows the image and the [Popup](#) controls



This simple example demonstrates how to set the [PlacementTarget](#) and [Placement](#) properties, but by using the [PlacementRectangle](#), [HorizontalOffset](#), and [VerticalOffset](#) properties, you have even more control over where the [Popup](#) is positioned.

## Definitions of Terms: The Anatomy of a Popup

The following terms are useful in understanding how the [PlacementTarget](#), [Placement](#), [PlacementRectangle](#), [HorizontalOffset](#), and [VerticalOffset](#) properties relate to each other and the [Popup](#):

- Target object
- Target area
- Target origin
- Popup alignment point

These terms provide a convenient way to refer to various aspects of the [Popup](#) and the control that it is associated with.

### Target Object

The *target object* is the element that the [Popup](#) is associated with. If the [PlacementTarget](#) property is set, it specifies the target object. If [PlacementTarget](#) is not set, and the [Popup](#) has a parent, the parent is the target object. If there is no [PlacementTarget](#) value and no parent, there is no target object, and the [Popup](#) is positioned relative to the screen.

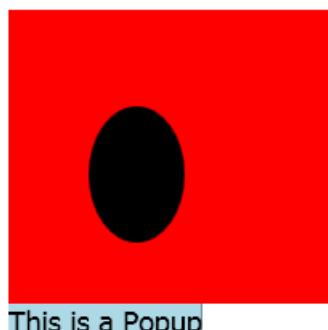
The following example creates a [Popup](#) that is the child of a [Canvas](#). The example does not set the [PlacementTarget](#) property on the [Popup](#). The default value for [Placement](#) is [PlacementMode.Bottom](#), so the [Popup](#) appears below the [Canvas](#).

```
<Canvas Margin="5" Background="Red" Width="200" Height="150" >

    <Ellipse Canvas.Top="60" Canvas.Left="50"
        Height="85" Width="60"
        Fill="Black"/>

    <Popup IsOpen="True" >
        <TextBlock Background="LightBlue" FontSize="18">This is a Popup</TextBlock>
    </Popup>
</Canvas>
```

The following illustration shows that the [Popup](#) is positioned relative to the [Canvas](#).



The following example creates a [Popup](#) that is the child of a [Canvas](#), but this time the [PlacementTarget](#) is set to `ellipse1`, so the popup appears below the [Ellipse](#).

```

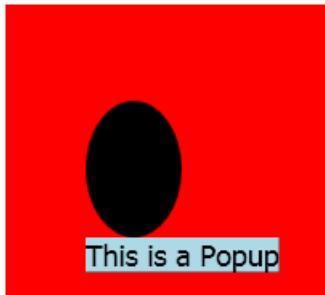
<Canvas Margin="5" Background="Red" Width="200" Height="150" >

    <Ellipse Name="ellipse1"
        Canvas.Top="60" Canvas.Left="50"
        Height="85" Width="60"
        Fill="Black"/>

    <Popup IsOpen="True" PlacementTarget="{Binding ElementName=ellipse1}">
        <TextBlock Background="LightBlue" FontSize="18">This is a Popup</TextBlock>
    </Popup>
</Canvas>

```

The following illustration shows that the [Popup](#) is positioned relative to the [Ellipse](#).



#### NOTE

For [ToolTip](#), the default value of [Placement](#) is [Mouse](#). For [ContextMenu](#), the default value of [Placement](#) is [MousePoint](#). These values are explained later, in "How the Properties Work Together."

### Target Area

The *target area* is the area on the screen that the [Popup](#) is relative to. In the previous examples, the [Popup](#) is aligned with the bounds of the target object, but in some cases, the [Popup](#) is aligned to other bounds, even if the [Popup](#) has a target object. If the [PlacementRectangle](#) property is set, the target area is different than the bounds of the target object.

The following example creates two [Canvas](#) objects, each one containing a [Rectangle](#) and a [Popup](#). In both cases, the target object for the [Popup](#) is the [Canvas](#). The [Popup](#) in the first [Canvas](#) has the [PlacementRectangle](#) set, with its [X](#), [Y](#), [Width](#), and [Height](#) properties set to 50, 50, 50, and 100, respectively. The [Popup](#) in the second [Canvas](#) does not have the [PlacementRectangle](#) set. As a result, the first [Popup](#) is positioned below the [PlacementRectangle](#) and the second [Popup](#) is positioned below the [Canvas](#). Each [Canvas](#) also contains a [Rectangle](#) that has the same bounds as the [PlacementRectangle](#) for the first [Popup](#). Note that the [PlacementRectangle](#) does not create a visible element in the application; the example creates a [Rectangle](#) to represent the [PlacementRectangle](#).

```

<StackPanel Orientation="Horizontal" Margin="50,50,0,0">

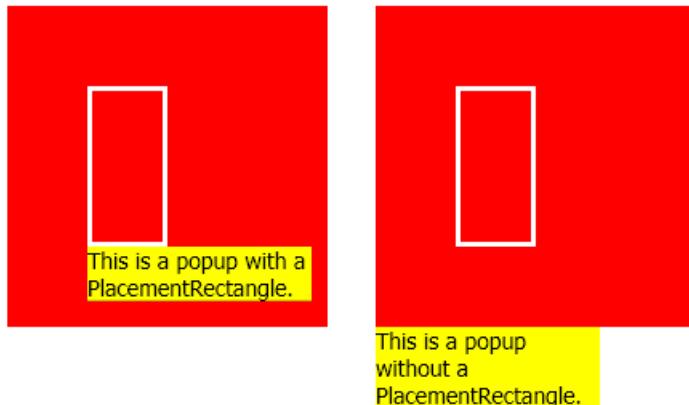
    <Canvas Width="200" Height="200" Background="Red">
        <Rectangle Canvas.Top="50" Canvas.Left="50"
            Width="50" Height="100"
            Stroke="White" StrokeThickness="3"/>
        <Popup IsOpen="True" PlacementRectangle="50,50,50,100">
            <TextBlock FontSize="14" Background="Yellow"
                Width="140" TextWrapping="Wrap">
                This is a popup with a PlacementRectangle.
            </TextBlock>
        </Popup>
    </Canvas>

    <Canvas Width="200" Height="200" Background="Red" Margin="30,0,0,0">
        <Rectangle Canvas.Top="50" Canvas.Left="50"
            Width="50" Height="100"
            Stroke="White" StrokeThickness="3"/>
        <Popup IsOpen="True">
            <TextBlock FontSize="14" Background="Yellow"
                Width="140" TextWrapping="Wrap">
                This is a popup without a PlacementRectangle.
            </TextBlock>
        </Popup>
    </Canvas>

</StackPanel>

```

The following illustration shows the result of the preceding example.



### Target Origin and Popup Alignment Point

The *target origin* and *popup alignment point* are reference points on the target area and popup, respectively, that are used for positioning. You can use the [HorizontalOffset](#) and [VerticalOffset](#) properties to offset the popup from the target area. The [HorizontalOffset](#) and [VerticalOffset](#) are relative to the target origin and the popup alignment point. The value of the [Placement](#) property determines where the target origin and popup alignment point are located.

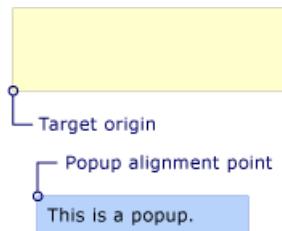
The following example creates a [Popup](#) and sets the [HorizontalOffset](#) and [VerticalOffset](#) properties to 20. The [Placement](#) property is set to [Bottom](#) (the default), so the target origin is the bottom-left corner of the target area and the popup alignment point is the top-left corner of the [Popup](#).

```

<Canvas Width="200" Height="200" Background="Yellow" Margin="20">
    <Popup IsOpen="True" Placement="Bottom"
        HorizontalOffset="20" VerticalOffset="20">
        <TextBlock FontSize="14" Background="#42F3FD">
            This is a popup.
        </TextBlock>
    </Popup>
</Canvas>

```

The following illustration shows the result of the preceding example.



## How the Properties Work Together

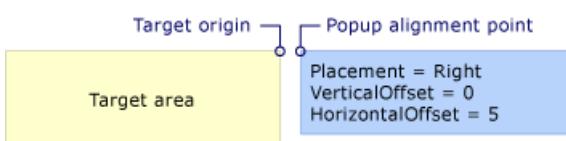
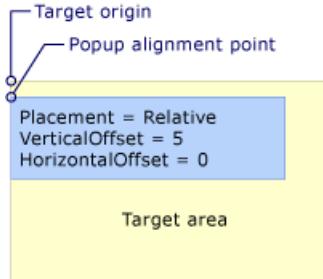
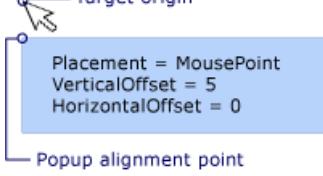
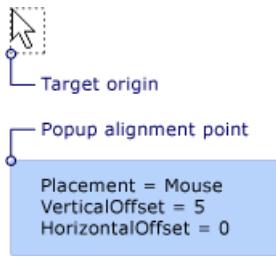
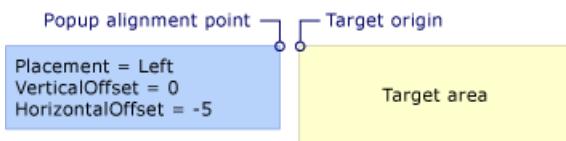
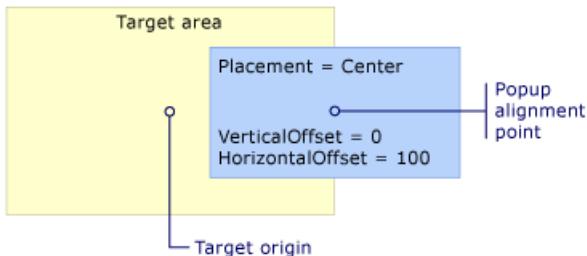
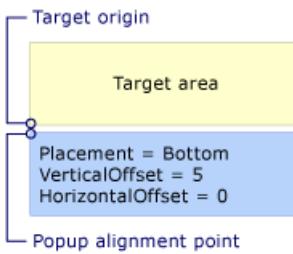
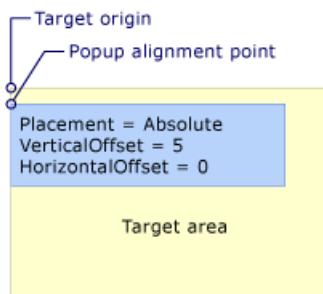
The values of [PlacementTarget](#), [PlacementRectangle](#), and [Placement](#) need to be considered together to figure out the correct target area, target origin, and popup alignment point. For example, if the value of [Placement](#) is [Mouse](#), there is no target object, the [PlacementRectangle](#) is ignored, and the target area is the bounds of the mouse pointer. On the other hand, if [Placement](#) is [Bottom](#), the [PlacementTarget](#) or parent determines the target object and [PlacementRectangle](#) determines the target area.

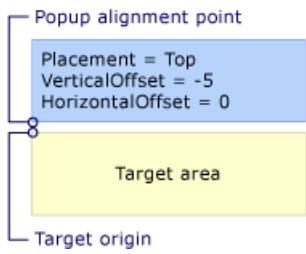
The following table describes the target object, target area, target origin, and popup alignment point and indicates whether [PlacementTarget](#) and [PlacementRectangle](#) are used for each [PlacementMode](#) enumeration value.

PLACEMENTMODE	TARGET OBJECT	TARGET AREA	TARGET ORIGIN	POPUP ALIGNMENT POINT
Absolute	Not applicable. <a href="#">PlacementTarget</a> is ignored.	The screen, or <a href="#">PlacementRectangle</a> if it is set. The <a href="#">PlacementRectangle</a> is relative to the screen.	The top-left corner of the target area.	The top-left corner of the <a href="#">Popup</a> .
AbsolutePoint	Not applicable. <a href="#">PlacementTarget</a> is ignored.	The screen, or <a href="#">PlacementRectangle</a> if it is set. The <a href="#">PlacementRectangle</a> is relative to the screen.	The top-left corner of the target area.	The top-left corner of the <a href="#">Popup</a> .
Bottom	<a href="#">PlacementTarget</a> or parent.	The target object, or <a href="#">PlacementRectangle</a> if it is set. The <a href="#">PlacementRectangle</a> is relative to the target object.	The bottom-left corner of the target area.	The top-left corner of the <a href="#">Popup</a> .
Center	<a href="#">PlacementTarget</a> or parent.	The target object, or <a href="#">PlacementRectangle</a> if it is set. The <a href="#">PlacementRectangle</a> is relative to the target object.	The center of the target area.	The center of the <a href="#">Popup</a> .

PLACEMENTMODE	TARGET OBJECT	TARGET AREA	TARGET ORIGIN	POPUP ALIGNMENT POINT
Custom	PlacementTarget or parent.	The target object, or PlacementRectangle if it is set. The PlacementRectangle is relative to the target object.	Defined by the CustomPopupPlacementCallback.	Defined by the CustomPopupPlacementCallback.
Left	PlacementTarget or parent.	The target object, or PlacementRectangle if it is set. The PlacementRectangle is relative to the target object.	The top-left corner of the target area.	The top-right corner of the Popup.
Mouse	Not applicable. PlacementTarget is ignored.	The bounds of the mouse pointer. PlacementRectangle is ignored.	The bottom-left corner of the target area.	The top-left corner of the Popup.
MousePoint	Not applicable. PlacementTarget is ignored.	The bounds of the mouse pointer. PlacementRectangle is ignored.	The top-left corner of the target area.	The top-left corner of the Popup.
Relative	PlacementTarget or parent.	The target object, or PlacementRectangle if it is set. The PlacementRectangle is relative to the target object.	The top-left corner of the target area.	The top-left corner of the Popup.
RelativePoint	PlacementTarget or parent.	The target object, or PlacementRectangle if it is set. The PlacementRectangle is relative to the target object.	The top-left corner of the target area.	The top-left corner of the Popup.
Right	PlacementTarget or parent.	The target object, or PlacementRectangle if it is set. The PlacementRectangle is relative to the target object.	The top-right corner of the target area.	The top-left corner of the Popup.
Top	PlacementTarget or parent.	The target object, or PlacementRectangle if it is set. The PlacementRectangle is relative to the target object.	The top-left corner of the target area.	The bottom-left corner of the Popup.

The following illustrations show the [Popup](#), target area, target origin, and popup alignment point for each PlacementMode value. In each figure, the target area is yellow, and the [Popup](#) is blue.





## When the Popup Encounters the Edge of the Screen

For security reasons, a [Popup](#) cannot be hidden by the edge of a screen. One of the following three things happens when the [Popup](#) encounters a screen edge:

- The popup realigns itself along the screen edge that would obscure the [Popup](#).
- The popup uses a different popup alignment point.
- The popup uses a different target origin and popup alignment point.

These options are described further later in this section.

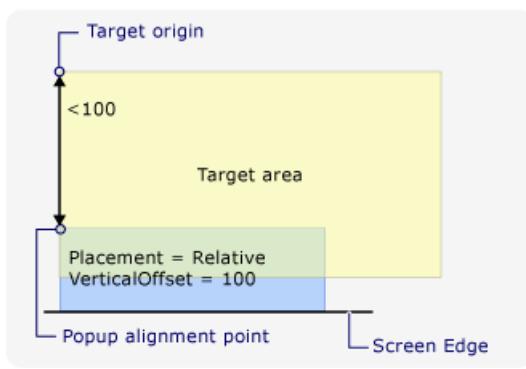
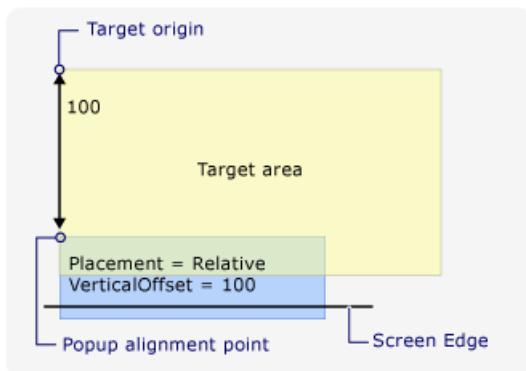
The behavior of the [Popup](#) when it encounters a screen edge depends on the value of the [Placement](#) property and which screen edge the popup encounters. The following table summarizes the behavior when the [Popup](#) encounters a screen edge for each [PlacementMode](#) value.

PLACEMENTMODE	TOP EDGE	BOTTOM EDGE	LEFT EDGE	RIGHT EDGE
Absolute	Aligns to the top edge.	Aligns to the bottom edge.	Aligns to the left edge.	Aligns to the right edge.
AbsolutePoint	Aligns to the top edge.	The popup alignment point changes to the bottom-left corner of the <a href="#">Popup</a> .	Aligns to the left edge.	The popup alignment point changes to the top-right corner of the <a href="#">Popup</a> .
Bottom	Aligns to the top edge.	The target origin changes to the top-left corner of the target area and the popup alignment point changes to the bottom-left corner of the <a href="#">Popup</a> .	Aligns to the left edge.	Aligns to the right edge.
Center	Aligns to the top edge.	Aligns to the bottom edge.	Aligns to the left edge.	Aligns to the right edge.
Left	Aligns to the top edge.	Aligns to the bottom edge.	The target origin changes to the top-right corner of the target area and the popup alignment point changes to the top-left corner of the <a href="#">Popup</a> .	Aligns to the right edge.

PlacementMode	Top Edge	Bottom Edge	Left Edge	Right Edge
Mouse	Aligns to the top edge.	The target origin changes to the top-left corner of the target area (the bounds of the mouse pointer) and the popup alignment point changes to the bottom-left corner of the <a href="#">Popup</a> .	Aligns to the left edge.	Aligns to the right edge.
MousePoint	Aligns to the top edge.	The popup alignment point changes to the bottom-left corner of the <a href="#">Popup</a> .	Aligns to the left edge.	The popup alignment point changes to the top-right corner of the popup.
Relative	Aligns to the top edge.	Aligns to the bottom edge.	Aligns to the left edge.	Aligns to the right edge.
RelativePoint	Aligns to the top edge.	The popup alignment point changes to the bottom-left corner of the <a href="#">Popup</a> .	Aligns to the left edge.	The popup alignment point changes to the top-right corner of the popup.
Right	Aligns to the top edge.	Aligns to the bottom edge.	Aligns to the left edge.	The target origin changes to the top-left corner of the target area and the popup alignment point changes to the top-right corner of the <a href="#">Popup</a> .
Top	The target origin changes to the bottom-left corner of the target area and the popup alignment point changes to the top-left corner of the <a href="#">Popup</a> . In effect, this is the same as when <a href="#">Placement</a> is Bottom.	Aligns to the bottom edge.	Aligns to the left edge.	Aligns to the right edge.

## Aligning to the Screen Edge

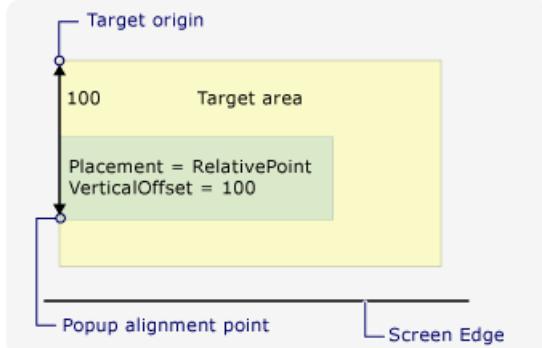
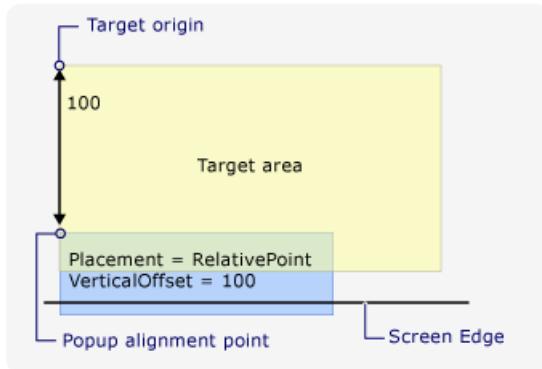
A [Popup](#) can align to the edge of the screen by repositioning itself so the entire [Popup](#) is visible on the screen. When this occurs, the distance between the target origin and popup alignment point might differ from the values of [HorizontalOffset](#) and [VerticalOffset](#). When [Placement](#) is [Absolute](#), [Center](#), or [Relative](#), the [Popup](#) aligns itself to every screen edge. For example, assume that a [Popup](#) has [Placement](#) set to [Relative](#) and [VerticalOffset](#) set to 100. If the bottom edge of the screen hides all or part of the [Popup](#), the [Popup](#) repositions itself along the bottom edge of the screen and the vertical distance between the target origin and popup alignment point is less than 100. The following illustration demonstrates this.



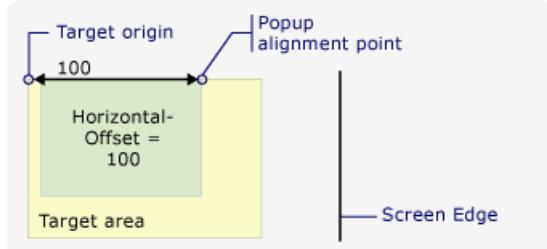
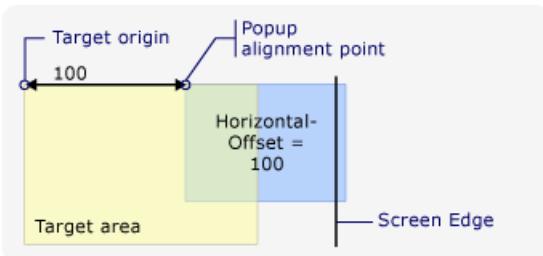
### Changing the Popup Alignment Point

If **Placement** is `AbsolutePoint`, `RelativePoint`, or `MousePoint`, the popup alignment point changes when the popup encounters the bottom or right screen edge.

The following illustration demonstrates that when the bottom screen edge hides all or part of the **Popup**, the **popup alignment point** is the bottom-left corner of the **Popup**.



The following illustration demonstrates that when the **Popup** is hidden by the right screen edge, the **popup alignment point** is the top-right corner of the **Popup**.

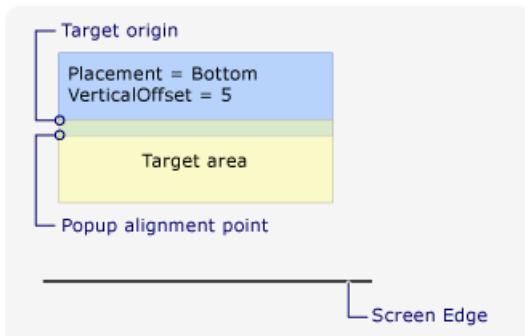
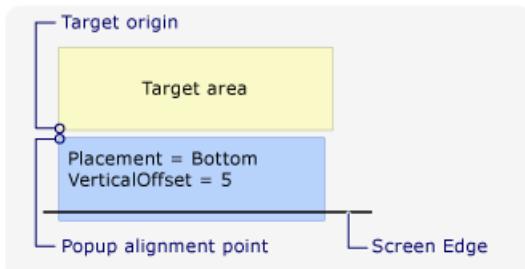


If the **Popup** encounters the bottom and right screen edges, the **popup alignment point** is the bottom-right corner of the **Popup**.

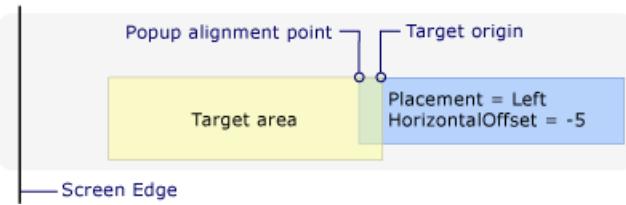
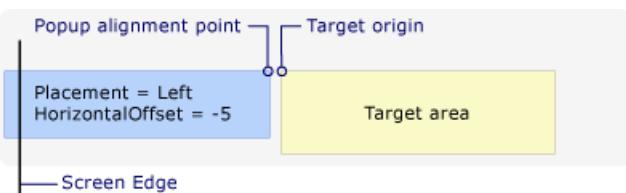
### Changing the Target Origin and Popup Alignment Point

When **Placement** is **Bottom**, **Left**, **Mouse**, **Right**, or **Top**, the target origin and **popup alignment point** change if a certain screen edge is encountered. The screen edge that causes the position to change depends on the **PlacementMode** value.

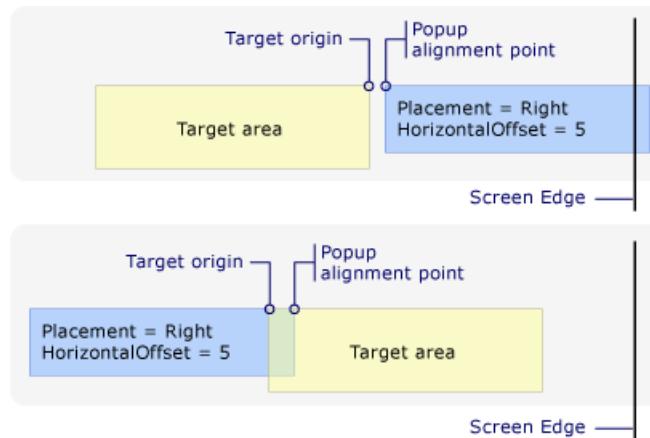
The following illustration demonstrates that when **Placement** is **Bottom** and the **Popup** encounters the bottom screen edge, the target origin is the top-left corner of the target area and the **popup alignment point** is the bottom-left corner of the **Popup**.



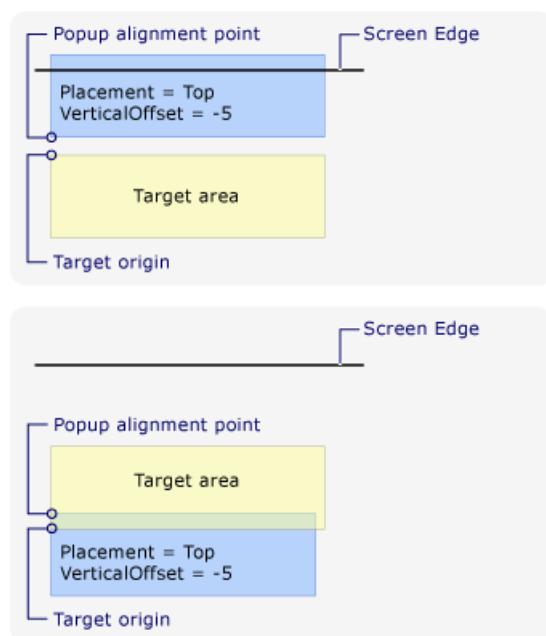
The following illustration demonstrates that when **Placement** is **Left** and the **Popup** encounters the left screen edge, the target origin is the top-right corner of the target area and the **popup alignment point** is the top-left corner of the **Popup**.



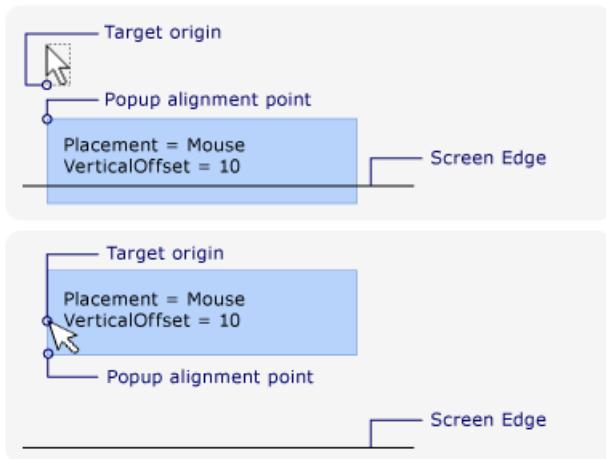
The following illustration demonstrates that when **Placement** is **Right** and the **Popup** encounters the right screen edge, the target origin is the top-left corner of the target area and the popup alignment point is the top-right corner of the **Popup**.



The following illustration demonstrates that when **Placement** is **Top** and the **Popup** encounters the top screen edge, the target origin is the bottom-left corner of the target area and the popup alignment point is the top-left corner of the **Popup**.



The following illustration demonstrates that when **Placement** is **Mouse** and the **Popup** encounters the bottom screen edge, the target origin is the top-left corner of the target area (the bounds of the mouse pointer) and the popup alignment point is the bottom-left corner of the **Popup**.



## Customizing Popup Placement

You can customize the target origin and popup alignment point by setting the `Placement` property to `Custom`. Then define a `CustomPopupPlacementCallback` delegate that returns a set of possible placement points and primary axes (in order of preference) for the `Popup`. The point that shows the largest portion of the `Popup` is selected. The position of the `Popup` is automatically adjusted if the `Popup` is hidden by the edge of the screen. For an example, see [Specify a Custom Popup Position](#).

## See also

- [Popup Placement Sample](#)

# Popup How-to Topics

2 minutes to read • [Edit Online](#)

The topics in this section describe how to use the [Popup](#) control to display content in a separate window that floats over the current application window.

## In This Section

[Animate a Popup](#)

[Specify a Custom Popup Position](#)

## Reference

[Popup](#)

## Related Sections

[Popup Overview](#)

# How to: Animate a Popup

2 minutes to read • [Edit Online](#)

This example shows two ways to animate a [Popup](#) control.

## Example

The following example sets the [PopupAnimation](#) property to a value of [Slide](#), which causes the [Popup](#) to "slide-in" when it appears.

In order to rotate the [Popup](#), this example assigns a [RotateTransform](#) to the [RenderTransform](#) property on the [Canvas](#), which is the child element of the [Popup](#).

For the transform to work correctly, the example must set the [AllowsTransparency](#) property to `true`. In addition, the [Margin](#) on the [Canvas](#) content must specify enough space for the [Popup](#) to rotate.

```
<Popup IsOpen="{Binding ElementName=myCheckBox, Path=IsChecked}"  
       PlacementTarget="{Binding ElementName=myCheckBox}"  
       AllowsTransparency="True"  
       PopupAnimation="Slide"  
       HorizontalOffset="50"  
       VerticalOffset="50"  
       >  
    <!--The Margin set on the Canvas provides the additional  
        area around the Popup so that the Popup is visible when  
        it rotates.-->  
    <Canvas Width="100" Height="100" Background="DarkBlue"  
           Margin="150">  
        <Canvas.RenderTransform>  
            <RotateTransform x:Name="theTransform" />  
        </Canvas.RenderTransform>  
        <TextBlock TextWrapping="Wrap" Foreground="White">  
            Rotating Popup  
        </TextBlock>  
    </Canvas>  
</Popup>
```

The following example shows how a [Click](#) event, which occurs when a [Button](#) is clicked, triggers the [Storyboard](#) that starts the animation.

```
<Button HorizontalAlignment="Left" Width="200" Margin="20,10,0,0">  
    <Button.Triggers>  
        <EventTrigger RoutedEvent="Button.Click">  
            <BeginStoryboard>  
                <Storyboard>  
                    <DoubleAnimation  
                        Storyboard.TargetName="theTransform"  
                        Storyboard.TargetProperty="(RotateTransform.Angle)"  
                        From="0" To="360" Duration="0:0:5" AutoReverse="True"/>  
                </Storyboard>  
            </BeginStoryboard>  
        </EventTrigger>  
    </Button.Triggers>  
    Click to see the Popup animate  
</Button>
```

## See also

- [RenderTransform](#)
- [BulletDecorator](#)
- [RotateTransform](#)
- [Storyboard](#)
- [Popup](#)
- [How-to Topics](#)
- [Popup Overview](#)

# How to: Specify a Custom Popup Position

2 minutes to read • [Edit Online](#)

This example shows how to specify a custom position for a [Popup](#) control when the [Placement](#) property is set to [Custom](#).

## Example

When the [Placement](#) property is set to [Custom](#), the [Popup](#) calls a defined instance of the [CustomPopupPlacementCallback](#) delegate. This delegate returns a set of possible points that are relative to the top left corner of the target area and the top left corner of the [Popup](#). The [Popup](#) placement occurs at the point that provides the best visibility.

The following example shows how to define the position of a [Popup](#) by setting the [Placement](#) property to [Custom](#). It also shows how to create and assign a [CustomPopupPlacementCallback](#) delegate in order to position the [Popup](#). The callback delegate returns two [CustomPopupPlacement](#) objects. If the [Popup](#) is hidden by a screen edge at the first position, the [Popup](#) is placed at the second position.

```
<Popup Name="popup1"
       PlacementTarget ="{Binding ElementName=myButton}"
       Placement="Custom">
    <TextBlock Height="60" Width="200"
              Background="LightGray"
              TextWrapping="Wrap">Popup positioned by using
              CustomPopupPlacement callback delegate</TextBlock>
</Popup>
```

```
public CustomPopupPlacement[] placePopup(Size popupSize,
                                         Size targetSize,
                                         Point offset)
{
    CustomPopupPlacement placement1 =
        new CustomPopupPlacement(new Point(-50, 100), PopupPrimaryAxis.Vertical);

    CustomPopupPlacement placement2 =
        new CustomPopupPlacement(new Point(10, 20), PopupPrimaryAxis.Horizontal);

    CustomPopupPlacement[] ttplaces =
        new CustomPopupPlacement[] { placement1, placement2 };
    return ttplaces;
}
```

```
Public Function placePopup(ByVal popupSize As Size, ByVal targetSize As Size, ByVal offset As Point) As
CustomPopupPlacement()
    Dim placement1 As New CustomPopupPlacement(New Point(-50, 100), PopupPrimaryAxis.Vertical)

    Dim placement2 As New CustomPopupPlacement(New Point(10, 20), PopupPrimaryAxis.Horizontal)

    Dim ttplaces() As CustomPopupPlacement = { placement1, placement2 }
    Return ttplaces
End Function
```

```
popup1.CustomPopupPlacementCallback =  
    new CustomPopupPlacementCallback(placePopup);
```

```
popup1.CustomPopupPlacementCallback = New CustomPopupPlacementCallback(AddressOf placePopup)
```

For the complete sample, see [Popup Placement Sample](#).

## See also

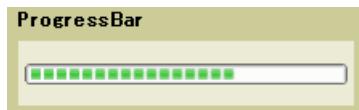
- [Popup](#)
- [Popup Overview](#)
- [How-to Topics](#)

# ProgressBar

2 minutes to read • [Edit Online](#)

A [ProgressBar](#) indicates the progress of an operation. The [ProgressBar](#) control consists of a window that is filled with the system highlight color as an operation progresses.

The following illustration shows a typical [ProgressBar](#).



## In This Section

## Reference

[ProgressBar](#)

[StatusBar](#)

## Related Sections

# PrintDialog

2 minutes to read • [Edit Online](#)

The [PrintDialog](#) control is used to instantiate a standard print dialog box that automatically configures a [PrintTicket](#) and [PrintQueue](#) according to user input.

## Reference

[PrintDialog](#)

[PrintTicket](#)

[PrintQueue](#)

## See also

- [Printing Overview](#)
- [Documents in WPF](#)

# RadioButton

2 minutes to read • [Edit Online](#)

**RadioButton** controls are usually grouped together to offer users a single choice among several options; only one button at a time can be selected.

The following illustration shows an example of a **RadioButton** control.



Typical RadioButton

## Reference

[ToggleButton](#)

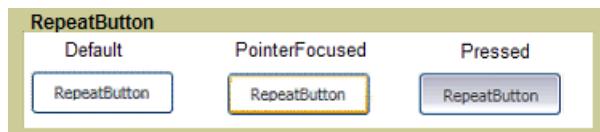
## Related Sections

# RepeatButton

2 minutes to read • [Edit Online](#)

The [RepeatButton](#) is similar to a [Button](#). However, [RepeatButton](#) elements give you control over when and how the [Click](#) event occurs.

The following graphic shows an example of the three states of a repeat button control, Default, PointerFocused, and Pressed. The first button shows the default state of the [RepeatButton](#). The second shows how the appearance of the button changes when the mouse pointer hovers over the button, giving it focus. The last button shows the appearance of the [RepeatButton](#) when the user presses the mouse button over the control.



Typical RepeatButton

## In This Section

### Reference

[RepeatButton](#)

### Related Sections

# RichTextBox

2 minutes to read • [Edit Online](#)

The [RichTextBox](#) element defines an editing control with built-in support for features such as cut and paste, rich document presentation, and content selection.

## In This Section

[RichTextBox Overview](#)

[How-to Topics](#)

## See also

- [TextBox](#)
- [Documents in WPF](#)
- [Flow Document Overview](#)

# RichTextBox Overview

8 minutes to read • [Edit Online](#)

The [RichTextBox](#) control enables you to display or edit flow content including paragraphs, images, tables, and more. This topic introduces the [TextBox](#) class and provides examples of how to use it in both Extensible Application Markup Language (XAML) and C#.

## TextBox or RichTextBox?

Both [RichTextBox](#) and [TextBox](#) allow users to edit text, however, the two controls are used in different scenarios. A [RichTextBox](#) is a better choice when it is necessary for the user to edit formatted text, images, tables, or other rich content. For example, editing a document, article, or blog that requires formatting, images, etc is best accomplished using a [RichTextBox](#). A [TextBox](#) requires less system resources than a [RichTextBox](#) and it is ideal when only plain text needs to be edited (i.e. usage in forms). See [TextBox Overview](#) for more information on [TextBox](#). The table below summarizes the main features of [TextBox](#) and [RichTextBox](#).

CONTROL	REAL-TIME SPELLCHECKING	CONTEXT MENU	FORMATTING COMMANDS LIKE <a href="#">TOGGLEBOLD</a> (CTR+B)	FLOWDOCUMENT CONTENT LIKE IMAGES, PARAGRAPHS, TABLES, ETC.
TextBox	Yes	Yes	No	No.
RichTextBox	Yes	Yes	Yes	Yes

### NOTE

Although [TextBox](#) does not support formatting related commands like [ToggleBold](#) (Ctr+B), many basic commands are supported by both controls such as [MoveToLineEnd](#).

The features from the table above are covered in more detail later.

## Creating a RichTextBox

The code below shows how to create a [RichTextBox](#) that a user can edit rich content in.

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <!-- A RichTextBox with no initial content in it. -->
    <RichTextBox />

</Page>
```

Specifically, the content edited in a [RichTextBox](#) is flow content. Flow content can contain many types of elements including formatted text, images, lists, and tables. See [Flow Document Overview](#) for in depth information on flow documents. In order to contain flow content, a [RichTextBox](#) hosts a [FlowDocument](#) object which in turn contains the editable content. To demonstrate flow content in a [RichTextBox](#), the following code shows how to create a [RichTextBox](#) with a paragraph and some bolded text.

```

<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <StackPanel>
        <RichTextBox>
            <FlowDocument>
                <Paragraph>
                    This is flow content and you can <Bold>edit me!</Bold>
                </Paragraph>
            </FlowDocument>
        </RichTextBox>
    </StackPanel>

</Page>

```

```

using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
using System.Windows.Documents;
namespace SDKSample
{
    public partial class BasicRichTextBoxWithContentExample : Page
    {
        public BasicRichTextBoxWithContentExample()
        {
            StackPanel myStackPanel = new StackPanel();

            // Create a FlowDocument to contain content for the RichTextBox.
            FlowDocument myFlowDoc = new FlowDocument();

            // Create a Run of plain text and some bold text.
            Run myRun = new Run("This is flow content and you can ");
            Bold myBold = new Bold(new Run("edit me!"));

            // Create a paragraph and add the Run and Bold to it.
            Paragraph myParagraph = new Paragraph();
            myParagraph.Inlines.Add(myRun);
            myParagraph.Inlines.Add(myBold);

            // Add the paragraph to the FlowDocument.
            myFlowDoc.Blocks.Add(myParagraph);

            RichTextBox myRichTextBox = new RichTextBox();

            // Add initial content to the RichTextBox.
            myRichTextBox.Document = myFlowDoc;

            myStackPanel.Children.Add(myRichTextBox);
            this.Content = myStackPanel;
        }
    }
}

```

```

Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Media
Imports System.Windows.Documents
Namespace SDKSample
    Partial Public Class BasicRichTextBoxWithContentExample
        Inherits Page
        Public Sub New()
            Dim myStackPanel As New StackPanel()

            ' Create a FlowDocument to contain content for the RichTextBox.
            Dim myFlowDoc As New FlowDocument()

            ' Create a Run of plain text and some bold text.
            Dim myRun As New Run("This is flow content and you can ")
            Dim myBold As New Bold(New Run("edit me!"))

            ' Create a paragraph and add the Run and Bold to it.
            Dim myParagraph As New Paragraph()
            myParagraph.Inlines.Add(myRun)
            myParagraph.Inlines.Add(myBold)

            ' Add the paragraph to the FlowDocument.
            myFlowDoc.Blocks.Add(myParagraph)

            Dim myRichTextBox As New RichTextBox()

            ' Add initial content to the RichTextBox.
            myRichTextBox.Document = myFlowDoc

            myStackPanel.Children.Add(myRichTextBox)
            Me.Content = myStackPanel
        End Sub
    End Class
End Namespace

```

The following illustration shows how this sample renders.

This is flow content and you can **edit me!**

Elements like **Paragraph** and **Bold** determine how the content inside a **RichTextBox** appears. As a user edits **RichTextBox** content, they change this flow content. To learn more about the features of flow content and how to work with it, see [Flow Document Overview](#).

#### NOTE

Flow content inside a **RichTextBox** does not behave exactly like flow content contained in other controls. For example, there are no columns in a **RichTextBox** and hence no automatic resizing behavior. Also, built in features like search, viewing mode, page navigation, and zoom are not available within a **RichTextBox**.

## Real-time Spell Checking

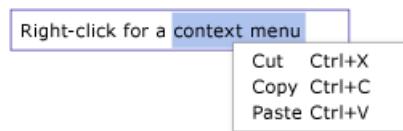
You can enable real-time spell checking in a **TextBox** or **RichTextBox**. When spellchecking is turned on, a red line appears underneath any misspelled words (see picture below).

The following word is Missspelled.|

See [Enable Spell Checking in a Text Editing Control](#) to learn how to enable spellchecking.

## Context Menu

By default, both `TextBox` and `RichTextBox` have a context menu that appears when a user right-clicks inside the control. The context menu allows the user to cut, copy, or paste (see illustration below).



You can create your own custom context menu to override the default one. See [Position a Custom Context Menu in a RichTextBox](#) for more information.

## Editing Commands

Editing commands enable users to format editable content inside a `RichTextBox`. Besides basic editing commands, `RichTextBox` includes formatting commands that `TextBox` does not support. For example, when editing in a `RichTextBox`, a user could press `Ctr+B` to toggle bold text formatting. See [EditingCommands](#) for a complete list of commands available. In addition to using keyboard shortcuts, you can hook commands up to other controls like buttons. The following example shows how to create a simple tool bar containing buttons that the user can use to change text formatting.

```
<Window x:Class="RichTextBoxInputPanelDemo.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Height="400" Width="600"
    >
<Grid>

    <!-- Set the styles for the tool bar. -->
    <Grid.Resources>
        <Style TargetType="{x:Type Button}" x:Key="formatTextStyle">
            <Setter Property="FontFamily" Value="Palatino Linotype"></Setter>
            <Setter Property="Width" Value="30"></Setter>
            <Setter Property="FontSize" Value = "14"></Setter>
            <Setter Property="CommandTarget" Value="{Binding ElementName=mainRTB}"></Setter>
        </Style>

        <Style TargetType="{x:Type Button}" x:Key="formatImageStyle">
            <Setter Property="Width" Value="30"></Setter>
            <Setter Property="CommandTarget" Value="{Binding ElementName=mainRTB}"></Setter>
        </Style>
    </Grid.Resources>

    <DockPanel Name="mainPanel">

        <!-- This tool bar contains all the editing buttons. -->
        <ToolBar Name="mainToolBar" Height="30" DockPanel.Dock="Top">

            <Button Style="{StaticResource formatImageStyle}" Command="ApplicationCommands.Cut" ToolTip="Cut">
                <Image Source="Images\EditCut.png"></Image>
            </Button>
            <Button Style="{StaticResource formatImageStyle}" Command="ApplicationCommands.Copy" ToolTip="Copy">
                <Image Source="Images\EditCopy.png"></Image>
            </Button>
            <Button Style="{StaticResource formatImageStyle}" Command="ApplicationCommands.Paste" ToolTip="Paste">
                <Image Source="Images\EditPaste.png"></Image>
            </Button>
            <Button Style="{StaticResource formatImageStyle}" Command="ApplicationCommands.Undo" ToolTip="Undo">
                <Image Source="Images\EditUndo.png"></Image>
            </Button>
            <Button Style="{StaticResource formatImageStyle}" Command="ApplicationCommands.Redo" ToolTip="Redo">
                <Image Source="Images\EditRedo.png"></Image>
            </Button>
        </ToolBar>
    </DockPanel>
</Window>
```

```

        </Button>
    
```

```

        <Button Style="{StaticResource formatTextStyle}" Command="EditingCommands.ToggleBold"
    ToolTip="Bold">
            <TextBlock FontWeight="Bold">B</TextBlock>
        </Button>
        <Button Style="{StaticResource formatTextStyle}" Command="EditingCommands.ToggleItalic"
    ToolTip="Italic">
            <TextBlock FontStyle="Italic" FontWeight="Bold">I</TextBlock>
        </Button>
        <Button Style="{StaticResource formatTextStyle}" Command="EditingCommands.ToggleUnderline"
    ToolTip="Underline">
            <TextBlock TextDecorations="Underline" FontWeight="Bold">U</TextBlock>
        </Button>
        <Button Style="{StaticResource formatImageStyle}" Command="EditingCommands.IncreaseFontSize"
    ToolTip="Grow Font">
            <Image Source="Images\CharacterGrowFont.png"></Image>
        </Button>
        <Button Style="{StaticResource formatImageStyle}" Command="EditingCommands.DecreaseFontSize"
    ToolTip="Shrink Font">
            <Image Source="Images\CharacterShrinkFont.png"></Image>
        </Button>

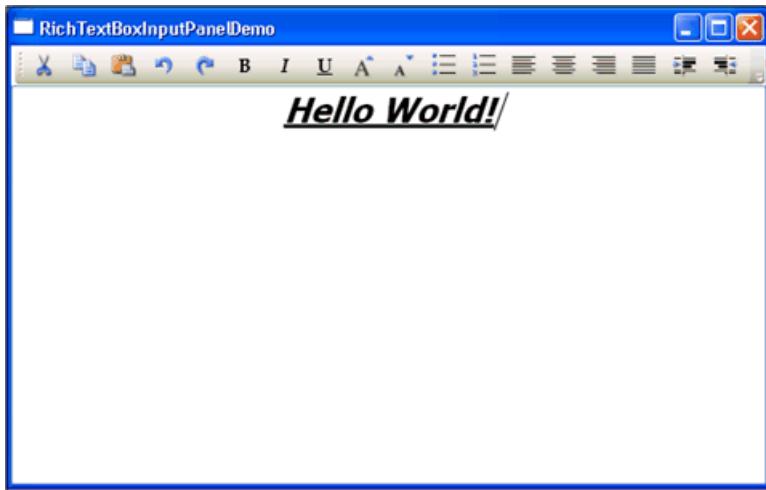
        <Button Style="{StaticResource formatImageStyle}" Command="EditingCommands.ToggleBullets"
    ToolTip="Bullets">
            <Image Source="Images\ListBullets.png"></Image>
        </Button>
        <Button Style="{StaticResource formatImageStyle}" Command="EditingCommands.ToggleNumbering"
    ToolTip="Numbering">
            <Image Source="Images/ListNumbering.png"></Image>
        </Button>
        <Button Style="{StaticResource formatImageStyle}" Command="EditingCommands.AlignLeft" ToolTip="Align
Left">
            <Image Source="Images\ParagraphLeftJustify.png"></Image>
        </Button>
        <Button Style="{StaticResource formatImageStyle}" Command="EditingCommands.AlignCenter"
    ToolTip="Align Center">
            <Image Source="Images\ParagraphCenterJustify.png"></Image>
        </Button>
        <Button Style="{StaticResource formatImageStyle}" Command="EditingCommands.AlignRight"
    ToolTip="Align Right">
            <Image Source="Images\ParagraphRightJustify.png"></Image>
        </Button>
        <Button Style="{StaticResource formatImageStyle}" Command="EditingCommands.AlignJustify"
    ToolTip="Align Justify">
            <Image Source="Images\ParagraphFullJustify.png"></Image>
        </Button>
        <Button Style="{StaticResource formatImageStyle}" Command="EditingCommands.IncreaseIndentation"
    ToolTip="Increase Indent">
            <Image Source="Images\ParagraphIncreaseIndentation.png"></Image>
        </Button>
        <Button Style="{StaticResource formatImageStyle}" Command="EditingCommands.DecreaseIndentation"
    ToolTip="Decrease Indent">
            <Image Source="Images\ParagraphDecreaseIndentation.png"></Image>
        </Button>

    </ToolBar>

    <!-- By default pressing tab moves focus to the next control. Setting AcceptsTab to true allows the
RichTextBox to accept tab characters. -->
    <RichTextBox Name="mainRTB" AcceptsTab="True"></RichTextBox>
</DockPanel>
</Grid>
</Window>

```

The following illustration shows how this sample displays.



## Detect when Content Changes

Usually the [TextChanged](#) event should be used to detect whenever the text in a [TextBox](#) or [RichTextBox](#) changes rather than [KeyDown](#) as you might expect. See [Detect When Text in a TextBox Has Changed](#) for an example.

## Save, Load, and Print RichTextBox Content

The following example shows how to save content of a [RichTextBox](#) to a file, load that content back into the [RichTextBox](#), and print the contents. Below is the markup for the example.

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      x:Class="SDKSample.SaveLoadPrintRTB" >

    <StackPanel>
        <RichTextBox Name="richTB">
            <FlowDocument>
                <Paragraph>
                    <Run>Paragraph 1</Run>
                </Paragraph>
            </FlowDocument>
        </RichTextBox>

        <Button Click="SaveRTBContent">Save RTB Content</Button>
        <Button Click="LoadRTBContent">Load RTB Content</Button>
        <Button Click="PrintRTBContent">Print RTB Content</Button>
    </StackPanel>

</Page>
```

Below is the code behind for the example.

```
using System;
using System.IO;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Media;

namespace SDKSample
{
    public partial class SaveLoadPrintRTB : Page
    {
        // Handle "Save RichTextBox Content" button click.
```

```

void SaveRTBContent(Object sender, RoutedEventArgs args)
{
    // Send an arbitrary URL and file name string specifying
    // the location to save the XAML in.
    SaveXamlPackage("C:\\test.xaml");
}

// Handle "Load RichTextBox Content" button click.
void LoadRTBContent(Object sender, RoutedEventArgs args)
{
    // Send URL string specifying what file to retrieve XAML
    // from to load into the RichTextBox.
    LoadXamlPackage("C:\\test.xaml");
}

// Handle "Print RichTextBox Content" button click.
void PrintRTBContent(Object sender, RoutedEventArgs args)
{
    PrintCommand();
}

// Save XAML in RichTextBox to a file specified by _fileName
void SaveXamlPackage(string _fileName)
{
    TextRange range;
    FileStream fStream;
    range = new TextRange(richTB.Document.ContentStart, richTB.Document.ContentEnd);
    fStream = new FileStream(_fileName, FileMode.Create);
    range.Save(fStream, DataFormats.XamlPackage);
    fStream.Close();
}

// Load XAML into RichTextBox from a file specified by _fileName
void LoadXamlPackage(string _fileName)
{
    TextRange range;
    FileStream fStream;
    if (File.Exists(_fileName))
    {
        range = new TextRange(richTB.Document.ContentStart, richTB.Document.ContentEnd);
        fStream = new FileStream(_fileName, FileMode.OpenOrCreate);
        range.Load(fStream, DataFormats.XamlPackage);
        fStream.Close();
    }
}

// Print RichTextBox content
private void PrintCommand()
{
    PrintDialog pd = new PrintDialog();
    if ((pd.ShowDialog() == true))
    {
        //use either one of the below
        pd.PrintVisual(richTB as Visual, "printing as visual");
        pd.PrintDocument(((IDocumentPaginatorSource)richTB.Document).DocumentPaginator, "printing
as paginator");
    }
}
}

```

```

Imports System.IO
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Documents
Imports System.Windows.Media

Namespace SDKSample

    Partial Public Class SaveLoadPrintRTB
        Inherits Page

        ' Handle "Save RichTextBox Content" button click.
        Private Sub SaveRTBContent(ByVal sender As Object, ByVal args As RoutedEventArgs)

            ' Send an arbitrary URL and file name string specifying
            ' the location to save the XAML in.
            SaveXamlPackage("C:\test.xaml")
        End Sub

        ' Handle "Load RichTextBox Content" button click.
        Private Sub LoadRTBContent(ByVal sender As Object, ByVal args As RoutedEventArgs)
            ' Send URL string specifying what file to retrieve XAML
            ' from to load into the RichTextBox.
            LoadXamlPackage("C:\test.xaml")
        End Sub

        ' Handle "Print RichTextBox Content" button click.
        Private Sub PrintRTBContent(ByVal sender As Object, ByVal args As RoutedEventArgs)
            PrintCommand()
        End Sub

        ' Save XAML in RichTextBox to a file specified by _fileName
        Private Sub SaveXamlPackage(ByVal _fileName As String)
            Dim range As TextRange
            Dim fStream As FileStream
            range = New TextRange(richTB.Document.ContentStart, richTB.Document.ContentEnd)
            fStream = New FileStream(_fileName, FileMode.Create)
            range.Save(fStream, DataFormats.XamlPackage)
            fStream.Close()
        End Sub

        ' Load XAML into RichTextBox from a file specified by _fileName
        Private Sub LoadXamlPackage(ByVal _fileName As String)
            Dim range As TextRange
            Dim fStream As FileStream
            If File.Exists(_fileName) Then
                range = New TextRange(richTB.Document.ContentStart, richTB.Document.ContentEnd)
                fStream = New FileStream(_fileName, FileMode.OpenOrCreate)
                range.Load(fStream, DataFormats.XamlPackage)
                fStream.Close()
            End If
        End Sub

        ' Print RichTextBox content
        Private Sub PrintCommand()
            Dim pd As New PrintDialog()
            If (pd.ShowDialog() = True) Then
                'use either one of the below
                pd.PrintVisual(TryCast(richTB, Visual), "printing as visual")
                pd.PrintDocument(((CType(richTB.Document, IDocumentPaginatorSource)).DocumentPaginator),
                "printing as paginator")
            End If
        End Sub
    End Class
End Namespace

```

## See also

- [How-to Topics](#)
- [TextBox Overview](#)

# RichTextBox How-to Topics

2 minutes to read • [Edit Online](#)

This section provides examples that demonstrate how to accomplish common tasks using the [RichTextBox](#) control.

## In This Section

- [Extract the Text Content from a RichTextBox](#)
- [Change Selection in a RichTextBox Programmatically](#)
- [Save, Load, and Print RichTextBox Content](#)
- [Position a Custom Context Menu in a RichTextBox](#)

## See also

- [TextBox](#)
- [Documents in WPF](#)
- [Flow Document Overview](#)

# How to: Extract the Text Content from a RichTextBox

2 minutes to read • [Edit Online](#)

This example shows how to extract the contents of a [RichTextBox](#) as plain text.

## Example

The following Extensible Application Markup Language (XAML) code describes a named [RichTextBox](#) control with simple content.

```
<RichTextBox Name="richTB">
  <FlowDocument>
    <Paragraph>
      <Run>Paragraph 1</Run>
    </Paragraph>
    <Paragraph>
      <Run>Paragraph 2</Run>
    </Paragraph>
    <Paragraph>
      <Run>Paragraph 3</Run>
    </Paragraph>
  </FlowDocument>
</RichTextBox>
```

## Example

The following code implements a method that takes a [RichTextBox](#) as an argument, and returns a string representing the plain text contents of the [RichTextBox](#).

The method creates a new [TextRange](#) from the contents of the [RichTextBox](#), using the [ContentStart](#) and [ContentEnd](#) to indicate the range of the contents to extract. [ContentStart](#) and [ContentEnd](#) properties each return a [TextPointer](#), and are accessible on the underlying FlowDocument that represents the contents of the [RichTextBox](#). [TextRange](#) provides a [Text](#) property, which returns the plain text portions of the [TextRange](#) as a string.

```
string StringFromRichTextBox(RichTextBox rtb)
{
  TextRange textRange = new TextRange(
    // TextPointer to the start of content in the RichTextBox.
    rtb.Document.ContentStart,
    // TextPointer to the end of content in the RichTextBox.
    rtb.Document.ContentEnd
  );

  // The Text property on a TextRange object returns a string
  // representing the plain text content of the TextRange.
  return textRange.Text;
}
```

```
Private Function StringFromRichTextBox(ByVal rtb As RichTextBox) As String
    ' TextPointer to the start of content in the RichTextBox.
    ' TextPointer to the end of content in the RichTextBox.
    Dim textRange As New TextRange(rtb.Document.ContentStart, rtb.Document.ContentEnd)

    ' The Text property on a TextRange object returns a string
    ' representing the plain text content of the TextRange.
    Return textRange.Text
End Function
```

## See also

- [RichTextBox Overview](#)
- [TextBox Overview](#)

# Change Selection in a RichTextBox Programmatically

2 minutes to read • [Edit Online](#)

This example shows how to programmatically change the current selection in a [RichTextBox](#). This selection is the same as if the user had selected the content by using the user interface.

## Example

The following Extensible Application Markup Language (XAML) code describes a named [RichTextBox](#) control with simple content.

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      x:Class="SDKSample.ChangeSelectionProgrammaticaly" >

    <StackPanel>
        <RichTextBox GotMouseCapture="ChangeSelection" Name="richTB">
            <FlowDocument>
                <Paragraph Name="myParagraph">
                    <Run>
                        When the user clicks in the RichTextBox, the selected
                        text changes programmatically.
                    </Run>
                </Paragraph>
            </FlowDocument>
        </RichTextBox>
    </StackPanel>

</Page>
```

## Example

The following code programmatically selects some arbitrary text when the user clicks inside the [RichTextBox](#).

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;

namespace SDKSample
{
    public partial class ChangeSelectionProgrammaticaly : Page
    {

        // Change the current selection.
        void ChangeSelection(Object sender, RoutedEventArgs args)
        {
            // Create two arbitrary TextPointers to specify the range of content to select.
            TextPointer myTextPointer1 = myParagraph.ContentStart.GetPositionAtOffset(20);
            TextPointer myTextPointer2 = myParagraph.ContentEnd.GetPositionAtOffset(-10);

            // Programmatically change the selection in the RichTextBox.
            richTB.Selection.Select(myTextPointer1, myTextPointer2);
        }
    }
}
```

```
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Documents

Namespace SDKSample
    Partial Public Class ChangeSelectionProgrammatically
        Inherits Page

        ' Change the current selection.
        Private Sub ChangeSelection(ByVal sender As Object, ByVal args As RoutedEventArgs)
            ' Create two arbitrary TextPointers to specify the range of content to select.
            Dim myTextPointer1 As TextPointer = myParagraph.ContentStart.GetPositionAtOffset(20)
            Dim myTextPointer2 As TextPointer = myParagraph.ContentEnd.GetPositionAtOffset(-10)

            ' Programmatically change the selection in the RichTextBox.
            richTB.Selection.Select(myTextPointer1, myTextPointer2)
        End Sub
    End Class
End Namespace
```

## See also

- [RichTextBox Overview](#)
- [TextBox Overview](#)

# How to: Save, Load, and Print RichTextBox Content

2 minutes to read • [Edit Online](#)

The following example shows how to save content of a [RichTextBox](#) to a file, load that content back into the [RichTextBox](#), and print the contents.

## Example

Below is the markup for the example.

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      x:Class="SDKSample.SaveLoadPrintRTB" >

    <StackPanel>
        <RichTextBox Name="richTB">
            <FlowDocument>
                <Paragraph>
                    <Run>Paragraph 1</Run>
                </Paragraph>
            </FlowDocument>
        </RichTextBox>

        <Button Click="SaveRTBContent">Save RTB Content</Button>
        <Button Click="LoadRTBContent">Load RTB Content</Button>
        <Button Click="PrintRTBContent">Print RTB Content</Button>
    </StackPanel>

</Page>
```

## Example

Below is the code behind for the example.

```
using System;
using System.IO;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Media;

namespace SDKSample
{

    public partial class SaveLoadPrintRTB : Page
    {

        // Handle "Save RichTextBox Content" button click.
        void SaveRTBContent(Object sender, RoutedEventArgs args)
        {

            // Send an arbitrary URL and file name string specifying
            // the location to save the XAML in.
            SaveXamlPackage("C:\\test.xaml");
        }

        // Handle "Load RichTextBox Content" button click.
        void LoadRTBContent(Object sender, RoutedEventArgs args)
    }
}
```

```

{
    // Send URL string specifying what file to retrieve XAML
    // from to load into the RichTextBox.
    LoadXamlPackage("C:\\test.xaml");
}

// Handle "Print RichTextBox Content" button click.
void PrintRTBContent(Object sender, RoutedEventArgs args)
{
    PrintCommand();
}

// Save XAML in RichTextBox to a file specified by _fileName
void SaveXamlPackage(string _fileName)
{
    TextRange range;
    FileStream fStream;
    range = new TextRange(richTB.Document.ContentStart, richTB.Document.ContentEnd);
    fStream = new FileStream(_fileName, FileMode.Create);
    range.Save(fStream, DataFormats.XamlPackage);
    fStream.Close();
}

// Load XAML into RichTextBox from a file specified by _fileName
void LoadXamlPackage(string _fileName)
{
    TextRange range;
    FileStream fStream;
    if (File.Exists(_fileName))
    {
        range = new TextRange(richTB.Document.ContentStart, richTB.Document.ContentEnd);
        fStream = new FileStream(_fileName, FileMode.OpenOrCreate);
        range.Load(fStream, DataFormats.XamlPackage);
        fStream.Close();
    }
}

// Print RichTextBox content
private void PrintCommand()
{
    PrintDialog pd = new PrintDialog();
    if ((pd.ShowDialog() == true))
    {
        //use either one of the below
        pd.PrintVisual(richTB as Visual, "printing as visual");
        pd.PrintDocument(((IDocumentPaginatorSource)richTB.Document).DocumentPaginator, "printing as
paginator");
    }
}
}

```

```

Imports System.IO
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Documents
Imports System.Windows.Media

Namespace SDKSample

    Partial Public Class SaveLoadPrintRTB
        Inherits Page

        ' Handle "Save RichTextBox Content" button click.
        Private Sub SaveRTBContent(ByVal sender As Object, ByVal args As RoutedEventArgs)

            ' Send an arbitrary URL and file name string specifying
            ' the location to save the XAML in.
            SaveXamlPackage("C:\test.xaml")
        End Sub

        ' Handle "Load RichTextBox Content" button click.
        Private Sub LoadRTBContent(ByVal sender As Object, ByVal args As RoutedEventArgs)
            ' Send URL string specifying what file to retrieve XAML
            ' from to load into the RichTextBox.
            LoadXamlPackage("C:\test.xaml")
        End Sub

        ' Handle "Print RichTextBox Content" button click.
        Private Sub PrintRTBContent(ByVal sender As Object, ByVal args As RoutedEventArgs)
            PrintCommand()
        End Sub

        ' Save XAML in RichTextBox to a file specified by _fileName
        Private Sub SaveXamlPackage(ByVal _fileName As String)
            Dim range As TextRange
            Dim fStream As FileStream
            range = New TextRange(richTB.Document.ContentStart, richTB.Document.ContentEnd)
            fStream = New FileStream(_fileName, FileMode.Create)
            range.Save(fStream, DataFormats.XamlPackage)
            fStream.Close()
        End Sub

        ' Load XAML into RichTextBox from a file specified by _fileName
        Private Sub LoadXamlPackage(ByVal _fileName As String)
            Dim range As TextRange
            Dim fStream As FileStream
            If File.Exists(_fileName) Then
                range = New TextRange(richTB.Document.ContentStart, richTB.Document.ContentEnd)
                fStream = New FileStream(_fileName, FileMode.OpenOrCreate)
                range.Load(fStream, DataFormats.XamlPackage)
                fStream.Close()
            End If
        End Sub

        ' Print RichTextBox content
        Private Sub PrintCommand()
            Dim pd As New PrintDialog()
            If (pd.ShowDialog() = True) Then
                'use either one of the below
                pd.PrintVisual(TryCast(richTB, Visual), "printing as visual")
                pd.PrintDocument(((CType(richTB.Document, IDocumentPaginatorSource)).DocumentPaginator),
                "printing as paginator")
            End If
        End Sub
    End Class
End Namespace

```

## See also

- [RichTextBox Overview](#)
- [TextBox Overview](#)

# How to: Position a Custom Context Menu in a RichTextBox

2 minutes to read • [Edit Online](#)

This example shows how to position a custom context menu for a **RichTextBox**.

When you implement a custom context menu for a **RichTextBox**, you are responsible for handling the placement of the context menu. By default, a custom context menu is opened at the center of the **RichTextBox**.

## Example

To override the default placement behavior, add a listener for the [ContextMenuOpening](#) event. The following example shows how to do this programmatically.

```
richTextBox.ContextMenuOpening += new ContextMenuEventHandler(richTextBox_ContextMenuOpening);
```

```
AddHandler richTextBox.ContextMenuOpening, AddressOf richTextBox_ContextMenuOpening
```

## Example

The following example shows an implementation the corresponding [ContextMenuOpening](#) event listener.

```
// This method is intended to listen for the ContextMenuOpening event from a RichTextBox.  
// It will position the custom context menu at the end of the current selection.  
void richTextBox_ContextMenuOpening(object sender, ContextMenuEventArgs e)  
{  
    // Sender must be RichTextBox.  
    RichTextBox rtb = sender as RichTextBox;  
    if (rtb == null) return;  
  
    ContextMenu contextMenu = rtb.ContextMenu;  
    contextMenu.PlacementTarget = rtb;  
  
    // This uses HorizontalOffset and VerticalOffset properties to position the menu,  
    // relative to the upper left corner of the parent element (RichTextBox in this case).  
    contextMenu.Placement = PlacementMode.RelativePoint;  
  
    // Compute horizontal and vertical offsets to place the menu relative to selection end.  
    TextPointer position = rtb.Selection.End;  
  
    if (position == null) return;  
  
    Rect positionRect = position.GetCharacterRect(LogicalDirection.Forward);  
    contextMenu.HorizontalOffset = positionRect.X;  
    contextMenu.VerticalOffset = positionRect.Y;  
  
    // Finally, mark the event has handled.  
    contextMenu.IsOpen = true;  
    e.Handled = true;  
}
```

```

' This method is intended to listen for the ContextMenuOpening event from a RichTextBox.
' It will position the custom context menu at the end of the current selection.
Private Sub richTextBox_ContextMenuOpening(ByVal sender As Object, ByVal e As ContextMenuEventArgs)
    ' Sender must be RichTextBox.
    Dim rtb As RichTextBox = TryCast(sender, RichTextBox)
    If rtb Is Nothing Then
        Return
    End If

    Dim contextMenu As ContextMenu = rtb.ContextMenu
    contextMenu.PlacementTarget = rtb

    ' This uses HorizontalOffset and VerticalOffset properties to position the menu,
    ' relative to the upper left corner of the parent element (RichTextBox in this case).
    contextMenu.Placement = PlacementMode.RelativePoint

    ' Compute horizontal and vertical offsets to place the menu relative to selection end.
    Dim position As TextPointer = rtb.Selection.End

    If position Is Nothing Then
        Return
    End If

    Dim positionRect As Rect = position.GetCharacterRect(LogicalDirection.Forward)
    contextMenu.HorizontalOffset = positionRect.X
    contextMenu.VerticalOffset = positionRect.Y

    ' Finally, mark the event has handled.
    contextMenu.IsOpen = True
    e.Handled = True
End Sub

```

## See also

- [RichTextBox Overview](#)
- [TextBox Overview](#)

# ScrollBar

2 minutes to read • [Edit Online](#)

A [ScrollBar](#) allows you to view content that is outside of the current viewing area by sliding the [Thumb](#) to make the content visible.

## In This Section

[Customize the Thumb Size on a ScrollBar](#)

## Reference

[ScrollBar](#)

[Track](#)

[Thumb](#)

[ScrollViewer](#)

# How to: Customize the Thumb Size on a ScrollBar

2 minutes to read • [Edit Online](#)

This topic explains how to set the [Thumb](#) of a [ScrollBar](#) to a fixed size and how to specify a minimum size for the [Thumb](#) of a [ScrollBar](#).

## Example

## Description

The following example creates a [ScrollBar](#) that has a [Thumb](#) with a fixed size. The example sets the [ViewPortSize](#) property of the [Thumb](#) to [NaN](#) and sets the height of the [Thumb](#). To create a horizontal [ScrollBar](#) with a [Thumb](#) that has a fixed width, set the width of the [Thumb](#).

## Code

```

<Style TargetType="ScrollBar">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="ScrollBar">
                <Grid Name="Bg"
                    Background="{TemplateBinding Background}"
                    SnapsToDevicePixels="true">
                    <Grid.RowDefinitions>
                        <RowDefinition MaxHeight="{DynamicResource {x:Static SystemParameters.VerticalScrollBarButtonHeightKey}}"/>
                        <RowDefinition Height="0.00001*"/>
                        <RowDefinition MaxHeight="{DynamicResource {x:Static SystemParameters.VerticalScrollBarButtonHeightKey}}"/>
                    </Grid.RowDefinitions>
                    <RepeatButton Style="{StaticResource ScrollBarButton}"
                        IsEnabled="{TemplateBinding IsMouseOver}"
                        Height="18"

                        Command="ScrollBar.LineUpCommand"
                        Content="M 0 4 L 8 4 L 4 0 Z" />
                    <!-- Set the ViewportSize to NaN to disable autosizing of the Thumb. -->
                    <Track Name="PART_Track"
                        ViewportSize="NaN"
                        IsDirectionReversed="true"
                        Grid.Row="1"
                        Grid.ZIndex="-1">
                        <Track.DecreaseRepeatButton>
                            <RepeatButton Style="{StaticResource VerticalScrollBarPageButton}"
                                Command="ScrollBar.PageUpCommand"/>
                        </Track.DecreaseRepeatButton>
                        <Track.IncreaseRepeatButton>
                            <RepeatButton Style="{StaticResource VerticalScrollBarPageButton}"
                                Command="ScrollBar.PageDownCommand"/>
                        </Track.IncreaseRepeatButton>
                    <Track.Thumb>
                        <!-- Set the height of the Thumb.-->
                        <Thumb Height="30"/>
                    </Track.Thumb>
                </Grid>
                <ControlTemplate.Triggers>
                    <Trigger SourceName="PART_Track" Property="IsEnabled" Value="false">
                        <Setter TargetName="PART_Track" Property="Visibility" Value="Hidden"/>
                    </Trigger>
                </ControlTemplate.Triggers>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

## Description

The following example creates a [ScrollBar](#) that has a [Thumb](#) with a minimum size. The example sets the value of [VerticalScrollBarButtonHeightKey](#). To create a horizontal [ScrollBar](#) with a [Thumb](#) that has a minimum width, set the [HorizontalScrollBarWidthKey](#).

# Code

```
<Style TargetType="ScrollBar">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="ScrollBar">
                <Grid Name="Bg"
                    Background="{TemplateBinding Background}"
                    SnapsToDevicePixels="true">
                    <Grid.RowDefinitions>
                        <RowDefinition MaxHeight="{DynamicResource {x:Static SystemParameters.VerticalScrollBarButtonHeightKey}}"/>
                        <RowDefinition Height="0.0001*"/>
                        <RowDefinition MaxHeight="{DynamicResource {x:Static SystemParameters.VerticalScrollBarButtonHeightKey}}"/>
                    </Grid.RowDefinitions>
                    <RepeatButton Style="{StaticResource ScrollBarButton}"
                        IsEnabled="{TemplateBinding IsMouseOver}"
                        Height="18"
                        Command="ScrollBar.LineUpCommand"
                        Content="M 0 4 L 8 4 L 4 0 Z" />
                    <Track Name="PART_Track"
                        IsDirectionReversed="true"
                        Grid.Row="1"
                        Grid.ZIndex="-1">
                        <Track.Resources>
                            <!-- Set the Thumb's minimum height to 50.
                            The Thumb's minimum height is half the
                            value of VerticalScrollBarButtonHeightKey. -->
                            <sys:Double
                                x:Key="{x:Static SystemParameters.VerticalScrollBarButtonHeightKey}">
                                100
                            </sys:Double>
                        </Track.Resources>
                        <Track.DecreaseRepeatButton>
                            <RepeatButton Style="{StaticResource VerticalScrollBarPageButton}"
                                Command="ScrollBar.PageUpCommand"/>
                        </Track.DecreaseRepeatButton>
                        <Track.IncreaseRepeatButton>
                            <RepeatButton Style="{StaticResource VerticalScrollBarPageButton}"
                                Command="ScrollBar.PageDownCommand"/>
                        </Track.IncreaseRepeatButton>
                        <Track.Thumb>
                            <Thumb/>
                        </Track.Thumb>
                    </Track>
                    <RepeatButton
                        Grid.Row="2"
                        Style="{StaticResource ScrollBarButton}"
                        Height="18"
                        Command="ScrollBar.LineDownCommand"
                        Content="M 0 0 L 4 4 L 8 0 Z"/>
                </Grid>
                <ControlTemplate.Triggers>
                    <Trigger SourceName="PART_Track"
                        Property="IsEnabled" Value="false">
                        <Setter TargetName="PART_Track"
                            Property="Visibility" Value="Hidden"/>
                    </Trigger>
                </ControlTemplate.Triggers>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

## See also

- [ScrollBar Styles and Templates](#)

# ScrollView

2 minutes to read • [Edit Online](#)

The [ScrollView](#) control creates a scrollable region wherein content can be scrolled horizontally or vertically.

## In This Section

[ScrollView Overview](#)

[How-to Topics](#)

## Reference

[ScrollBar](#)

[ScrollView](#)

## See also

- [Panels Overview](#)
- [Layout](#)

# ScrollViewer Overview

4 minutes to read • [Edit Online](#)

Content within a user interface is often larger than a computer screen's display area. The [ScrollViewer](#) control provides a convenient way to enable scrolling of content in Windows Presentation Foundation (WPF) applications. This topic introduces the [ScrollViewer](#) element and provides several usage examples.

## The ScrollViewer Control

There are two predefined elements that enable scrolling in WPF applications: [ScrollBar](#) and [ScrollViewer](#). The [ScrollViewer](#) control encapsulates horizontal and vertical [ScrollBar](#) elements and a content container (such as a [Panel](#) element) in order to display other visible elements in a scrollable area. You must build a custom object in order to use the [ScrollBar](#) element for content scrolling. However, you can use the [ScrollViewer](#) element by itself because it is a composite control that encapsulates [ScrollBar](#) functionality.

The [ScrollViewer](#) control responds to both mouse and keyboard commands, and defines numerous methods with which to scroll content by predetermined increments. You can use the [ScrollChanged](#) event to detect a change in a [ScrollViewer](#) state.

A [ScrollViewer](#) can only have one child, typically a [Panel](#) element that can host a [Children](#) collection of elements. The [Content](#) property defines the sole child of the [ScrollViewer](#).

## Physical vs. Logical Scrolling

Physical scrolling is used to scroll content by a predetermined physical increment, typically by a value that is declared in pixels. Logical scrolling is used to scroll to the next item in the logical tree. Physical scrolling is the default scroll behavior for most [Panel](#) elements. WPF supports both types of scrolling.

### The [IScrollInfo](#) Interface

The [IScrollInfo](#) interface represents the main scrolling region within a [ScrollViewer](#) or derived control. The interface defines scrolling properties and methods that can be implemented by [Panel](#) elements that require scrolling by logical unit, rather than by a physical increment. Casting an instance of [IScrollInfo](#) to a derived [Panel](#) and then using its scrolling methods provides a useful way to scroll to the next logical unit in a child collection, rather than by pixel increment. By default, the [ScrollViewer](#) control supports scrolling by physical units.

[StackPanel](#) and [VirtualizingStackPanel](#) both implement [IScrollInfo](#) and natively support logical scrolling. For layout controls that natively support logical scrolling, you can still achieve physical scrolling by wrapping the host [Panel](#) element in a [ScrollViewer](#) and setting the [CanContentScroll](#) property to `false`.

The following code example demonstrates how to cast an instance of [IScrollInfo](#) to a [StackPanel](#) and use content scrolling methods ([LineUp](#) and [LineDown](#)) defined by the interface.

```
private void spLineUp(object sender, RoutedEventArgs e)
{
    ((IScrollInfo)sp1).LineUp();
}

private void spLineDown(object sender, RoutedEventArgs e)
{
    ((IScrollInfo)sp1).LineDown();
}
```

```

Private Sub spLineUp(ByVal sender As Object, ByVal args As RoutedEventArgs)
    CType(sp1, IScrollInfo).LineUp()
End Sub
Private Sub spLineDown(ByVal sender As Object, ByVal args As RoutedEventArgs)
    CType(sp1, IScrollInfo).LineDown()
End Sub

```

## Defining and Using a ScrollViewer Element

The following example creates a [ScrollViewer](#) in a window that contains some text and a rectangle. [ScrollBar](#) elements appear only when they are necessary. When you resize the window, the [ScrollBar](#) elements appear and disappear, due to updated values of the [ComputedHorizontalScrollBarVisibility](#) and [ComputedVerticalScrollBarVisibility](#) properties.

```

// Create the application's main window
mainWindow = gcnew System::Windows::Window();
mainWindow->Title = "ScrollViewer Sample";

// Define a ScrollViewer
myScrollViewer = gcnew ScrollViewer();
myScrollViewer->HorizontalScrollBarVisibility = ScrollBarVisibility::Auto;

// Add Layout control
myStackPanel = gcnew StackPanel();
myStackPanel->HorizontalAlignment = HorizontalAlignment::Left;
myStackPanel->VerticalAlignment = VerticalAlignment::Top;

TextBlock^ myTextBlock = gcnew TextBlock();
myTextBlock->TextWrapping = TextWrapping::Wrap;
myTextBlock->Margin = System::Windows::Thickness(0, 0, 0, 20);
myTextBlock->Text = "Scrolling is enabled when it is necessary. Resize the Window, making it larger and smaller.";

Rectangle^ myRectangle = gcnew Rectangle();
myRectangle->Fill = Brushes::Red;
myRectangle->Width = 500;
myRectangle->Height = 500;

// Add child elements to the parent StackPanel
myStackPanel->Children->Add(myTextBlock);
myStackPanel->Children->Add(myRectangle);

// Add the StackPanel as the lone Child of the Border
myScrollViewer->Content = myStackPanel;

// Add the Border as the Content of the Parent Window Object
mainWindow->Content = myScrollViewer;
mainWindow->Show();

```

```

// Create the application's main window
mainWindow = new Window();
mainWindow.Title = "ScrollViewer Sample";

// Define a ScrollViewer
myScrollViewer = new ScrollViewer();
myScrollViewer.HorizontalScrollBarVisibility = ScrollBarVisibility.Auto;

// Add Layout control
myStackPanel = new StackPanel();
myStackPanel.HorizontalAlignment = HorizontalAlignment.Left;
myStackPanel.VerticalAlignment = VerticalAlignment.Top;

TextBlock myTextBlock = new TextBlock();
myTextBlock.TextWrapping = TextWrapping.Wrap;
myTextBlock.Margin = new Thickness(0, 0, 0, 20);
myTextBlock.Text = "Scrolling is enabled when it is necessary. Resize the Window, making it larger and smaller.";

Rectangle myRectangle = new Rectangle();
myRectangle.Fill = Brushes.Red;
myRectangle.Width = 500;
myRectangle.Height = 500;

// Add child elements to the parent StackPanel
myStackPanel.Children.Add(myTextBlock);
myStackPanel.Children.Add(myRectangle);

// Add the StackPanel as the lone Child of the Border
myScrollViewer.Content = myStackPanel;

// Add the Border as the Content of the Parent Window Object
mainWindow.Content = myScrollViewer;
mainWindow.Show ();

```

```

'Define a ScrollViewer.
Dim myScrollViewer As New ScrollViewer
myScrollViewer.HorizontalScrollBarVisibility = ScrollBarVisibility.Auto

'Add Layout control.
Dim myStackPanel As New StackPanel
myStackPanel.HorizontalAlignment = System.Windows.HorizontalAlignment.Left
myStackPanel.VerticalAlignment = System.Windows.VerticalAlignment.Top

Dim myTextBlock As New TextBlock
myTextBlock.TextWrapping = TextWrapping.Wrap
myTextBlock.Margin = New Thickness(0, 0, 0, 20)
myTextBlock.Text = "Scrolling is enabled when it is necessary. Resize the Window, making it larger and smaller."

Dim myRectangle As New Rectangle
myRectangle.Fill = Brushes.Red
myRectangle.Width = 500
myRectangle.Height = 500

'Add child elements to the parent StackPanel.
myStackPanel.Children.Add(myTextBlock)
myStackPanel.Children.Add(myRectangle)

'Add the StackPanel as the lone Child of the Border
myScrollViewer.Content = myStackPanel
Me.Content = myScrollViewer

```

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      WindowTitle="ScrollViewer Sample">
  <ScrollViewer HorizontalScrollBarVisibility="Auto">
    <StackPanel VerticalAlignment="Top" HorizontalAlignment="Left">
      <TextBlock TextWrapping="Wrap" Margin="0,0,0,20">Scrolling is enabled when it is necessary.  
Resize the window, making it larger and smaller.</TextBlock>
      <Rectangle Fill="Red" Width="500" Height="500"></Rectangle>
    </StackPanel>
  </ScrollViewer>
</Page>
```

## Styling a ScrollViewer

Like all controls in Windows Presentation Foundation, the [ScrollViewer](#) can be styled in order to change the default rendering behavior of the control. For additional information on control styling, see [Styling and Templating](#).

## Paginating Documents

For document content, an alternative to scrolling is to choose a document container that supports pagination. [FlowDocument](#) is for documents that are designed to be hosted within a viewing control, such as [FlowDocumentPageViewer](#), that supports paginating content across multiple pages, preventing the need for scrolling. [DocumentViewer](#) provides a solution for viewing [FixedDocument](#) content, which uses traditional scrolling to display content outside the realm of the display area.

For additional information about document formats and presentation options, see [Documents in WPF](#).

## See also

- [ScrollViewer](#)
- [ScrollBar](#)
- [IScrollInfo](#)
- [How to: Create a Scroll Viewer](#)
- [Documents in WPF](#)
- [ScrollBar Styles and Templates](#)
- [Controls](#)

# ScrollViewer How-to Topics

2 minutes to read • [Edit Online](#)

The topics in this section demonstrate how to use the [ScrollViewer](#) element to create scrollable regions in your applications.

## In This Section

[Handle the ScrollChanged Event](#)

[Scroll Content by Using the IScrollInfo Interface](#)

[Use the Content-Scrolling Methods of ScrollViewer](#)

## Reference

[ScrollBar](#)

[ScrollViewer](#)

## See also

- [Panels Overview](#)
- [Layout](#)

# How to: Handle the ScrollChanged Event

2 minutes to read • [Edit Online](#)

## Example

This example shows how to handle the `ScrollChanged` event of a `ScrollView`.

A `FlowDocument` element with `Paragraph` parts is defined in XAML. When the `ScrollChanged` event occurs due to user interaction, a handler is invoked, and text is written to a `TextBlock` indicating that the event has occurred.

```
<ScrollView Name="svrContent" CanContentScroll="False" ScrollChanged="sChanged">

<FlowDocument FontFamily="Arial" PageWidth="400">
    <Paragraph>
        Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure.
        Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure.
        Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure.
    </Paragraph>
```

```
<Paragraph>
    Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure.
    Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure.
    Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure.
</Paragraph>

</FlowDocument>
</ScrollView>
```

```

private void sChanged(object sender, ScrollChangedEventArgs e)
{
    if (svrContent.CanContentScroll == true)
    {
        tBlock1.Foreground = System.Windows.Media.Brushes.Red;
        tBlock1.Text = "ScrollChangedEvent just Occurred";
        tBlock2.Text = "ExtentHeight is now " + e.ExtentHeight.ToString();
        tBlock3.Text = "ExtentWidth is now " + e.ExtentWidth.ToString();
        tBlock4.Text = "ExtentHeightChange was " + e.ExtentHeightChange.ToString();
        tBlock5.Text = "ExtentWidthChange was " + e.ExtentWidthChange.ToString();
        tBlock6.Text = "HorizontalOffset is now " + e.HorizontalOffset.ToString();
        tBlock7.Text = "VerticalOffset is now " + e.VerticalOffset.ToString();
        tBlock8.Text = "HorizontalChange was " + e.HorizontalChange.ToString();
        tBlock9.Text = "VerticalChange was " + e.VerticalChange.ToString();
        tBlock10.Text = "ViewportHeight is now " + e.ViewportHeight.ToString();
        tBlock11.Text = "ViewportWidth is now " + e.ViewportWidth.ToString();
        tBlock12.Text = "ViewportHeightChange was " + e.ViewportHeightChange.ToString();
        tBlock13.Text = "ViewportWidthChange was " + e.ViewportWidthChange.ToString();
    }
    else
    {
        tBlock1.Text = "";
    }
}

```

```

Private Sub sChanged(ByVal sender As Object, ByVal args As ScrollChangedEventArgs)

If (sv1.CanContentScroll = True) Then
    tBlock1.Foreground = System.Windows.Media.Brushes.Red
    tBlock1.Text = "ScrollChangedEvent just Occurred"
    tBlock3.Text = "ExtentWidth is now " + args.ExtentWidth.ToString()
    tBlock4.Text = "ExtentHeightChange was " + args.ExtentHeightChange.ToString()
    tBlock5.Text = "ExtentWidthChange was " + args.ExtentWidthChange.ToString()
    tBlock6.Text = "HorizontalOffset is now " + args.HorizontalOffset.ToString()
    tBlock7.Text = "VerticalOffset is now " + args.VerticalOffset.ToString()
    tBlock8.Text = "HorizontalChange was " + args.HorizontalChange.ToString()
    tBlock9.Text = "VerticalChange was " + args.VerticalChange.ToString()
    tBlock10.Text = "ViewportHeight is now " + args.ViewportHeight.ToString()
    tBlock11.Text = "ViewportWidth is now " + args.ViewportWidth.ToString()
    tBlock12.Text = "ViewportHeightChange was " + args.ViewportHeightChange.ToString()
    tBlock13.Text = "ViewportWidthChange was " + args.ViewportWidthChange.ToString()

Else
    tBlock1.Text = ""

End If

```

## See also

- [ScrollViewer](#)
- [ScrollChanged](#)
- [ScrollChangedEventHandler](#)
- [ScrollChangedEventArgs](#)

# How to: Scroll Content by Using the [IScrollInfo](#) Interface

2 minutes to read • [Edit Online](#)

This example shows how to scroll content by using the [IScrollInfo](#) interface.

## Example

The following example demonstrates the features of the [IScrollInfo](#) interface. The example creates a [StackPanel](#) element in Extensible Application Markup Language (XAML) that is nested in a parent [ScrollViewer](#). The child elements of the [StackPanel](#) can be scrolled logically by using the methods defined by the [IScrollInfo](#) interface and cast to the instance of [StackPanel](#) (`sp1`) in code.

```
<Border BorderBrush="Black" Background="White" BorderThickness="2" Width="500" Height="500">
    <ScrollViewer Name="sv1" CanContentScroll="True" VerticalScrollBarVisibility="Visible"
    HorizontalScrollBarVisibility="Visible">
        <StackPanel Name="sp1">
            <Button>Button 1</Button>
            <Button>Button 2</Button>
            <Button>Button 3</Button>
            <Button>Button 4</Button>
            <Button>Button 5</Button>
            <Rectangle Width="700" Height="500" Fill="Purple"/>
            <TextBlock>Rectangle 1</TextBlock>
            <Rectangle Width="700" Height="500" Fill="Red"/>
            <TextBlock>Rectangle 2</TextBlock>
            <Rectangle Width="700" Height="500" Fill="Green"/>
            <TextBlock>Rectangle 3</TextBlock>
        </StackPanel>
    </ScrollViewer>
</Border>
```

Each [Button](#) in the XAML file triggers an associated custom method that controls scrolling behavior in [StackPanel](#). The following example shows how to use the [LineUp](#) and [LineDown](#) methods; it also generically shows how to use all the positioning methods that the [IScrollInfo](#) class defines.

```
private void spLineUp(object sender, RoutedEventArgs e)
{
    ((IScrollInfo)sp1).LineUp();
}
private void spLineDown(object sender, RoutedEventArgs e)
{
    ((IScrollInfo)sp1).LineDown();
}
```

```
Private Sub spLineUp(ByVal sender As Object, ByVal args As RoutedEventArgs)
    CType(sp1, IScrollInfo).LineUp()
End Sub
Private Sub spLineDown(ByVal sender As Object, ByVal args As RoutedEventArgs)
    CType(sp1, IScrollInfo).LineDown()
End Sub
```

## See also

- [ScrollViewer](#)
- [IScrollInfo](#)
- [StackPanel](#)
- [ScrollViewer Overview](#)
- [How-to Topics](#)
- [Panels Overview](#)

# How to: Use the Content-Scrolling Methods of ScrollViewer

2 minutes to read • [Edit Online](#)

This example shows how to use the scrolling methods of the `ScrollViewer` element. These methods provide incremental scrolling of content, either by line or by page, in a `ScrollViewer`.

## Example

The following example creates a `ScrollViewer` named `sv1`, which hosts a child `TextBlock` element. Because the `TextBlock` is larger than the parent `ScrollViewer`, scroll bars appear in order to enable scrolling. `Button` elements that represent the various scrolling methods are docked on the left in a separate `StackPanel`. Each `Button` in the XAML file calls a related custom method that controls scrolling behavior in `ScrollViewer`.

```
<StackPanel DockPanel.Dock="Left" Width="150">
    <Button Margin="3,0,0,2" Background="White" Click="svLineUp">Adjust Line Up</Button>
    <Button Margin="3,0,0,2" Background="White" Click="svLineDown">Adjust Line Down</Button>
    <Button Margin="3,0,0,2" Background="White" Click="svLineRight">Adjust Line Right</Button>
    <Button Margin="3,0,0,2" Background="White" Click="svLineLeft">Adjust Line Left</Button>
    <Button Margin="3,0,0,2" Background="White" Click="svPageUp">Adjust Page Up</Button>
    <Button Margin="3,0,0,2" Background="White" Click="svPageDown">Adjust Page Down</Button>
    <Button Margin="3,0,0,2" Background="White" Click="svPageRight">Adjust Page Right</Button>
    <Button Margin="3,0,0,2" Background="White" Click="svPageLeft">Adjust Page Left</Button>
    <TextBlock Name="txt2" TextWrapping="Wrap"/>
</StackPanel>

<Border BorderBrush="Black" Background="White" BorderThickness="2" Height="520" Width="520"
VerticalAlignment="Top">
    <ScrollViewer VerticalScrollBarVisibility="Visible" HorizontalScrollBarVisibility="Auto" Name="sv1">
        <TextBlock TextWrapping="Wrap" Width="800" Height="1000" Name="txt1"/>
    </ScrollViewer>
</Border>
```

The following example uses the `LineUp` and `LineDown` methods.

```
private void svLineUp(object sender, RoutedEventArgs e)
{
    sv1.LineUp();
}
private void svLineDown(object sender, RoutedEventArgs e)
{
    sv1.LineDown();
}
```

```
Private Sub svLineUp(ByVal sender As Object, ByVal args As RoutedEventArgs)
    sv1.LineUp()
End Sub
Private Sub svLineDown(ByVal sender As Object, ByVal args As RoutedEventArgs)
    sv1.LineDown()
End Sub
```

## See also

- [ScrollViewer](#)
- [StackPanel](#)

# Separator

2 minutes to read • [Edit Online](#)

A **Separator** control draws a line, horizontal or vertical, between items in controls, such as [ListBox](#), [Menu](#), and [ToolBar](#).

## In This Section

### Reference

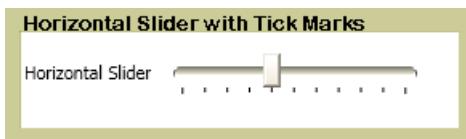
[Separator](#)

# Slider

2 minutes to read • [Edit Online](#)

The [Slider](#) allows you select from a range of values by moving a [Thumb](#) along a [Track](#).

The following illustration shows an example of a horizontal [Slider](#) control.



## In This Section

[Customize the Ticks on a Slider](#)

## Reference

[Slider](#)

[Track](#)

[Thumb](#)

# How to: Customize the Ticks on a Slider

2 minutes to read • [Edit Online](#)

This example shows how to create a [Slider](#) control that has tick marks.

## Example

The [TickBar](#) displays when you set the [TickPlacement](#) property to a value other than [None](#), which is the default value.

The following example shows how to create a [Slider](#) with a [TickBar](#) that displays tick marks. The [TickPlacement](#) and [TickFrequency](#) properties define the location of the tick marks and the interval between them. When you move the [Thumb](#), tooltips display the value of the [Slider](#). The [AutoToolTipPlacement](#) property defines where the tooltips occur. The [Thumb](#) movements correspond to the location of the tick marks because [IsSnapToTickEnabled](#) is set to `true`.

The following example shows how to use the [Ticks](#) property to create tick marks along the [Slider](#) at irregular intervals.

```
<Slider Width="100" Value="50" Orientation="Horizontal" HorizontalAlignment="Left"
        IsSnapToTickEnabled="True" Maximum="3" TickPlacement="BottomRight"
        AutoToolTipPlacement="BottomRight" AutoToolTipPrecision="2"
        Ticks="0, 1.1, 2.5, 3"/>
```

## See also

- [Slider](#)
- [TickBar](#)
- [TickPlacement](#)
- [How to: Bind a Slider to a Property Value](#)

# StackPanel

2 minutes to read • [Edit Online](#)

The [StackPanel](#) element is used to stack child elements horizontally or vertically.

## In This Section

### [How-to Topics](#)

### Reference

[Panel](#)

[Canvas](#)

[DockPanel](#)

[Grid](#)

[StackPanel](#)

[VirtualizingStackPanel](#)

[WrapPanel](#)

## Related Sections

[Layout](#)

[Walkthrough: My first WPF desktop application](#)

[ScrollViewer Overview](#)

# StackPanel How-to Topics

2 minutes to read • [Edit Online](#)

The topics in this section describe how to use the [StackPanel](#) element to stack content horizontally or vertically.

## In This Section

[Choose Between StackPanel and DockPanel](#)

[Create a StackPanel](#)

[Horizontally or Vertically Align Content in a StackPanel](#)

## Reference

[Panel](#)

[Canvas](#)

[DockPanel](#)

[Grid](#)

[StackPanel](#)

[VirtualizingStackPanel](#)

[WrapPanel](#)

## Related Sections

[Layout](#)

[Walkthrough: My first WPF desktop application](#)

[ScrollViewer Overview](#)

# How to: Choose Between StackPanel and DockPanel

4 minutes to read • [Edit Online](#)

This example shows how to choose between using a [StackPanel](#) or a [DockPanel](#) when you stack content in a [Panel](#).

## Example

Although you can use either [DockPanel](#) or [StackPanel](#) to stack child elements, the two controls do not always produce the same results. For example, the order that you place child elements can affect the size of child elements in a [DockPanel](#) but not in a [StackPanel](#). This different behavior occurs because [StackPanel](#) measures in the direction of stacking at [Double.PositiveInfinity](#); however, [DockPanel](#) measures only the available size.

The following example demonstrates this key difference between [DockPanel](#) and [StackPanel](#).

```
// Create the application's main window
mainWindow = gcnew Window();
mainWindow->Title = "StackPanel vs. DockPanel";

// Add root Grid
myGrid = gcnew Grid();
myGrid->Width = 175;
myGrid->Height = 150;
RowDefinition^ myRowDef1 = gcnew RowDefinition();
RowDefinition^ myRowDef2 = gcnew RowDefinition();
myGrid->RowDefinitions->Add(myRowDef1);
myGrid->RowDefinitions->Add(myRowDef2);

// Define the DockPanel
myDockPanel = gcnew DockPanel();
Grid::SetRow(myDockPanel, 0);

//Define an Image and Source
Image^ myImage = gcnew Image();
BitmapImage^ bi = gcnew BitmapImage();
bi->BeginInit();
bi->UriSource = gcnew System::Uri("smiley_stackpanel.png", UriKind::Relative);
bi->EndInit();
myImage->Source = bi;

Image^ myImage2 = gcnew Image();
BitmapImage^ bi2 = gcnew BitmapImage();
bi2->BeginInit();
bi2->UriSource = gcnew System::Uri("smiley_stackpanel.png", UriKind::Relative);
bi2->EndInit();
myImage2->Source = bi2;

Image^ myImage3 = gcnew Image();
BitmapImage^ bi3 = gcnew BitmapImage();
bi3->BeginInit();
bi3->UriSource = gcnew System::Uri("smiley_stackpanel.PNG", UriKind::Relative);
bi3->EndInit();
myImage3->Stretch = Stretch::Fill;
myImage3->Source = bi3;

// Add the images to the parent DockPanel
myDockPanel->Children->Add(myImage);
myDockPanel->Children->Add(myImage2);
myDockPanel->Children->Add(myImage3);
```

```

//Define a StackPanel
myStackPanel = gcnew StackPanel();
myStackPanel->Orientation = Orientation::Horizontal;
Grid::SetRow(myStackPanel, 1);

Image^ myImage4 = gcnew Image();
BitmapImage^ bi4 = gcnew BitmapImage();
bi4->BeginInit();
bi4->UriSource = gcnew System::Uri("smiley_stackpanel.png", UriKind::Relative);
bi4->EndInit();
myImage4->Source = bi4;

Image^ myImage5 = gcnew Image();
BitmapImage^ bi5 = gcnew BitmapImage();
bi5->BeginInit();
bi5->UriSource = gcnew System::Uri("smiley_stackpanel.png", UriKind::Relative);
bi5->EndInit();
myImage5->Source = bi5;

Image^ myImage6 = gcnew Image();
BitmapImage^ bi6 = gcnew BitmapImage();
bi6->BeginInit();
bi6->UriSource = gcnew System::Uri("smiley_stackpanel.PNG", UriKind::Relative);
bi6->EndInit();
myImage6->Stretch = Stretch::Fill;
myImage6->Source = bi6;

// Add the images to the parent StackPanel
myStackPanel->Children->Add(myImage4);
myStackPanel->Children->Add(myImage5);
myStackPanel->Children->Add(myImage6);

// Add the layout panels as children of the Grid
myGrid->Children->Add(myDockPanel);
myGrid->Children->Add(myStackPanel);

// Add the Grid as the Content of the Parent Window Object
mainWindow->Content = myGrid;
mainWindow->Show();

```

```

// Create the application's main window
mainWindow = new Window ();
mainWindow.Title = "StackPanel vs. DockPanel";

// Add root Grid
myGrid = new Grid();
myGrid.Width = 175;
myGrid.Height = 150;
RowDefinition myRowDef1 = new RowDefinition();
RowDefinition myRowDef2 = new RowDefinition();
myGrid.RowDefinitions.Add(myRowDef1);
myGrid.RowDefinitions.Add(myRowDef2);

// Define the DockPanel
myDockPanel = new DockPanel();
Grid::SetRow(myDockPanel, 0);

//Define an Image and Source
Image myImage = new Image();
BitmapImage bi = new BitmapImage();
bi.BeginInit();
bi.UriSource = new Uri("smiley_stackpanel.png", UriKind.Relative);
bi.EndInit();
myImage.Source = bi;

```

```

Image myImage2 = new Image();
BitmapImage bi2 = new BitmapImage();
bi2.BeginInit();
bi2.UriSource = new Uri("smiley_stackpanel.png", UriKind.Relative);
bi2.EndInit();
myImage2.Source = bi2;

Image myImage3 = new Image();
BitmapImage bi3 = new BitmapImage();
bi3.BeginInit();
bi3.UriSource = new Uri("smiley_stackpanel.PNG", UriKind.Relative);
bi3.EndInit();
myImage3.Stretch = Stretch.Fill;
myImage3.Source = bi3;

// Add the images to the parent DockPanel
myDockPanel.Children.Add(myImage);
myDockPanel.Children.Add(myImage2);
myDockPanel.Children.Add(myImage3);

//Define a StackPanel
myStackPanel = new StackPanel();
myStackPanel.Orientation = Orientation.Horizontal;
Grid.SetRow(myStackPanel, 1);

Image myImage4 = new Image();
BitmapImage bi4 = new BitmapImage();
bi4.BeginInit();
bi4.UriSource = new Uri("smiley_stackpanel.png", UriKind.Relative);
bi4.EndInit();
myImage4.Source = bi4;

Image myImage5 = new Image();
BitmapImage bi5 = new BitmapImage();
bi5.BeginInit();
bi5.UriSource = new Uri("smiley_stackpanel.png", UriKind.Relative);
bi5.EndInit();
myImage5.Source = bi5;

Image myImage6 = new Image();
BitmapImage bi6 = new BitmapImage();
bi6.BeginInit();
bi6.UriSource = new Uri("smiley_stackpanel.PNG", UriKind.Relative);
bi6.EndInit();
myImage6.Stretch = Stretch.Fill;
myImage6.Source = bi6;

// Add the images to the parent StackPanel
myStackPanel.Children.Add(myImage4);
myStackPanel.Children.Add(myImage5);
myStackPanel.Children.Add(myImage6);

// Add the layout panels as children of the Grid
myGrid.Children.Add(myDockPanel);
myGrid.Children.Add(myStackPanel);

// Add the Grid as the Content of the Parent Window Object
mainWindow.Content = myGrid;
mainWindow.Show ();

```

```

'Add root Grid
Dim myGrid As New Grid
myGrid.Width = 175
myGrid.Height = 150
Dim myRowDef1 As New RowDefinition
Dim myColDef1 As New ColumnDefinition

```

```

Dim myRowDef2 As New RowDefinition
myGrid.RowDefinitions.Add(myRowDef1)
myGrid.RowDefinitions.Add(myRowDef2)

'Define the DockPanel
Dim myDockPanel As New DockPanel
Grid.SetRow(myDockPanel, 0)

'Define an Image and Source.
Dim myImage As New Image
Dim bi As New BitmapImage
bi.BeginInit()
bi.UriSource = New Uri("smiley_stackpanel.png", UriKind.Relative)
bi.EndInit()
myImage.Source = bi

Dim myImage2 As New Image
Dim bi2 As New BitmapImage
bi2.BeginInit()
bi2.UriSource = New Uri("smiley_stackpanel.png", UriKind.Relative)
bi2.EndInit()
myImage2.Source = bi2

Dim myImage3 As New Image
Dim bi3 As New BitmapImage
bi3.BeginInit()
bi3.UriSource = New Uri("smiley_stackpanel.PNG", UriKind.Relative)
bi3.EndInit()
myImage3.Stretch = Stretch.Fill
myImage3.Source = bi3

'Add the images to the parent DockPanel.
myDockPanel.Children.Add(myImage)
myDockPanel.Children.Add(myImage2)
myDockPanel.Children.Add(myImage3)

'Define a StackPanel.
Dim myStackPanel As New StackPanel
myStackPanel.Orientation = Orientation.Horizontal
Grid.SetRow(myStackPanel, 1)

Dim myImage4 As New Image
Dim bi4 As New BitmapImage
bi4.BeginInit()
bi4.UriSource = New Uri("smiley_stackpanel.png", UriKind.Relative)
bi4.EndInit()
myImage4.Source = bi4

Dim myImage5 As New Image
Dim bi5 As New BitmapImage
bi5.BeginInit()
bi5.UriSource = New Uri("smiley_stackpanel.png", UriKind.Relative)
bi5.EndInit()
myImage5.Source = bi5

Dim myImage6 As New Image
Dim bi6 As New BitmapImage
bi6.BeginInit()
bi6.UriSource = New Uri("smiley_stackpanel.PNG", UriKind.Relative)
bi6.EndInit()
myImage6.Stretch = Stretch.Fill
myImage6.Source = bi6

'Add the images to the parent StackPanel.
myStackPanel.Children.Add(myImage4)
myStackPanel.Children.Add(myImage5)
myStackPanel.Children.Add(myImage6)

'Add the layout panels as children of the Grid

```

```
myGrid.Children.Add(myDockPanel)
myGrid.Children.Add(myStackPanel)
```

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      WindowTitle="StackPanel vs. DockPanel">
    <Grid Width="175" Height="150">
        <Grid.ColumnDefinitions>
            <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition />
        </Grid.RowDefinitions>

        <DockPanel Grid.Column="0" Grid.Row="0">
            <Image Source="smiley_stackpanel.png" />
            <Image Source="smiley_stackpanel.png" />
            <Image Source="smiley_stackpanel.png" Stretch="Fill"/>
        </DockPanel>

        <StackPanel Grid.Column="0" Grid.Row="1" Orientation="Horizontal">
            <Image Source="smiley_stackpanel.png" />
            <Image Source="smiley_stackpanel.png" />
            <Image Source="smiley_stackpanel.png" Stretch="Fill"/>
        </StackPanel>
    </Grid>
</Page>
```

## See also

- [StackPanel](#)
- [DockPanel](#)
- [Panels Overview](#)

# How to: Create a StackPanel

2 minutes to read • [Edit Online](#)

This example shows how to create a [StackPanel](#).

## Example

A [StackPanel](#) allows you to stack elements in a specified direction. By using properties that are defined on [StackPanel](#), content can flow both vertically, which is the default setting, or horizontally.

The following example vertically stacks five [TextBlock](#) controls, each with a different [Border](#) and [Background](#), by using [StackPanel](#). The child elements that have no specified [Width](#) stretch to fill the parent window; however, the child elements that have a specified [Width](#), are centered within the window.

The default stack direction in a [StackPanel](#) is vertical. To control content flow in a [StackPanel](#), use the [Orientation](#) property. You can control horizontal alignment by using the [HorizontalAlignment](#) property.

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" WindowTitle="StackPanel Sample">
  <StackPanel>
    <Border Background="SkyBlue" BorderBrush="Black" BorderThickness="1">
      <TextBlock Foreground="Black" FontSize="12">Stacked Item #1</TextBlock>
    </Border>
    <Border Width="400" Background="CadetBlue" BorderBrush="Black" BorderThickness="1">
      <TextBlock Foreground="Black" FontSize="14">Stacked Item #2</TextBlock>
    </Border>
    <Border Background="LightGoldenRodYellow" BorderBrush="Black" BorderThickness="1">
      <TextBlock Foreground="Black" FontSize="16">Stacked Item #3</TextBlock>
    </Border>
    <Border Width="200" Background="PaleGreen" BorderBrush="Black" BorderThickness="1">
      <TextBlock Foreground="Black" FontSize="18">Stacked Item #4</TextBlock>
    </Border>
    <Border Background="White" BorderBrush="Black" BorderThickness="1">
      <TextBlock Foreground="Black" FontSize="20">Stacked Item #5</TextBlock>
    </Border>
  </StackPanel>
</Page>
```

## See also

- [StackPanel](#)
- [Panels Overview](#)
- [How-to Topics](#)

# How to: Horizontally or Vertically Align Content in a StackPanel

2 minutes to read • [Edit Online](#)

This example shows how to adjust the [Orientation](#) of content within a [StackPanel](#) element, and also how to adjust the [HorizontalAlignment](#) and [VerticalAlignment](#) of child content.

## Example

The following example creates three [ListBox](#) elements in Extensible Application Markup Language (XAML). Each [ListBox](#) represents the possible values of the [Orientation](#), [HorizontalAlignment](#), and [VerticalAlignment](#) properties of a [StackPanel](#). When a user selects a value in any of the [ListBox](#) elements, the associated property of the [StackPanel](#) and its child [Button](#) elements change.

```
<ListBox VerticalAlignment="Top" SelectionChanged="changeOrientation" Grid.Row="2" Grid.Column="1"
Width="100" Height="50" Margin="0,0,0,10">
    <ListBoxItem>Horizontal</ListBoxItem>
    <ListBoxItem>Vertical</ListBoxItem>
</ListBox>

<ListBox VerticalAlignment="Top" SelectionChanged="changeHorAlign" Grid.Row="2" Grid.Column="3"
Width="100" Height="50" Margin="0,0,0,10">
    <ListBoxItem>Left</ListBoxItem>
    <ListBoxItem>Right</ListBoxItem>
    <ListBoxItem>Center</ListBoxItem>
    <ListBoxItem>Stretch</ListBoxItem>
</ListBox>

<ListBox VerticalAlignment="Top" SelectionChanged="changeVertAlign" Grid.Row="2" Grid.Column="5"
Width="100" Height="50" Margin="0,0,0,10">
    <ListBoxItem>Top</ListBoxItem>
    <ListBoxItem>Bottom</ListBoxItem>
    <ListBoxItem>Center</ListBoxItem>
    <ListBoxItem>Stretch</ListBoxItem>
</ListBox>

<StackPanel Grid.ColumnSpan="6" Grid.Row="3" Name="sp1" Background="Yellow">
    <Button>Button One</Button>
    <Button>Button Two</Button>
    <Button>Button Three</Button>
    <Button>Button Four</Button>
    <Button>Button Five</Button>
    <Button>Button Six</Button>
</StackPanel>
```

The following code-behind file defines the changes to the events that are associated with the [ListBox](#) selection changes. [StackPanel](#) is identified by the [Name](#) `sp1`.

```

private void changeOrientation(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    if (li.Content.ToString() == "Horizontal")
    {
        sp1.Orientation = System.Windows.Controls.Orientation.Horizontal;
    }
    else if (li.Content.ToString() == "Vertical")
    {
        sp1.Orientation = System.Windows.Controls.Orientation.Vertical;
    }
}

private void changeHorAlign(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    if (li.Content.ToString() == "Left")
    {
        sp1.HorizontalAlignment = System.Windows.HorizontalAlignment.Left;
    }
    else if (li.Content.ToString() == "Right")
    {
        sp1.HorizontalAlignment = System.Windows.HorizontalAlignment.Right;
    }
    else if (li.Content.ToString() == "Center")
    {
        sp1.HorizontalAlignment = System.Windows.HorizontalAlignment.Center;
    }
    else if (li.Content.ToString() == "Stretch")
    {
        sp1.HorizontalAlignment = System.Windows.HorizontalAlignment.Stretch;
    }
}

private void changeVertAlign(object sender, SelectionChangedEventArgs args)
{
    ListBoxItem li = ((sender as ListBox).SelectedItem as ListBoxItem);
    if (li.Content.ToString() == "Top")
    {
        sp1.VerticalAlignment = System.Windows.VerticalAlignment.Top;
    }
    else if (li.Content.ToString() == "Bottom")
    {
        sp1.VerticalAlignment = System.Windows.VerticalAlignment.Bottom;
    }
    else if (li.Content.ToString() == "Center")
    {
        sp1.VerticalAlignment = System.Windows.VerticalAlignment.Center;
    }
    else if (li.Content.ToString() == "Stretch")
    {
        sp1.VerticalAlignment = System.Windows.VerticalAlignment.Stretch;
    }
}

```

```

Private Sub changeOrientation(ByVal sender As Object, ByVal args As SelectionChangedEventArgs)
    Dim li As ListBoxItem = CType(CType(sender, ListBox).SelectedItem, ListBoxItem)
    If (li.Content.ToString() = "Horizontal") Then
        sp1.Orientation = System.Windows.Controls.Orientation.Horizontal
    ElseIf li.Content.ToString() = "Vertical" Then
        sp1.Orientation = System.Windows.Controls.Orientation.Vertical
    End If
End Sub

Private Sub changeHorAlign(ByVal sender As Object, ByVal args As SelectionChangedEventArgs)
    Dim li As ListBoxItem = CType(CType(sender, ListBox).SelectedItem, ListBoxItem)
    If (li.Content.ToString() = "Left") Then
        sp1.HorizontalAlignment = System.Windows.HorizontalAlignment.Left
    ElseIf (li.Content.ToString() = "Right") Then
        sp1.HorizontalAlignment = System.Windows.HorizontalAlignment.Right
    ElseIf (li.Content.ToString() = "Center") Then
        sp1.HorizontalAlignment = System.Windows.HorizontalAlignment.Center
    ElseIf (li.Content.ToString() = "Stretch") Then
        sp1.HorizontalAlignment = System.Windows.HorizontalAlignment.Stretch
    End If
End Sub

Private Sub changeVertAlign(ByVal sender As Object, ByVal args As SelectionChangedEventArgs)
    Dim li As ListBoxItem = CType(CType(sender, ListBox).SelectedItem, ListBoxItem)
    If (li.Content.ToString() = "Top") Then
        sp1.VerticalAlignment = System.Windows.VerticalAlignment.Top
    ElseIf (li.Content.ToString() = "Bottom") Then
        sp1.VerticalAlignment = System.Windows.VerticalAlignment.Bottom
    ElseIf (li.Content.ToString() = "Center") Then
        sp1.VerticalAlignment = System.Windows.VerticalAlignment.Center
    ElseIf (li.Content.ToString() = "Stretch") Then
        sp1.VerticalAlignment = System.Windows.VerticalAlignment.Stretch
    End If
End Sub

```

## See also

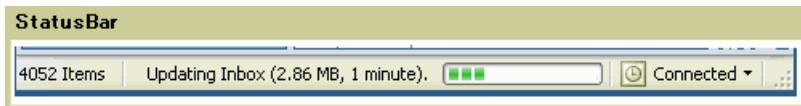
- [StackPanel](#)
- [ListBox](#)
- [HorizontalAlignment](#)
- [VerticalAlignment](#)
- [Panels Overview](#)

# StatusBar

2 minutes to read • [Edit Online](#)

A [StatusBar](#) is a horizontal area at the bottom of a window where an application can display status information.

The following illustration shows an example of a [StatusBar](#).



## In This Section

### Reference

[StatusBar](#)

[StatusBarItem](#)

### Related Sections

# TabControl

2 minutes to read • [Edit Online](#)

[TabControl](#) elements display content on discrete pages accessed by selecting the appropriate tab. Each tab contains a [TabItem](#).

The following illustration shows a [TabControl](#).



Typical TabControl

## Reference

[TabControl](#)

[TabItem](#)

## Related Sections

# TextBlock

2 minutes to read • [Edit Online](#)

The [TextBlock](#) control provides flexible text support for WPF applications. The element is targeted primarily toward basic UI scenarios that do not require more than one paragraph of text.

## In This Section

[TextBlock Overview](#)

## Reference

[Label](#)

## Related Sections

[Documents in WPF](#)

[Flow Document Overview](#)

# TextBlock Overview

2 minutes to read • [Edit Online](#)

The [TextBlock](#) control provides flexible text support for WPF applications. The element is targeted primarily toward basic UI scenarios that do not require more than one paragraph of text. It supports a number of properties that enable precise control of presentation, such as [FontFamily](#), [FontSize](#), [FontWeight](#), [TextEffects](#), and [TextWrapping](#). Text content can be added using the [Text](#) property. When used in XAML, content between the open and closing tag is implicitly added as the text of the element.

A [TextBlock](#) element can be instantiated very simply using XAML.

```
<TextBlock FontSize="18" FontWeight="Bold" FontStyle="Italic">
    Hello, world!
</TextBlock>
```

Similarly, usage of the [TextBlock](#) element in code is relatively simple.

```
TextBlock myTextBlock = new TextBlock();
myTextBlock.FontSize = 18;
myTextBlock.FontWeight = FontWeights.Bold;
myTextBlock.FontStyle = FontStyles.Italic;
myTextBlock.Text = "Hello, world!";
```

```
Dim myTextBlock As New TextBlock()
myTextBlock.FontSize = 18
myTextBlock.FontWeight = FontWeights.Bold
myTextBlock.FontStyle = FontStyles.Italic
myTextBlock.Text = "Hello, world!"
```

## See also

- [Label](#)

# TextBox

2 minutes to read • [Edit Online](#)

The [TextBox](#) control provides support for basic text input in WPF applications.

## In This Section

[TextBox Overview](#)

[How-to Topics](#)

## Reference

[TextBox](#)

[RichTextBox](#)

[TextBlock](#)

[PasswordBox](#)

## See also

- [WPF Controls Gallery Sample](#)
- [TextBox Styles and Templates](#)

# TextBox Overview

3 minutes to read • [Edit Online](#)

The [TextBox](#) class enables you to display or edit unformatted text. A common use of a [TextBox](#) is editing unformatted text in a form. For example, a form asking for the user's name, phone number, etc would use [TextBox](#) controls for text input. This topic introduces the [TextBox](#) class and provides examples of how to use it in both Extensible Application Markup Language (XAML) and C#.

## TextBox or RichTextBox?

Both [TextBox](#) and [RichTextBox](#) allow users to input text but the two controls are used for different scenarios. A [TextBox](#) requires less system resources than a [RichTextBox](#) so it is ideal when only plain text needs to be edited (i.e., usage in a form). A [RichTextBox](#) is a better choice when it is necessary for the user to edit formatted text, images, tables, or other supported content. For example, editing a document, article, or blog that requires formatting, images, etc is best accomplished using a [RichTextBox](#). The table below summarizes the primary features of [TextBox](#) and [TextBox](#).

CONTROL	REAL-TIME SPELLCHECKING	CONTEXT MENU	FORMATTING COMMANDS LIKE <a href="#">TOGGLEBOLD</a> (CTR+B)	<a href="#">FLOWDOCUMENT</a> CONTENT LIKE IMAGES, PARAGRAPHS, TABLES, ETC.
TextBox	Yes	Yes	No	No.
RichTextBox	Yes	Yes	Yes (see <a href="#">RichTextBox Overview</a> )	Yes (see <a href="#">RichTextBox Overview</a> )

### NOTE

Although [TextBox](#) does not support formatting related editing commands like [ToggleBold](#) (Ctr+B), many basic commands are supported by both controls such as [MoveToLineEnd](#). See [EditingCommands](#) for more information.

Features supported by [TextBox](#) are covered in the sections below. For more information about [RichTextBox](#), see [RichTextBox Overview](#).

### Real-time Spellchecking

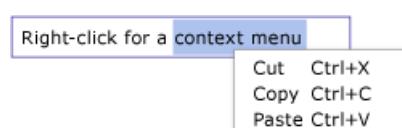
You can enable real-time spellchecking in a [TextBox](#) or [RichTextBox](#). When spellchecking is turned on, a red line appears underneath any misspelled words (see picture below).

The following word is Missspelled. |

See [Enable Spell Checking in a Text Editing Control](#) to learn how to enable spellchecking.

### Context Menu

By default, both [TextBox](#) and [RichTextBox](#) have a context menu that appears when a user right-clicks inside the control. The context menu allows the user to cut, copy, or paste (see picture below).



You can create your own custom context menu to override the default behavior. See [Use a Custom Context Menu with a TextBox](#) for more information.

## Creating TextBoxes

A [TextBox](#) can be a single line in height or comprise multiple lines. A single line [TextBox](#) is best for inputting small amounts of plain text (i.e. "Name", "Phone Number", etc. in a form). The following example shows how to create a single line [TextBox](#).

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <StackPanel>
        <TextBox Width="200" MaxLength="100" />
    </StackPanel>
</Page>
```

You can also create a [TextBox](#) that allows the user to enter multiple lines of text. For example, if your form asked for a biographical sketch of the user, you would want to use a [TextBox](#) that supports multiple lines of text. The following example shows how to use Extensible Application Markup Language (XAML) to define a [TextBox](#) control that automatically expands to accommodate multiple lines of text.

```
<TextBox
    Name="tbMultiLine"
    TextWrapping="Wrap"
    AcceptsReturn="True"
    VerticalScrollBarVisibility="Visible"
>
    This TextBox will allow the user to enter multiple lines of text. When the RETURN key is pressed,
    or when typed text reaches the edge of the text box, a new line is automatically inserted.
</TextBox>
```

Setting the [TextWrapping](#) attribute to `Wrap` causes text to wrap to a new line when the edge of the [TextBox](#) control is reached, automatically expanding the [TextBox](#) control to include room for a new line, if necessary.

Setting the [AcceptsReturn](#) attribute to `true` causes a new line to be inserted when the RETURN key is pressed, once again automatically expanding the [TextBox](#) to include room for a new line, if necessary.

The [VerticalScrollBarVisibility](#) attribute adds a scroll bar to the [TextBox](#), so that the contents of the [TextBox](#) can be scrolled through if the [TextBox](#) expands beyond the size of the frame or window that encloses it.

For more information on different tasks associated with using a [TextBox](#), see [How-to Topics](#).

## Detect When Content Changes

Usually the [TextChanged](#) event should be used to detect whenever the text in a [TextBox](#) or [RichTextBox](#) changes, rather than [KeyDown](#) as you might expect. See [Detect When Text in a TextBox Has Changed](#) for an example.

## See also

- [How-to Topics](#)
- [RichTextBox Overview](#)

# TextBox How-to Topics

2 minutes to read • [Edit Online](#)

This section provides examples that demonstrate how to accomplish common tasks using the [TextBox](#) control.

## In This Section

[Create a Multiline TextBox Control](#)

[Detect When Text in a TextBox Has Changed](#)

[Enable Tab Characters in a TextBox Control](#)

[Get a Collection of Lines from a TextBox](#)

[Make a TextBox Control Read-Only](#)

[Position the Cursor at the Beginning or End of Text in a TextBox Control](#)

[Retrieve a Text Selection](#)

[Set Focus in a TextBox Control](#)

[Set the Text Content of a TextBox Control](#)

[Enable Spell Checking in a Text Editing Control](#)

[Use a Custom Context Menu with a TextBox](#)

[Use Spell Checking with a Context Menu](#)

[Add a Watermark to a TextBox](#)

## Reference

[TextBox](#)

[RichTextBox](#)

[TextBlock](#)

[PasswordBox](#)

## See also

- [WPF Controls Gallery Sample](#)

- [TextBox Styles and Templates](#)

# How to: Create a Multiline TextBox Control

2 minutes to read • [Edit Online](#)

This example shows how to use Extensible Application Markup Language (XAML) to define a [TextBox](#) control that will automatically expand to accommodate multiple lines of text.

## Example

Setting the [TextWrapping](#) attribute to **Wrap** will cause entered text to wrap to a new line when the edge of the [TextBox](#) control is reached, automatically expanding the [TextBox](#) control to include room for a new line, if necessary.

Setting the [AcceptsReturn](#) attribute to **true** causes a new line to be inserted when the RETURN key is pressed, once again automatically expanding the [TextBox](#) to include room for a new line, if necessary.

The [VerticalScrollBarVisibility](#) attribute adds a scroll bar to the [TextBox](#), so that the contents of the [TextBox](#) can be scrolled through if the [TextBox](#) expands beyond the size of the frame or window that encloses it.

```
<TextBox  
    Name="tbMultiLine"  
    TextWrapping="Wrap"  
    AcceptsReturn="True"  
    VerticalScrollBarVisibility="Visible"  
>  
    This TextBox will allow the user to enter multiple lines of text. When the RETURN key is pressed,  
    or when typed text reaches the edge of the text box, a new line is automatically inserted.  
</TextBox>
```

## See also

- [TextWrapping](#)
- [TextBox Overview](#)
- [RichTextBox Overview](#)

# How to: Detect When Text in a TextBox Has Changed

2 minutes to read • [Edit Online](#)

This example shows one way to use the [TextChanged](#) event to execute a method whenever the text in a [TextBox](#) control has changed.

In the code-behind class for the XAML that contains the [TextBox](#) control that you want to monitor for changes, insert a method to call whenever the [TextChanged](#) event fires. This method must have a signature that matches what is expected by the [TextChangedEventHandler](#) delegate.

The event handler is called whenever the contents of the [TextBox](#) control are changed, either by a user or programmatically.

## NOTE

This event fires when the [TextBox](#) control is created and initially populated with text.

## Example

In the Extensible Application Markup Language (XAML) that defines your [TextBox](#) control, specify the [TextChanged](#) attribute with a value that matches the event handler method name.

```
<TextBox TextChanged="textChangedEventHandler">
    Here is the initial text in my TextBox.  Each time the contents of this TextBox are changed,
    the TextChanged event fires and textChangedEventHandler is called.
</TextBox>
```

## Example

In the code-behind class for the XAML that contains the [TextBox](#) control that you want to monitor for changes, insert a method to call whenever the [TextChanged](#) event fires. This method must have a signature that matches what is expected by the [TextChangedEventHandler](#) delegate.

```
// TextChangedEventHandler delegate method.
private void textChangedEventHandler(object sender, TextChangedEventArgs args)
{
    // Omitted Code: Insert code that does something whenever
    // the text changes...
} // end textChangedEventHandler
```

```
' TextChangedEventHandler delegate method.
Private Sub textChangedEventHandler(ByVal sender As Object, ByVal args As TextChangedEventArgs)
    ' Omitted Code: Insert code that does something whenever
    ' the text changes...
End Sub
```

The event handler is called whenever the contents of the [TextBox](#) control are changed, either by a user or programmatically.

**NOTE**

This event fires when the [TextBox](#) control is created and initially populated with text.

Comments

## See also

- [TextChangedEventArgs](#)
- [TextBox Overview](#)
- [RichTextBox Overview](#)

# How to: Enable Tab Characters in a TextBox Control

2 minutes to read • [Edit Online](#)

This example shows how to enable the acceptance of tab characters as normal input in a [TextBox](#) control.

## Example

To enable the acceptance of tab characters as input in a [TextBox](#) control, set the [AcceptsTab](#) attribute to **true**.

```
<TextBox AcceptsTab="True">
    If the AcceptsTab element is "True", the TextBox control will accept tab characters as regular input when
    the TAB key is pressed.
    If AcceptsTab is "False" (the default), pressing TAB moves the focus to the next focusable control.
</TextBox>
```

## See also

- [TextBox Overview](#)
- [RichTextBox Overview](#)

# How to: Get a Collection of Lines from a TextBox

2 minutes to read • [Edit Online](#)

This example shows how to get a collection of lines of text from a [TextBox](#).

## Example

The following example shows a simple method that takes a [TextBox](#) as the argument, and returns a [StringCollection](#) containing the lines of text in the [TextBox](#). The [LineCount](#) property is used to determine how many lines are currently in the [TextBox](#), and the [GetLineText](#) method is then used to extract each line and add it to the collection of lines.

```
StringCollection GetLinesCollectionFromTextBox(TextBox textBox)
{
    StringCollection lines = new StringCollection();

    // lineCount may be -1 if TextBox layout info is not up-to-date.
    int lineCount = textBox.LineCount;

    for (int line = 0; line < lineCount; line++)
        // GetLineText takes a zero-based line index.
        lines.Add(textBox.GetLineText(line));

    return lines;
}
```

## See also

- [TextBox Overview](#)
- [RichTextBox Overview](#)

# How to: Make a TextBox Control Read-Only

2 minutes to read • [Edit Online](#)

This example shows how to configure a [TextBox](#) control to not allow user input or modification.

## Example

To prevent users from modifying the contents of a [TextBox](#) control, set the [IsReadOnly](#) attribute to **true**.

```
<TextBox  
    IsReadOnly="True"  
>  
    The user may not modify the contents of this TextBox.  
</TextBox>
```

The [IsReadOnly](#) attribute affects user input only; it does not affect text set in the Extensible Application Markup Language (XAML) description of a [TextBox](#) control, or text set programmatically through the [Text](#) property.

The default value of [IsReadOnly](#) is **false**.

## See also

- [TextBox Overview](#)
- [RichTextBox Overview](#)

# How to: Position the Cursor at the Beginning or End of Text in a TextBox Control

2 minutes to read • [Edit Online](#)

This example shows how to position the cursor at the beginning or end of the text contents of a [TextBox](#) control.

## Example

The following Extensible Application Markup Language (XAML) code describes a [TextBox](#) control and assigns it a Name.

```
<TextBox  
    Name="tbPositionCursor"  
>  
    Here is some text in my text box...  
</TextBox>
```

## Example

To position the cursor at the beginning of the contents of a [TextBox](#) control, call the [Select](#) method and specify the selection start position of 0, and a selection length of 0.

```
tbPositionCursor.Select(0, 0);
```

```
tbPositionCursor.Select(0, 0)
```

## Example

To position the cursor at the end of the contents of a [TextBox](#) control, call the [Select](#) method and specify the selection start position equal to the length of the text content, and a selection length of 0.

```
tbPositionCursor.Select(tbPositionCursor.Text.Length, 0);
```

```
tbPositionCursor.Select(tbPositionCursor.Text.Length, 0)
```

## See also

- [TextBox Overview](#)
- [RichTextBox Overview](#)

# How to: Retrieve a Text Selection

2 minutes to read • [Edit Online](#)

This example shows one way to use the [SelectedText](#) property to retrieve text that the user has selected in a [TextBox](#) control.

## Example

The following Extensible Application Markup Language (XAML) example shows the definition of a [TextBox](#) control that contains some text to select, and a [Button](#) control with a specified [OnClick](#) method.

In this example, a button with an associated [Click](#) event handler is used to retrieve the text selection. When the user clicks the button, the [OnClick](#) method copies any selected text in the textbox into a string. The particular circumstances by which the text selection is retrieved (clicking a button), as well as the action taken with that selection (copying the text selection to a string), can easily be modified to accommodate a wide variety of scenarios.

```
<TextBox Name="tbSelectSomeText">
    Some text to select...
</TextBox>

<Button Click="OnClick">Retrieve Selection</Button>
```

## Example

The following C# example shows an [OnClick](#) event handler for the button defined in the XAML for this example.

```
void OnClick(object sender, RoutedEventArgs e)
{
    String sSelectedText = tbSelectSomeText.SelectedText;
}
```

```
Private Sub OnClick(ByVal senter As Object, ByVal e As RoutedEventArgs)
    Dim sSelectedText As String = tbSelectSomeText.SelectedText
End Sub
```

## See also

- [TextBox Overview](#)
- [RichTextBox Overview](#)

# How to: Set Focus in a TextBox Control

2 minutes to read • [Edit Online](#)

This example shows how to use the [Focus](#) method to set focus on a [TextBox](#) control.

## Example

The following Extensible Application Markup Language (XAML) example describes a simple [TextBox](#) control named *tbFocusMe*

```
<TextBox Name="tbFocusMe">
    This is the text in my text box.
</TextBox>
```

## Example

The following example calls the [Focus](#) method to set the focus on the [TextBox](#) control with the Name *tbFocusMe*.

```
tbFocusMe.Focus();
```

```
tbFocusMe.Focus()
```

## See also

- [Focusable](#)
- [IsFocused](#)
- [TextBox Overview](#)
- [RichTextBox Overview](#)

# How to: Set the Text Content of a TextBox Control

2 minutes to read • [Edit Online](#)

This example shows how to use the [Text](#) property to set the initial text contents of a [TextBox](#) control.

## NOTE

Although the Extensible Application Markup Language (XAML) version of the example could use the `<TextBox.Text>` tags around the text of each button's [TextBox](#) content, it is not necessary because the [TextBox](#) applies the [ContentPropertyAttribute](#) attribute to the [Text](#) property. For more information, see [XAML Overview \(WPF\)](#).

## Example

```
<TextBox Name="tbSettingText">
    Initial text contents of the TextBox.
</TextBox>
```

## Example

```
tbSettingText.Text = "Initial text contents of the TextBox.;"
```

```
tbSettingText.Text = "Initial text contents of the TextBox."
```

## See also

- [TextBox Overview](#)
- [RichTextBox Overview](#)

# How to: Enable Spell Checking in a Text Editing Control

2 minutes to read • [Edit Online](#)

The following example shows how to enable real-time spell checking in a `TextBox` by using the `.IsEnabled` property of the `SpellCheck` class.

## Example

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <StackPanel>
        <TextBox SpellCheck.IsEnabled="True" Name="myTextBox"></TextBox>
    </StackPanel>

</Page>
```

```
using System;
using System.Windows;
using System.Windows.Controls;

namespace SDKSample
{
    public partial class SpellCheckExample : Page
    {
        public SpellCheckExample()
        {
            StackPanel myStackPanel = new StackPanel();

            //Create TextBox
            TextBox myTextBox = new TextBox();
            myTextBox.Width = 200;

            // Enable spellchecking on the TextBox.
            myTextBox.SpellCheck.IsEnabled = true;

            // Alternatively, the SetIsEnabled method could be used
            // to enable or disable spell checking like this:
            // SpellCheck.SetIsEnabled(myTextBox, true);

            myStackPanel.Children.Add(myTextBox);
            this.Content = myStackPanel;
        }
    }
}
```

```
Imports System.Windows
Imports System.Windows.Controls

Namespace SDKSample
    Partial Public Class SpellCheckExample
        Inherits Page
        Public Sub New()
            Dim myStackPanel As New StackPanel()

            'Create TextBox
            Dim myTextBox As New TextBox()
            myTextBox.Width = 200

            ' Enable spellchecking on the TextBox.
            myTextBox.SpellCheck.IsEnabled = True

            ' Alternatively, the SetIsEnabled method could be used
            ' to enable or disable spell checking like this:
            ' SpellCheck.SetIsEnabled(myTextBox, True)

            myStackPanel.Children.Add(myTextBox)
            Me.Content = myStackPanel
        End Sub
    End Class
End Namespace
```

## See also

- [Use Spell Checking with a Context Menu](#)
- [TextBox Overview](#)
- [RichTextBox Overview](#)

# How to: Use a Custom Context Menu with a TextBox

3 minutes to read • [Edit Online](#)

This example shows how to define and implement a simple custom context menu for a [TextBox](#).

## Example

The following Extensible Application Markup Language (XAML) example defines a [TextBox](#) control that includes a custom context menu.

The context menu is defined using a [ContextMenu](#) element. The context menu itself consists of a series of [MenuItem](#) elements and [Separator](#) elements. Each [MenuItem](#) element defines a command in the context menu; the [Header](#) attribute defines the display text for the menu command, and the [Click](#) attribute specifies a handler method for each menu item. The [Separator](#) element simply causes a separating line to be rendered between the previous and subsequent menu items.

```

<TextBox
    Name="c xm TextBox"
    Grid.Row="1"
    AcceptsReturn="True"
    AcceptsTab="True"
    VerticalScrollBarVisibility="Visible"
    TextWrapping="Wrap"
>
<TextBox.ContextMenu>
    <ContextMenu
        Name="c xm"
        Opened="C xm Opened"
    >
        <MenuItem
            Header="Cut"
            Name="c xm Item Cut"
            Click="ClickCut"
        />
        <MenuItem
            Header="Copy"
            Name="c xm Item Copy"
            Click="ClickCopy"
        />
        <MenuItem
            Header="Paste"
            Name="c xm Item Paste"
            Click="ClickPaste"
        />
        <Separator/>
        <MenuItem
            Header="Select All"
            Name="c xm Item Select All"
            Click="ClickSelectAll"
        />
        <MenuItem
            Header="Select Current Line"
            Name="c xm Item Select Line"
            Click="ClickSelectLine"
        />
        <Separator/>
        <MenuItem
            Header="Undo Last Action"
            Name="c xm Item Undo"
            Click="ClickUndo"
        />
        <MenuItem
            Header="Redo Last Action"
            Name="c xm Item Redo"
            Click="ClickRedo"
        />
        <Separator/>
        <MenuItem
            Header="Clear All Text"
            Name="c xm Item Clear"
            Click="ClickClear"
        />
    </ContextMenu>
</TextBox.ContextMenu>
This TextBox uses a simple custom context menu. The context menu can be disabled by checking
the CheckBox above, which simply sets the TextBox.ContextMenu property to null.
</TextBox>

```

## Example

The following example shows the implementation code for the preceding context menu definition, as well as the

code that enables and disables the context menu. The [Opened](#) event is used to dynamically enable or disable certain commands depending on the current state of the [TextBox](#).

To restore the default context menu, use the [ClearValue](#) method to clear the value of the [ContextMenu](#) property. To disable the context menu altogether, set the [ContextMenu](#) property to a null reference ([Nothing](#) in Visual Basic).

```
private void MenuChange(Object sender, RoutedEventArgs args)
{
    RadioButton rb = sender as RadioButton;
    if (rb == null || cxm == null) return;

    switch (rb.Name)
    {
        case "rbCustom":
            cxmTextBox.ContextMenu = cxm;
            break;
        case "rbDefault":
            // Clearing the value of the ContextMenu property
            // restores the default TextBox context menu.
            cxmTextBox.ClearValue(ContextMenuProperty);
            break;
        case "rbDisabled":
            // Setting the ContextMenu property to
            // null disables the context menu.
            cxmTextBox.ContextMenu = null;
            break;
        default:
            break;
    }
}

void ClickPaste(Object sender, RoutedEventArgs args)      { cxmTextBox.Paste(); }
void ClickCopy(Object sender, RoutedEventArgs args)       { cxmTextBox.Copy(); }
void ClickCut(Object sender, RoutedEventArgs args)        { cxmTextBox.Cut(); }
void ClickSelectAll(Object sender, RoutedEventArgs args) { cxmTextBox.SelectAll(); }
void ClickClear(Object sender, RoutedEventArgs args)     { cxmTextBox.Clear(); }
void ClickUndo(Object sender, RoutedEventArgs args)       { cxmTextBox.Undo(); }
void ClickRedo(Object sender, RoutedEventArgs args)       { cxmTextBox.Redo(); }

void ClickSelectLine(Object sender, RoutedEventArgs args)
{
    int lineIndex = cxmTextBox.GetLineIndexFromCharacterIndex(cxmTextBox.CaretIndex);
    int lineStartingCharIndex = cxmTextBox.GetCharacterIndexFromLineIndex(lineIndex);
    int lineLength = cxmTextBox.GetLineLength(lineIndex);
    cxmTextBox.Select(lineStartingCharIndex, lineLength);
}

void CxmOpened(Object sender, RoutedEventArgs args)
{
    // Only allow copy/cut if something is selected to copy/cut.
    if (c xmTextBox.SelectedText == "")
        cxmItemCopy.IsEnabled = cxmItemCut.IsEnabled = false;
    else
        cxmItemCopy.IsEnabled = cxmItemCut.IsEnabled = true;

    // Only allow paste if there is text on the clipboard to paste.
    if (Clipboard.ContainsText())
        cxmItemPaste.IsEnabled = true;
    else
        cxmItemPaste.IsEnabled = false;
}
```

```

Private Sub MenuChange(ByVal sender As Object, ByVal args As RoutedEventArgs)
    Dim rb As RadioButton = CType(sender, RadioButton)
    If myGrid.Children.Contains(cxmTextBox) Then
        Select Case rb.Name
            Case "rbCustom"
                cxmTextBox.ContextMenu = cxm
            Case "rbDefault"
                'Clearing the value of the ContextMenu property
                'restores the default TextBox context menu.
                cxmTextBox.ClearValue(ContextMenuProperty)
            Case "rbDisabled"
                'Setting the ContextMenu property to
                'null disables the context menu.
                cxmTextBox.ContextMenu = Nothing
        End Select
    Else : Return
    End If
End Sub

Private Sub ClickPaste(ByVal sender As Object, ByVal args As RoutedEventArgs)
    cxmTextBox.Paste()
End Sub
Private Sub ClickCopy(ByVal sender As Object, ByVal args As RoutedEventArgs)
    cxmTextBox.Copy()
End Sub
Private Sub ClickCut(ByVal sender As Object, ByVal args As RoutedEventArgs)
    cxmTextBox.Cut()
End Sub
Private Sub ClickSelectAll(ByVal sender As Object, ByVal args As RoutedEventArgs)
    cxmTextBox.SelectAll()
End Sub
Private Sub ClickClear(ByVal sender As Object, ByVal args As RoutedEventArgs)
    cxmTextBox.Clear()
End Sub
Private Sub ClickUndo(ByVal sender As Object, ByVal args As RoutedEventArgs)
    cxmTextBox.Undo()
End Sub
Private Sub ClickRedo(ByVal sender As Object, ByVal args As RoutedEventArgs)
    cxmTextBox.Redo()
End Sub
Private Sub ClickSelectLine(ByVal sender As Object, ByVal args As RoutedEventArgs)
    Dim lineIndex As Integer = cxmTextBox.GetLineIndexFromCharacterIndex(cxmTextBox.CaretIndex)
    Dim lineStartingCharIndex As Integer = cxmTextBox.GetCharacterIndexFromLineIndex(lineIndex)
    Dim lineLength As Integer = cxmTextBox.GetLineLength(lineIndex)
    cxmTextBox.Select(lineStartingCharIndex, lineLength)
End Sub
Private Sub CxmOpened(ByVal sender As Object, ByVal args As RoutedEventArgs)
    'Only allow copy/cut if something is selected to copy/cut.
    If cxmTextBox.SelectedText = "" Then
        cxmItemCopy.IsEnabled = cxmItemCut.IsEnabled = False
    Else
        cxmItemCopy.IsEnabled = cxmItemCut.IsEnabled = True
        'Only allow paste if there is text on the clipboard to paste.
        If Clipboard.ContainsText() Then
            cxmItemPaste.IsEnabled = True
        Else
            cxmItemPaste.IsEnabled = False
        End If
    End If
End Sub

```

## See also

- [Use Spell Checking with a Context Menu](#)
- [TextBox Overview](#)

- [RichTextBox Overview](#)

# How to: Use Spell Checking with a Context Menu

3 minutes to read • [Edit Online](#)

By default, when you enable spell checking in an editing control like [TextBox](#) or [RichTextBox](#), you get spell-checking choices in the context menu. For example, when users right-click a misspelled word, they get a set of spelling suggestions or the option to **Ignore All**. However, when you override the default context menu with your own custom context menu, this functionality is lost, and you need to write code to reenable the spell-checking feature in the context menu. The following example shows how to enable this on a [TextBox](#).

## Example

The following example shows the Extensible Application Markup Language (XAML) that creates a [TextBox](#) with some events that are used to implement the context menu.

```
<Page x:Class="SDKSample.SpellerCustomContextMenu"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      Loaded="OnWindowLoaded">

    <TextBox
        Name="myTextBox"
        TextWrapping="Wrap"
        SpellCheck.IsEnabled="True"
        ContextMenuOpening="tb_ContextMenuOpening">
        In a custum menu you need to write code to add speler choices
        because everything in a custom context menu has to be added explicitly.
    </TextBox>

</Page>
```

## Example

The following example shows the code that implements the context menu.

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace SDKSample
{
    public partial class SpellerCustomContextMenu : Page
    {

        void OnWindowLoaded(object sender, RoutedEventArgs e)
        {
            //This is required for the first time ContextMenu invocation so that TextEditor doesnt handle it.
            myTextBox.ContextMenu = GetContextMenu();
        }

        void tb_ContextMenuOpening(object sender, RoutedEventArgs e)
        {
            int caretIndex, cmdIndex;
            SpellingError spellingError;
```

```

myTextBox.ContextMenu = GetContextMenu();
caretIndex = myTextBox.CaretIndex;

cmdIndex = 0;
spellingError = myTextBox.GetSpellingError(caretIndex);
if (spellingError != null)
{
    foreach (string str in spellingError.Suggestions)
    {
        MenuItem mi = new MenuItem();
        mi.Header = str;
        mi.FontWeight = FontWeights.Bold;
        mi.Command = EditingCommands.CorrectSpellingError;
        mi.CommandParameter = str;
        mi.CommandTarget = myTextBox;
        myTextBox.ContextMenu.Items.Insert(cmdIndex, mi);
        cmdIndex++;
    }
    Separator separatorMenuItem1 = new Separator();
    myTextBox.ContextMenu.Items.Insert(cmdIndex, separatorMenuItem1);
    cmdIndex++;
    MenuItem ignoreAllMI = new MenuItem();
    ignoreAllMI.Header = "Ignore All";
    ignoreAllMI.Command = EditingCommands.IgnoreSpellingError;
    ignoreAllMI.CommandTarget = myTextBox;
    myTextBox.ContextMenu.Items.Insert(cmdIndex, ignoreAllMI);
    cmdIndex++;
    Separator separatorMenuItem2 = new Separator();
    myTextBox.ContextMenu.Items.Insert(cmdIndex, separatorMenuItem2);
}
}

// Gets a fresh context menu.
private ContextMenu GetContextMenu()
{
    ContextMenu cm = new ContextMenu();

    //Can create STATIC custom menu items if exists here...
    MenuItem m1, m2, m3, m4;
    m1 = new MenuItem();
    m1.Header = "File";
    m2 = new MenuItem();
    m2.Header = "Save";
    m3 = new MenuItem();
    m3.Header = "SaveAs";
    m4 = new MenuItem();
    m4.Header = "Recent Files";

    //Can add functionality for the custom menu items here...

    cm.Items.Add(m1);
    cm.Items.Add(m2);
    cm.Items.Add(m3);
    cm.Items.Add(m4);

    return cm;
}
}
}

```

Namespace SDKSample  
Partial Public Class SpellerCustomContextMenu  
Inherits Page

```

Private Sub OnWindowLoaded(ByVal sender As Object, ByVal e As RoutedEventArgs)

```

```

'This is required for the first time ContextMenu invocation
'so that TextEditor doesnt handle it.
myTextBox.ContextMenu = GetContextMenu()
End Sub

Private Sub tb_ContextMenuOpening(ByVal sender As Object,
                                  ByVal e As RoutedEventArgs)

    Dim caretIndex, cmdIndex As Integer
    Dim spellingError As SpellingError

    myTextBox.ContextMenu = GetContextMenu()
    caretIndex = myTextBox.CaretIndex

    cmdIndex = 0
    spellingError = myTextBox.GetSpellingError(caretIndex)
    If spellingError IsNot Nothing Then
        For Each str As String In spellingError.Suggestions
            Dim mi As New MenuItem()
            mi.Header = str
            mi.FontWeight = FontWeights.Bold
            mi.Command = EditingCommands.CorrectSpellingError
            mi.CommandParameter = str
            mi.CommandTarget = myTextBox
            myTextBox.ContextMenu.Items.Insert(cmdIndex, mi)
            cmdIndex += 1
        Next str
        Dim separatorMenuItem1 As New Separator()
        myTextBox.ContextMenu.Items.Insert(cmdIndex, separatorMenuItem1)
        cmdIndex += 1
        Dim ignoreAllMI As New MenuItem()
        ignoreAllMI.Header = "Ignore All"
        ignoreAllMI.Command = EditingCommands.IgnoreSpellingError
        ignoreAllMI.CommandTarget = myTextBox
        myTextBox.ContextMenu.Items.Insert(cmdIndex, ignoreAllMI)
        cmdIndex += 1
        Dim separatorMenuItem2 As New Separator()
        myTextBox.ContextMenu.Items.Insert(cmdIndex, separatorMenuItem2)
    End If
End Sub

' Gets a fresh context menu.
Private Function GetContextMenu() As ContextMenu
    Dim cm As New ContextMenu()

    'Can create STATIC custom menu items if exists here...
    Dim m1, m2, m3, m4 As MenuItem
    m1 = New MenuItem()
    m1.Header = "File"
    m2 = New MenuItem()
    m2.Header = "Save"
    m3 = New MenuItem()
    m3.Header = "SaveAs"
    m4 = New MenuItem()
    m4.Header = "Recent Files"

    'Can add functionality for the custom menu items here...

    cm.Items.Add(m1)
    cm.Items.Add(m2)
    cm.Items.Add(m3)
    cm.Items.Add(m4)

    Return cm
End Function

End Class
End Namespace

```

The code used for doing this with a [RichTextBox](#) is similar. The main difference is in the parameter passed to the `GetSpellingError` method. For a [TextBox](#), pass the integer index of the caret position:

```
spellingError = myTextBox.GetSpellingError(caretIndex);
```

For a [RichTextBox](#), pass the [TextPointer](#) that specifies the caret position:

```
spellingError = myRichTextBox.GetSpellingError(myRichTextBox.CaretPosition);
```

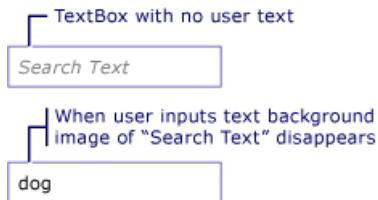
## See also

- [TextBox Overview](#)
- [RichTextBox Overview](#)
- [Enable Spell Checking in a Text Editing Control](#)
- [Use a Custom Context Menu with a TextBox](#)

# How to: Add a Watermark to a TextBox

2 minutes to read • [Edit Online](#)

The following example shows how to aid usability of a [TextBox](#) by displaying an explanatory background image inside of the [TextBox](#) until the user inputs text, at which point the image is removed. In addition, the background image is restored again if the user removes their input. See illustration below.



## NOTE

The reason a background image is used in this example rather than simply manipulating the [Text](#) property of [TextBox](#), is that a background image will not interfere with data binding.

## Example

```
<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.TextBoxBackgroundExample"
>

<StackPanel>
    <TextBox Name="myTextBox" TextChanged="OnTextBoxTextChanged" Width="200">
        <TextBox.Background>
            <ImageBrush ImageSource="TextBoxBackground.gif" AlignmentX="Left" Stretch="None" />
        </TextBox.Background>
    </TextBox>
</StackPanel>
</Page>
```

```
using System;
using System.Windows;
using System.Windows.Input;
using System.Windows.Controls;
using System.Windows.Media;
using System.Windows.Media.Imaging;

namespace SDKSample
{
    public partial class TextBoxBackgroundExample : Page
    {

        void OnTextBoxTextChanged(object sender, TextChangedEventArgs e)
        {

            if (myTextBox.Text == "")
            {
                // Create an ImageBrush.
                ImageBrush textImageBrush = new ImageBrush();
                textImageBrush.ImageSource =
                    new BitmapImage(
                        new Uri(@"TextBoxBackground.gif", UriKind.Relative)
                    );
                textImageBrush.AlignmentX = AlignmentX.Left;
                textImageBrush.Stretch = Stretch.None;
                // Use the brush to paint the button's background.
                myTextBox.Background = textImageBrush;
            }
            else
            {

                myTextBox.Background = null;
            }
        }
    }
}
```

```
Namespace SDKSample
    Partial Public Class TextBoxBackgroundExample
        Inherits Page

        Private Sub OnTextBoxTextChanged(ByVal sender As Object, ByVal e As TextChangedEventArgs)
            If myTextBox.Text = "" Then
                ' Create an ImageBrush.
                Dim textImageBrush As New ImageBrush()

                textImageBrush.ImageSource =
                    New BitmapImage(New Uri("TextBoxBackground.gif", UriKind.Relative))
                textImageBrush.AlignmentX = AlignmentX.Left
                textImageBrush.Stretch = Stretch.None

                ' Use the brush to paint the button's background.
                myTextBox.Background = textImageBrush
            Else
                myTextBox.Background = Nothing
            End If
        End Sub

    End Class

End Namespace
```

## See also

- [TextBox Overview](#)
- [RichTextBox Overview](#)

# ToolBar

2 minutes to read • [Edit Online](#)

The **ToolBar** control is a container for a group of commands or controls that are typically related in their function.

The following illustrations show horizontal and vertical **ToolBar** controls.



Horizontal Toolbar



Vertical Toolbar

## In This Section

[ToolBar Overview](#)

[Style Controls on a ToolBar](#)

## Reference

[ToolBar](#)

[ToolBarTray](#)

## Related Sections

# ToolBar Overview

2 minutes to read • [Edit Online](#)

ToolBar controls are containers for a group of commands or controls which are typically related in their function. A ToolBar usually contains buttons which invoke commands.

## ToolBar Control

The [ToolBar](#) control takes its name from the bar-like arrangement of buttons or other controls into a single row or column. WPF [ToolBar](#) controls provide an overflow mechanism which places any items that do not fit naturally within a size-constrained [ToolBar](#) into a special overflow area. Also, WPF [ToolBar](#) controls are usually used with the related [ToolBarTray](#) control, which provides special layout behavior as well as support for user-initiated sizing and arranging of toolbars.

## Specifying the Position of ToolBars in a ToolBarTray

Use the [Band](#) and [BandIndex](#) properties to position the [ToolBar](#) in the [ToolBarTray](#). [Band](#) indicates the position in which the [ToolBar](#) is placed within its parent [ToolBarTray](#). [BandIndex](#) indicates the order in which the [ToolBar](#) is placed within its band. The following example shows how use this property to place [ToolBar](#) controls inside a [ToolBarTray](#).

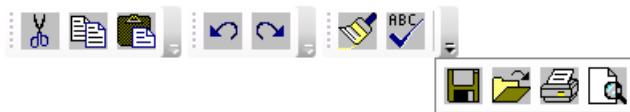
```

<ToolBarTray Background="White">
  <ToolBar Band="1" BandIndex="1">
    <Button>
      <Image Source="toolbargraphics\cut.bmp" />
    </Button>
    <Button>
      <Image Source="toolbargraphics\copy.bmp" />
    </Button>
    <Button>
      <Image Source="toolbargraphics\paste.bmp" />
    </Button>
  </ToolBar>
  <ToolBar Band="2" BandIndex="1">
    <Button>
      <Image Source="toolbargraphics\undo.bmp" />
    </Button>
    <Button>
      <Image Source="toolbargraphics\redo.bmp" />
    </Button>
  </ToolBar>
  <ToolBar Band="2" BandIndex="2">
    <Button>
      <Image Source="toolbargraphics\paint.bmp" />
    </Button>
    <Button>
      <Image Source="toolbargraphics\spell.bmp" />
    </Button>
    <Separator/>
    <Button>
      <Image Source="toolbargraphics\save.bmp" />
    </Button>
    <Button>
      <Image Source="toolbargraphics\open.bmp" />
    </Button>
  </ToolBar>
</ToolBarTray>

```

## ToolBars with Overflow Items

Often [ToolBar](#) controls contain more items than can fit into the toolbar's size. When this happens, the [ToolBar](#) displays an overflow button. To see the overflow items, a user clicks the overflow button and the items are shown in a pop-up window below the [ToolBar](#). The following graphic shows a [ToolBar](#) with overflow items:



You can specify when an item on a toolbar is placed on the overflow panel by setting the [ToolBar.OverflowMode](#) attached property to [OverflowMode.Always](#), [OverflowMode.Never](#), or [OverflowMode.AsNeeded](#). The following example specifies that the last four buttons on the toolbar should always be on the overflow panel.

```

<ToolBarTray Background="White">
    <ToolBar Band="1" BandIndex="1">
        <Button>
            <Image Source="toolbargraphics\cut.bmp" />
        </Button>
        <Button>
            <Image Source="toolbargraphics\copy.bmp" />
        </Button>
        <Button>
            <Image Source="toolbargraphics\paste.bmp" />
        </Button>
        <Button>
            <Image Source="toolbargraphics\undo.bmp" />
        </Button>
        <Button>
            <Image Source="toolbargraphics\redo.bmp" />
        </Button>
        <Button>
            <Image Source="toolbargraphics\paint.bmp" />
        </Button>
        <Button>
            <Image Source="toolbargraphics\spell.bmp" />
        </Button>
        <Separator/>
        <Button ToolBar.OverflowMode="Always">
            <Image Source="toolbargraphics\save.bmp" />
        </Button>
        <Button ToolBar.OverflowMode="Always">
            <Image Source="toolbargraphics\open.bmp" />
        </Button>
        <Button ToolBar.OverflowMode="Always">
            <Image Source="toolbargraphics\print.bmp" />
        </Button>
        <Button ToolBar.OverflowMode="Always">
            <Image Source="toolbargraphics\preview.bmp" />
        </Button>
    </ToolBar>
</ToolBarTray>

```

The [ToolBar](#) uses a [ToolBarPanel](#) and a [ToolBarOverflowPanel](#) in its [ControlTemplate](#). The [ToolBarPanel](#) is responsible for the layout of the items on the toolbar. The [ToolBarOverflowPanel](#) is responsible for the layout of the items that do not fit on the [ToolBar](#). For an example of a [ControlTemplate](#) for a [ToolBar](#), see

[ToolBar Styles and Templates](#).

## See also

- [ToolBarPanel](#)
- [ToolBarOverflowPanel](#)
- [Style Controls on a ToolBar](#)
- [WPF Controls Gallery Sample](#)

# How to: Style Controls on a ToolBar

2 minutes to read • [Edit Online](#)

The [ToolBar](#) defines [ResourceKey](#) objects to specify the style of controls within the [ToolBar](#). To style a control in a [ToolBar](#), set the `x:key` attribute of the style to a [ResourceKey](#) defined in [ToolBar](#).

The [ToolBar](#) defines the following [ResourceKey](#) objects:

- [ButtonStyleKey](#)
- [CheckBoxStyleKey](#)
- [ComboBoxStyleKey](#)
- [MenuStyleKey](#)
- [RadioButtonStyleKey](#)
- [SeparatorStyleKey](#)
- [TextBoxStyleKey](#)
- [ToggleButtonStyleKey](#)

## Example

The following example defines styles for the controls within a [ToolBar](#).

```
<Window.Resources>

    <!--Styles for controls in a toolbar.-->
    <Style x:Key="{x:Static ToolBar.SeparatorStyleKey}" TargetType="Separator">
        <Setter Property="Background" Value="DarkBlue"/>
        <Setter Property="Width" Value="2"/>
    </Style>

    <Style x:Key="{x:Static ToolBar.ButtonStyleKey}" TargetType="Button">
        <Setter Property="Foreground" Value="Blue"/>
        <Setter Property="FontSize" Value="14"/>
        <Setter Property="HorizontalAlignment" Value="Center"/>
        <Setter Property="VerticalAlignment" Value="Center"/>
    </Style>

    <Style x:Key="{x:Static ToolBar.CheckBoxStyleKey}" TargetType="CheckBox">
        <Setter Property="Foreground" Value="DarkSlateBlue"/>
        <Setter Property="FontSize" Value="14"/>
        <Setter Property="HorizontalAlignment" Value="Center"/>
        <Setter Property="VerticalAlignment" Value="Center"/>
    </Style>

    <Style x:Key="{x:Static ToolBar.MenuStyleKey}" TargetType="Menu">
        <Setter Property="FontSize" Value="14"/>
        <Setter Property="FontStyle" Value="Italic"/>
        <Setter Property="FontWeight" Value="Bold"/>
        <Setter Property="Background" Value="LightSteelBlue"/>
    </Style>

    <Style x:Key="{x:Static ToolBar.RadioButtonStyleKey}" TargetType="RadioButton">
        <Setter Property="Background" Value="LightSteelBlue"/>
        <Setter Property="FontSize" Value="14"/>
    </Style>
```

```

<Setter Property="HorizontalAlignment" Value="Center"/>
<Setter Property="VerticalAlignment" Value="Center"/>
</Style>

<Style x:Key="{x:Static ToolBar.TextBoxStyleKey}" TargetType="TextBox">
    <Setter Property="Background" Value="DarkBlue"/>
    <Setter Property="Foreground" Value="White"/>
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="FontStyle" Value="Italic"/>
    <Setter Property="HorizontalAlignment" Value="Center"/>
    <Setter Property="VerticalAlignment" Value="Center"/>
    <Setter Property="Width" Value="75"/>
</Style>

<Style x:Key="{x:Static ToolBar.ComboBoxStyleKey}" TargetType="ComboBox">
    <Setter Property="Background" Value="LightSteelBlue"/>
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="MinWidth" Value="60"/>
    <Setter Property="HorizontalAlignment" Value="Center"/>
    <Setter Property="VerticalAlignment" Value="Center"/>
</Style>
<!--*****Styles for controls that are not in a toolbar.*****-->
<Style TargetType="Separator">
    <Setter Property="Background" Value="DarkBlue"/>
    <Setter Property="Width" Value="2"/>
</Style>

<Style TargetType="Button">
    <Setter Property="Foreground" Value="Blue"/>
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="HorizontalAlignment" Value="Center"/>
    <Setter Property="VerticalAlignment" Value="Center"/>
</Style>

<Style TargetType="CheckBox">
    <Setter Property="Foreground" Value="DarkSlateBlue"/>
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="HorizontalAlignment" Value="Center"/>
    <Setter Property="VerticalAlignment" Value="Center"/>
</Style>

<Style TargetType="Menu">
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="FontStyle" Value="Italic"/>
    <Setter Property="FontWeight" Value="Bold"/>
    <Setter Property="Background" Value="LightSteelBlue"/>
</Style>

<Style TargetType="RadioButton">
    <Setter Property="Background" Value="LightSteelBlue"/>
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="HorizontalAlignment" Value="Center"/>
    <Setter Property="VerticalAlignment" Value="Center"/>
</Style>

<Style TargetType="TextBox">
    <Setter Property="Background" Value="DarkBlue"/>
    <Setter Property="Foreground" Value="White"/>
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="FontStyle" Value="Italic"/>
    <Setter Property="HorizontalAlignment" Value="Center"/>
    <Setter Property="VerticalAlignment" Value="Center"/>
    <Setter Property="Width" Value="75"/>
</Style>

<Style TargetType="ComboBox">
    <Setter Property="Background" Value="LightSteelBlue"/>
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="MinWidth" Value="60"/>

```

```
<Setter Property="HorizontalAlignment" Value="Center"/>
<Setter Property="VerticalAlignment" Value="Center"/>
</Style>
</Window.Resources>
```

```
<ToolBarTray Margin="10,10,3,3"
    Grid.Column="0" Grid.Row="2"
    Background="LightBlue">
<ToolBar >
    <Button Content="Button 1"/>
    <Button Content="Button 2"/>
    <Separator/>
    <CheckBox Content="CheckBox 1"/>
    <CheckBox Content="CheckBox 2"/>
    <Separator/>
    <RadioButton>One</RadioButton>
    <RadioButton>Two</RadioButton>
    <Separator/>
    <ComboBox>
        <ComboBoxItem IsSelected="True">Item 1</ComboBoxItem>
        <ComboBoxItem>Item 2</ComboBoxItem>
        <ComboBoxItem>Item 3</ComboBoxItem>
        <ComboBoxItem>Item 4</ComboBoxItem>
    </ComboBox>
    <TextBox/>
    <Separator/>
    <Menu>
        <MenuItem Header="Menu">
            <MenuItem Header="File">
                <MenuItem Header="Copy"/>
                <MenuItem Header="Paste"/>
            </MenuItem>
        </MenuItem>
    </Menu>
</ToolBar>
</ToolBarTray>
```

## See also

- [Styling and Templating](#)

# ToolTip

2 minutes to read • [Edit Online](#)

A tooltip is a small pop-up window that appears when a user pauses the mouse pointer over an element, such as over a [Button](#).

The following illustration shows a mouse pointer that points to the [CloseButton](#), which then displays its identifying [ToolTip](#).

Close button with its tooltip displayed



## In This Section

[ToolTip Overview](#)

[How-to Topics](#)

## Reference

[ToolTip](#)

[ToolTipService](#)

[Popup](#)

## Related Sections

[Popup Overview](#)

[How-to Topics](#)

# ToolTip Overview

3 minutes to read • [Edit Online](#)

A tooltip is a small pop-up window that appears when a user pauses the mouse pointer over an element, such as over a [Button](#). This topic introduces the tooltip and discusses how to create and customize tooltip content.

## What Is a Tooltip?

When a user moves the mouse pointer over an element that has a tooltip, a window that contains tooltip content (for example, text content that describes the function of a control) appears for a specified amount of time. If the user moves the mouse pointer away from the control, the window disappears because the tooltip content cannot receive focus.

The content of a tooltip can contain one or more lines of text, images, shapes, or other visual content. You define a tooltip for a control by setting one of the following properties to the tooltip content.

- [FrameworkContentElement.ToolTip](#)
- [FrameworkElement.ToolTip](#)

Which property you use depends on whether the control that defines the tooltip inherits from the [FrameworkContentElement](#) or [FrameworkElement](#) class.

## Creating a ToolTip

The following example shows how to create a simple tooltip by setting the [ToolTip](#) property for a [Button](#) control to a text string.

```
<Button ToolTip="Click to submit your information"  
       Click="SubmitCode" Height="20" Width="50">Submit</Button>
```

You can also define a tooltip as a [ToolTip](#) object. The following example uses XAML to specify a [ToolTip](#) object as the tooltip of a [TextBox](#) element. Note that the example specifies the [ToolTip](#) by setting the [FrameworkElement.ToolTip](#) property.

```
<TextBox HorizontalAlignment="Left">ToolTip with non-text content  
<TextBox.ToolTip>  
  <ToolTip>  
    <DockPanel Width="50" Height="70">  
      <Image Source="data\flower.jpg"/>  
      <TextBlock>Useful information goes here.</TextBlock>  
    </DockPanel>  
  </ToolTip>  
</TextBox.ToolTip>  
</TextBox>
```

The following example uses code to generate a [ToolTip](#) object. The example creates a [ToolTip](#) (`tt`) and associates it with a [Button](#).

```
button = new Button();
button.Content = "Hover over me.";
tt = new ToolTip();
tt.Content = "Created with C#";
button.ToolTip = tt;
cv2.Children.Add(button);
```

```
button = New Button()
button.Content = "Hover over me."
tt = New ToolTip()
tt.Content = "Created with Visual Basic"
button.ToolTip = tt
cv2.Children.Add(button)
```

You can also create tooltip content that is not defined as a [ToolTip](#) object by enclosing the tooltip content in a layout element, such as a [DockPanel](#). The following example shows how to set the [ToolTip](#) property of a [TextBox](#) to content that is enclosed in a [DockPanel](#) control.

```
<TextBox>
    ToolTip with image and text
    <TextBox.ToolTip>
        <StackPanel>
            <Image Source="data\flower.jpg"/>
            <TextBlock>Useful information goes here.</TextBlock>
        </StackPanel>
    </TextBox.ToolTip>
```

## Using the Properties of the ToolTip and ToolTipService Classes

You can customize tooltip content by setting visual properties and applying styles. If you define the tooltip content as a [ToolTip](#) object, you can set the visual properties of the [ToolTip](#) object. Otherwise, you must set equivalent attached properties on the [ToolTipService](#) class.

For an example of how to set properties in order to specify the position of tooltip content by using the [ToolTip](#) and [ToolTipService](#) properties, see [Position a ToolTip](#).

## Styling a ToolTip

You can style a [ToolTip](#) by defining a custom [Style](#). The following example defines a [Style](#) called `simple` that shows how to offset the placement of the [ToolTip](#) and change its appearance by setting the [Background](#), [Foreground](#), [FontSize](#), and [FontWeight](#).

```
<Style TargetType="ToolTip">
    <Setter Property = "HorizontalOffset" Value="10"/>
    <Setter Property = "VerticalOffset" Value="10"/>
    <Setter Property = "Background" Value="LightBlue"/>
    <Setter Property = "Foreground" Value="Purple"/>
    <Setter Property = "FontSize" Value="14"/>
    <Setter Property = "FontWeight" Value="Bold"/>
</Style>
```

## Using the Time Interval Properties of ToolTipService

The [ToolTipService](#) class provides the following properties for you to set tooltip display times: [InitialShowDelay](#), [BetweenShowDelay](#), and [ShowDuration](#).

Use the [InitialShowDelay](#) and [ShowDuration](#) properties to specify a delay, typically brief, before a [ToolTip](#) appears and also to specify how long a [ToolTip](#) remains visible. For more information, see [How to: Delay the Display of a ToolTip](#).

The [BetweenShowDelay](#) property determines if tooltips for different controls appear without an initial delay when you move the mouse pointer quickly between them. For more information about the [BetweenShowDelay](#) property, see [Use the BetweenShowDelay Property](#).

The following example shows how to set these properties for a tooltip.

```
<Ellipse Height="25" Width="50"
    Fill="Gray"
    HorizontalAlignment="Left"
    ToolTipService.InitialShowDelay="1000"
    ToolTipService.ShowDuration="7000"
    ToolTipService.BetweenShowDelay="2000">
<Ellipse.ToolTip>
    <ToolTip Placement="Right"
        PlacementRectangle="50,0,0,0"
        HorizontalOffset="10"
        VerticalOffset="20"
        HasDropShadow="false"
        Opened="whenToolTipOpens"
        Closed="whenToolTipCloses"
        >
    <BulletDecorator>
        <BulletDecorator.Bullet>
            <Ellipse Height="10" Width="20" Fill="Blue"/>
        </BulletDecorator.Bullet>
        <TextBlock>Uses the ToolTip Class</TextBlock>
    </BulletDecorator>
    </ToolTip>
</Ellipse.ToolTip>
</Ellipse>
```

## See also

- [ToolTipService](#)
- [ToolTip](#)
- [ToolTipEventArgs](#)
- [ToolTipEventHandler](#)
- [How-to Topics](#)

# ToolTip How-to Topics

2 minutes to read • [Edit Online](#)

## In This Section

[Position a ToolTip](#)

[Use the BetweenShowDelay Property](#)

## Reference

[ToolTip](#)

[ToolTipService](#)

[Popup](#)

## Related Sections

[Popup Overview](#)

[How-to Topics](#)

# How to: Position a ToolTip

3 minutes to read • [Edit Online](#)

This example shows how to specify the position of a tooltip on the screen.

## Example

You can position a tooltip by using a set of five properties that are defined in both the [ToolTip](#) and [ToolTipService](#) classes. The following table shows these two sets of five properties and provides links to their reference documentation according to class.

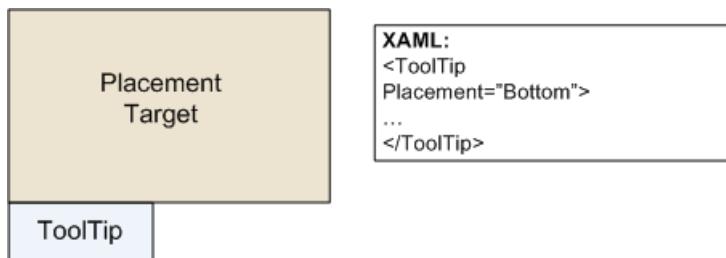
### Corresponding tooltip properties according to class

SYSTEM.WINDOWS.CONTROLS.TOOLTIP CLASS PROPERTIES	SYSTEM.WINDOWS.CONTROLS.TOOLTIPSERVICE CLASS PROPERTIES
ToolTip.Placement	ToolTipService.Placement
ToolTip.PlacementTarget	ToolTipService.PlacementTarget
ToolTip.PlacementRectangle	ToolTipService.PlacementRectangle
ToolTip.HorizontalOffset	ToolTipService.HorizontalOffset
ToolTip.VerticalOffset	ToolTipService.VerticalOffset

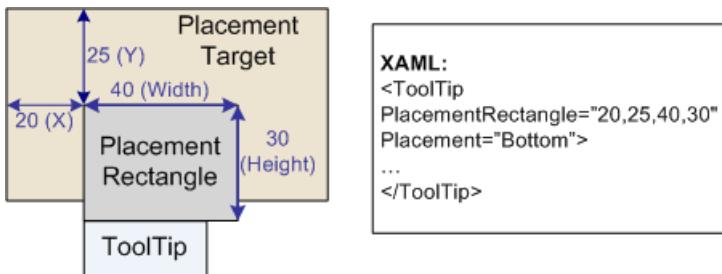
If you define the contents of a tooltip by using a [ToolTip](#) object, you can use the properties of either class; however, the [ToolTipService](#) properties take precedence. Use the [ToolTipService](#) properties for tooltips that are not defined as [ToolTip](#) objects.

The following illustrations show how to position a tooltip by using these properties. Although, the Extensible Application Markup Language (XAML) examples in these illustrations show how to set the properties that are defined by the [ToolTip](#) class, the corresponding properties of the [ToolTipService](#) class follow the same layout rules. For more information about the possible values for the Placement property, see [Popup Placement Behavior](#).

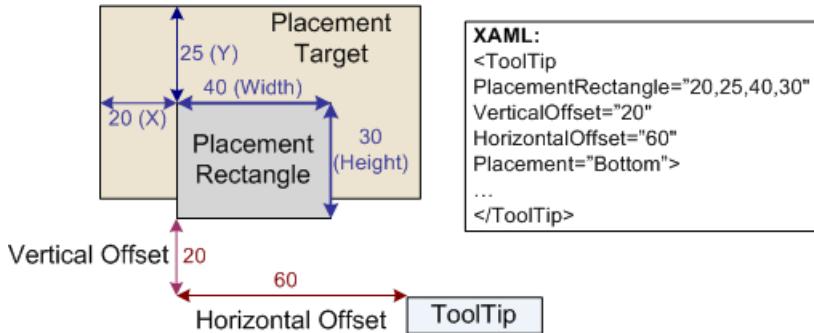
The following image shows tooltip placement by using the Placement property:



The following image shows tooltip placement by using the Placement and PlacementRectangle properties:



The following image shows tooltip placement by using the Placement, PlacementRectangle, and Offset properties:



The following example shows how to use the [ToolTip](#) properties to specify the position of a tooltip whose content is a [ToolTip](#) object.

```
<Ellipse Height="25" Width="50"
    Fill="Gray"
    HorizontalAlignment="Left"
    ToolTipService.InitialShowDelay="1000"
    ToolTipService.ShowDuration="7000"
    ToolTipService.BetweenShowDelay="2000">
    <Ellipse.ToolTip>
        <ToolTip Placement="Right"
            PlacementRectangle="50,0,0,0"
            HorizontalOffset="10"
            VerticalOffset="20"
            HasDropShadow="false"
            Opened="whenToolTipOpens"
            Closed="whenToolTipCloses"
            >
            <BulletDecorator>
                <BulletDecorator.Bullet>
                    <Ellipse Height="10" Width="20" Fill="Blue"/>
                </BulletDecorator.Bullet>
                <TextBlock>Uses the ToolTip Class</TextBlock>
            </BulletDecorator>
        </ToolTip>
    </Ellipse.ToolTip>
</Ellipse>
```

```

//Create an ellipse that will have a
//ToolTip control.
Ellipse ellipse1 = new Ellipse();
ellipse1.Height = 25;
ellipse1.Width = 50;
ellipse1.Fill = Brushes.Gray;
ellipse1.HorizontalAlignment = HorizontalAlignment.Left;

//Create a tooltip and set its position.
ToolTip tooltip = new ToolTip();
tooltip.Placement = PlacementMode.Right;
tooltip.PlacementRectangle = new Rect(50, 0, 0, 0);
tooltip.HorizontalOffset = 10;
tooltip.VerticalOffset = 20;

//Create BulletDecorator and set it
//as the tooltip content.
BulletDecorator bdec = new BulletDecorator();
Ellipse littleEllipse = new Ellipse();
littleEllipse.Height = 10;
littleEllipse.Width = 20;
littleEllipse.Fill = Brushes.Blue;
bdec.Bullet = littleEllipse;
TextBlock tipText = new TextBlock();
tipText.Text = "Uses the ToolTip class";
bdec.Child = tipText;
tooltip.Content = bdec;

//set tooltip on ellipse
ellipse1.ToolTip = tooltip;

```

```

'Create an ellipse that will have a
'ToolTip control.
Dim ellipse1 As New Ellipse()
ellipse1.Height = 25
ellipse1.Width = 50
ellipse1.Fill = Brushes.Gray
ellipse1.HorizontalAlignment = HorizontalAlignment.Left

'Create a tooltip and set its position.
Dim tooltip As New ToolTip()
tooltip.Placement = PlacementMode.Right
tooltip.PlacementRectangle = New Rect(50, 0, 0, 0)
tooltip.HorizontalOffset = 10
tooltip.VerticalOffset = 20

'Create BulletDecorator and set it
'as the tooltip content.
Dim bdec As New BulletDecorator()
Dim littleEllipse As New Ellipse()
littleEllipse.Height = 10
littleEllipse.Width = 20
littleEllipse.Fill = Brushes.Blue
bdec.Bullet = littleEllipse
Dim tipText As New TextBlock()
tipText.Text = "Uses the ToolTip class"
bdec.Child = tipText
tooltip.Content = bdec

'set tooltip on ellipse
ellipse1.ToolTip = tooltip

```

The following example shows how to use the [ToolTipService](#) properties to specify the position of a tooltip whose content is not a [ToolTip](#) object.

```
<Ellipse Height="25" Width="50"
    Fill="Gray"
    HorizontalAlignment="Left"
    ToolTipService.InitialShowDelay="1000"
    ToolTipService.ShowDuration="7000"
    ToolTipService.BetweenShowDelay="2000"
    ToolTipService.Placement="Right"
    ToolTipService.PlacementRectangle="50,0,0,0"
    ToolTipService.HorizontalOffset="10"
    ToolTipService.VerticalOffset="20"
    ToolTipService.HasDropShadow="false"
    ToolTipService.ShowOnDisabled="true"
    ToolTipService.IsEnabled="true"
    ToolTipOpening="whenToolTipOpens"
    ToolTipClosing="whenToolTipCloses"
    >
<Ellipse.ToolTip>
    <BulletDecorator>
        <BulletDecorator.Bullet>
            <Ellipse Height="10" Width="20" Fill="Blue"/>
        </BulletDecorator.Bullet>
        <TextBlock>Uses the ToolTipService class</TextBlock>
    </BulletDecorator>
</Ellipse.ToolTip>
</Ellipse>
```

```

//Create and Ellipse with the BulletDecorator as
//the tooltip
Ellipse ellipse2 = new Ellipse();
ellipse2.Name = "ellipse2";
this.RegisterName(ellipse2.Name, ellipse2);
ellipse2.Height = 25;
ellipse2.Width = 50;
ellipse2.Fill = Brushes.Gray;
ellipse2.HorizontalAlignment = HorizontalAlignment.Left;

//set tooltip timing
ToolTipService.SetInitialShowDelay(ellipse2, 1000);
ToolTipService.SetBetweenShowDelay(ellipse2, 2000);
ToolTipService.SetShowDuration(ellipse2, 7000);

//set tooltip placement

ToolTipService.SetPlacement(ellipse2, PlacementMode.Right);

ToolTipService.SetPlacementRectangle(ellipse2,
    new Rect(50, 0, 0, 0));

ToolTipService.SetHorizontalOffset(ellipse2, 10.0);

ToolTipService.SetVerticalOffset(ellipse2, 20.0);

ToolTipService.SetHasDropShadow(ellipse2, false);

ToolTipService.SetIsEnabled(ellipse2, true);

ToolTipService.SetShowOnDisabled(ellipse2, true);

ellipse2.AddHandler(ToolTipService.ToolTipOpeningEvent,
    new RoutedEventHandler(whenToolTipOpens));
ellipse2.AddHandler(ToolTipService.ToolTipClosingEvent,
    new RoutedEventHandler(whenToolTipCloses));

//define tooltip content
BulletDecorator bdec2 = new BulletDecorator();
Ellipse littleEllipse2 = new Ellipse();
littleEllipse2.Height = 10;
littleEllipse2.Width = 20;
littleEllipse2.Fill = Brushes.Blue;
bdec2.Bullet = littleEllipse2;
TextBlock tipText2 = new TextBlock();
tipText2.Text = "Uses the ToolTipService class";
bdec2.Child = tipText2;
ToolTipService.SetToolTip(ellipse2, bdec2);
stackPanel_1_2.Children.Add(ellipse2);

```

```

'Create and Ellipse with the BulletDecorator as
'the tooltip
Dim ellipse2 As New Ellipse()
ellipse2.Name = "ellipse2"
Me.RegisterName(ellipse2.Name, ellipse2)
ellipse2.Height = 25
ellipse2.Width = 50
ellipse2.Fill = Brushes.Gray
ellipse2.HorizontalAlignment = HorizontalAlignment.Left

'set tooltip timing
ToolTipService.SetInitialShowDelay(ellipse2, 1000)
ToolTipService.SetBetweenShowDelay(ellipse2, 2000)
ToolTipService.SetShowDuration(ellipse2, 7000)

'set tooltip placement

ToolTipService.SetPlacement(ellipse2, PlacementMode.Right)

ToolTipService.SetPlacementRectangle(ellipse2, New Rect(50, 0, 0, 0))

ToolTipService.SetHorizontalOffset(ellipse2, 10.0)

ToolTipService.SetVerticalOffset(ellipse2, 20.0)

ToolTipService.SetHasDropShadow(ellipse2, False)

ToolTipService.SetIsEnabled(ellipse2, True)

ToolTipService.SetShowOnDisabled(ellipse2, True)

ellipse2.AddHandler(ToolTipService.ToolTipOpeningEvent, New RoutedEventHandler(AddressOf whenToolTipOpens))
ellipse2.AddHandler(ToolTipService.ToolTipClosingEvent, New RoutedEventHandler(AddressOf whenToolTipCloses))

'define tooltip content
Dim bdec2 As New BulletDecorator()
Dim littleEllipse2 As New Ellipse()
littleEllipse2.Height = 10
littleEllipse2.Width = 20
littleEllipse2.Fill = Brushes.Blue
bdec2.Bullet = littleEllipse2
Dim tipText2 As New TextBlock()
tipText2.Text = "Uses the ToolTipService class"
bdec2.Child = tipText2
ToolTipService.SetToolTip(ellipse2, bdec2)
stackPanel_1_2.Children.Add(ellipse2)

```

## See also

- [ToolTip](#)
- [ToolTipService](#)
- [How-to Topics](#)
- [ToolTip Overview](#)

# How to: Use the BetweenShowDelay Property

2 minutes to read • [Edit Online](#)

This example shows how to use the [BetweenShowDelay](#) time property so that tooltips appear quickly—with little or no delay—when a user moves the mouse pointer from one tooltip directly to another.

## Example

In the following example, the [InitialShowDelay](#) property is set to one second (1000 milliseconds) and the [BetweenShowDelay](#) is set to two seconds (2000 milliseconds) for the tooltips of both [Ellipse](#) controls. If you display the tooltip for one of the ellipses and then move the mouse pointer to another ellipse within two seconds and pause on it, the tooltip of the second ellipse displays immediately.

In either of the following scenarios, the [InitialShowDelay](#) applies, which causes the tooltip for the second ellipse to wait one second before it appears:

- If the time it takes to move to the second button is more than two seconds.
- If the tooltip is not visible at the beginning of the time interval for the first ellipse.

```
<Ellipse Height="25" Width="50"
    Fill="Gray"
    HorizontalAlignment="Left"
    ToolTipService.InitialShowDelay="1000"
    ToolTipService.ShowDuration="7000"
    ToolTipService.BetweenShowDelay="2000">
<Ellipse.ToolTip>
<ToolTip Placement="Right"
    PlacementRectangle="50,0,0,0"
    HorizontalOffset="10"
    VerticalOffset="20"
    HasDropShadow="false"
    Opened="whenToolTipOpens"
    Closed="whenToolTipCloses"
    >
<BulletDecorator>
<BulletDecorator.Bullet>
<Ellipse Height="10" Width="20" Fill="Blue"/>
</BulletDecorator.Bullet>
<TextBlock>Uses the ToolTip Class</TextBlock>
</BulletDecorator>
</ToolTip>
</Ellipse.ToolTip>
</Ellipse>
```

```
<Ellipse Height="25" Width="50"
    Fill="Gray"
    HorizontalAlignment="Left"
    ToolTipService.InitialShowDelay="1000"
    ToolTipService.ShowDuration="7000"
    ToolTipService.BetweenShowDelay="2000"
    ToolTipService.Placement="Right"
    ToolTipService.PlacementRectangle="50,0,0,0"
    ToolTipService.HorizontalOffset="10"
    ToolTipService.VerticalOffset="20"
    ToolTipService.HasDropShadow="false"
    ToolTipService.ShowOnDisabled="true"
    ToolTipService.IsEnabled="true"
    ToolTipOpening="whenToolTipOpens"
    ToolTipClosing="whenToolTipCloses"
    >
<Ellipse.ToolTip>
    <BulletDecorator>
        <BulletDecorator.Bullet>
            <Ellipse Height="10" Width="20" Fill="Blue"/>
        </BulletDecorator.Bullet>
        <TextBlock>Uses the ToolTipService class</TextBlock>
    </BulletDecorator>
</Ellipse.ToolTip>
</Ellipse>
```

## See also

- [ToolTip](#)
- [ToolTipService](#)
- [How-to Topics](#)
- [ToolTip Overview](#)

# TreeView

2 minutes to read • [Edit Online](#)

The [TreeView](#) control displays information in a hierarchical structure by using collapsible nodes.

The following illustration is an example of a [TreeView](#) control that has nested [TreeViewItem](#) controls:

- Employee1
  - Jesper Aaberg
  - Employee Number
    - 12345
  - Work Days
    - Monday
    - Tuesday
    - Thursday
- ⊕ Employee2

## In This Section

[TreeView Overview](#)

[How-to Topics](#)

## Reference

[TreeView](#)

[TreeViewItem](#)

## Related Sections

[Data Binding Overview](#)

[Data Templating Overview](#)

# TreeView Overview

3 minutes to read • [Edit Online](#)

The [TreeView](#) control provides a way to display information in a hierarchical structure by using collapsible nodes. This topic introduces the [TreeView](#) and [TreeViewItem](#) controls, and provides simple examples of their use.

## What Is a TreeView?

[TreeView](#) is an [ItemsControl](#) that nests the items by using [TreeViewItem](#) controls. The following example creates a [TreeView](#).

```
<TreeView Name="myTreeViewEvent" >
  <TreeViewItem Header="Employee1" IsSelected="True">
    <TreeViewItem Header="Jesper Aaberg"/>
    <TreeViewItem Header="Employee Number">
      <TreeViewItem Header="12345"/>
    </TreeViewItem>
    <TreeViewItem Header="Work Days">
      <TreeViewItem Header="Monday"/>
      <TreeViewItem Header="Tuesday"/>
      <TreeViewItem Header="Thursday"/>
    </TreeViewItem>
  </TreeViewItem>
  <TreeViewItem Header="Employee2">
    <TreeViewItem Header="Dominik Paiha"/>
    <TreeViewItem Header="Employee Number">
      <TreeViewItem Header="98765"/>
    </TreeViewItem>
    <TreeViewItem Header="Work Days">
      <TreeViewItem Header="Tuesday"/>
      <TreeViewItem Header="Wednesday"/>
      <TreeViewItem Header="Friday"/>
    </TreeViewItem>
  </TreeViewItem>
</TreeView>
```

## Creating a TreeView

The [TreeView](#) control contains a hierarchy of [TreeViewItem](#) controls. A [TreeViewItem](#) control is a [HeaderedItemsControl](#) that has a [Header](#) and an [Items](#) collection.

If you are defining a [TreeView](#) by using Extensible Application Markup Language (XAML), you can explicitly define the [Header](#) content of a [TreeViewItem](#) control and the items that make up its collection. The previous illustration demonstrates this method.

You can also specify an [ItemsSource](#) as a data source and then specify a [HeaderTemplate](#) and [ItemTemplate](#) to define the [TreeViewItem](#) content.

To define the layout of a [TreeViewItem](#) control, you can also use [HierarchicalDataTemplate](#) objects. For more information and an example, see [Use SelectedValue, SelectedValuePath, and SelectedItem](#).

If an item is not a [TreeViewItem](#) control, it is automatically enclosed by a [TreeViewItem](#) control when the [TreeView](#) control is displayed.

## Expanding and Collapsing a TreeViewItem

If the user expands a [TreeViewItem](#), the [IsExpanded](#) property is set to `true`. You can also expand or collapse a [TreeViewItem](#) without any direct user action by setting the [IsExpanded](#) property to `true` (expand) or `false` (collapse). When this property changes, an [Expanded](#) or [Collapsed](#) event occurs.

When the [BringIntoView](#) method is called on a [TreeViewItem](#) control, the [TreeViewItem](#) and its parent [TreeViewItem](#) controls expand. If a [TreeViewItem](#) is not visible or partially visible, the [TreeView](#) scrolls to make it visible.

## TreeViewItem Selection

When a user clicks a [TreeViewItem](#) control to select it, the [Selected](#) event occurs, and its [IsSelected](#) property is set to `true`. The [TreeViewItem](#) also becomes the [SelectedItem](#) of the [TreeView](#) control. Conversely, when the selection changes from a [TreeViewItem](#) control, its [Unselected](#) event occurs and its [IsSelected](#) property is set to `false`.

The [SelectedItem](#) property on the [TreeView](#) control is a read-only property; hence, you cannot explicitly set it. The [SelectedItem](#) property is set if the user clicks on a [TreeViewItem](#) control or when the [IsSelected](#) property is set to `true` on the [TreeViewItem](#) control.

Use the [SelectedValuePath](#) property to specify a [SelectedValue](#) of a [SelectedItem](#). For more information, see [Use SelectedValue, SelectedValuePath, and SelectedItem](#).

You can register an event handler on the [SelectedItemChanged](#) event in order to determine when a selected [TreeViewItem](#) changes. The [RoutedPropertyChangedEventArgs<T>](#) that is provided to the event handler specifies the [OldValue](#), which is the previous selection, and the [NewValue](#), which is the current selection. Either value can be `null` if the application or user has not made a previous or current selection.

## TreeView Style

The default style for a [TreeView](#) control places it inside a [StackPanel](#) object that contains a [ScrollViewer](#) control. When you set the [Width](#) and [Height](#) properties for a [TreeView](#), these values are used to size the [StackPanel](#) object that displays the [TreeView](#). If the content to display is larger than the display area, a [ScrollViewer](#) automatically displays so that the user can scroll through the [TreeView](#) content.

To customize the appearance of a [TreeViewItem](#) control, set the [Style](#) property to a custom [Style](#).

The following example shows how to set the [Foreground](#) and [FontSize](#) property values for a [TreeViewItem](#) control by using a [Style](#).

```
<Style TargetType="{x:Type TreeViewItem}">
    <Setter Property="Foreground" Value="Blue"/>
    <Setter Property="FontSize" Value="12"/>
</Style>
```

## Adding Images and Other Content to TreeView Items

You can include more than one object in the [Header](#) content of a [TreeViewItem](#). To include multiple objects in [Header](#) content, enclose the objects inside a layout control, such as a [Panel](#) or [StackPanel](#).

The following example shows how to define the [Header](#) of a [TreeViewItem](#) as a [CheckBox](#) and [TextBlock](#) that are both enclosed in a [DockPanel](#) control.

```
<TreeViewItem>
  <TreeViewItem.Header>
    <DockPanel>
      <CheckBox/>
      <TextBlock>
        TreeViewItem Text
      </TextBlock>
    </DockPanel>
  </TreeViewItem.Header>
</TreeViewItem>
```

The following example shows how to define a [DataTemplate](#) that contains an [Image](#) and a [TextBlock](#) that are enclosed in a [DockPanel](#) control. You can use a [DataTemplate](#) to set the [HeaderTemplate](#) or [ItemTemplate](#) for a [TreeViewItem](#).

```
<DataTemplate x:Key="NewspaperTVItem">
  <DockPanel>
    <Image Source="images\icon.jpg"/>
    <TextBlock VerticalAlignment="center" Text ="{Binding Path=Name}"/>
  </DockPanel>
</DataTemplate>
```

## See also

- [TreeView](#)
- [TreeViewItem](#)
- [How-to Topics](#)
- [WPF Content Model](#)

# TreeView How-to Topics

2 minutes to read • [Edit Online](#)

The topics in this section describe how to use the [TreeView](#) control to display information in a hierarchical structure.

## In This Section

[Create Simple or Complex TreeViews](#)

[Use SelectedValue, SelectedValuePath, and SelectedItem](#)

[Bind a TreeView to Data That Has an Indeterminable Depth](#)

[Improve the Performance of a TreeView](#)

[Find a TreeViewItem in a TreeView](#)

## Reference

[TreeView](#)

[TreeViewItem](#)

## Related Sections

# How to: Create Simple or Complex TreeViews

2 minutes to read • [Edit Online](#)

This example shows how to create simple or complex [TreeView](#) controls.

A [TreeView](#) consists of a hierarchy of [TreeViewItem](#) controls, which can contain simple text strings and also more complex content, such as [Button](#) controls or a [StackPanel](#) with embedded content. You can explicitly define the [TreeView](#) content or a data source can provide the content. This topic provides examples of these concepts.

## Example

The [Header](#) property of the [TreeViewItem](#) contains the content that the [TreeView](#) displays for that item. A [TreeViewItem](#) can also have [TreeViewItem](#) controls as its child elements and you can define these child elements by using the [Items](#) property.

The following example shows how to explicitly define [TreeViewItem](#) content by setting the [Header](#) property to a text string.

```
<TreeView>
  <TreeViewItem Header="Employee1">
    <TreeViewItem Header="Jesper"/>
    <TreeViewItem Header="Aaberg"/>
    <TreeViewItem Header="12345"/>
  </TreeViewItem>
  <TreeViewItem Header="Employee2">
    <TreeViewItem Header="Dominik"/>
    <TreeViewItem Header="Paiha"/>
    <TreeViewItem Header="98765"/>
  </TreeViewItem>
</TreeView>
```

The following example show how to define child elements of a [TreeViewItem](#) by defining [Items](#) that are [Button](#) controls.

```
<TreeView>
  <TreeViewItem Header = "Employee1">
    <TreeViewItem.Items>
      <Button>Jesper</Button>
      <Button>Aaberg</Button>
      <Button>12345</Button>
    </TreeViewItem.Items>
  </TreeViewItem>
  <TreeViewItem Header="Employee2">
    <TreeViewItem.Items>
      <Button>Dominik</Button>
      <Button>Paiha</Button>
      <Button>98765</Button>
    </TreeViewItem.Items>
  </TreeViewItem>
</TreeView>
```

The following example shows how to create a [TreeView](#) where an [XmlDataProvider](#) provides [TreeViewItem](#) content and a [HierarchicalDataTemplate](#) defines the appearance of the content.

```

<XmlDataProvider x:Key="myEmployeeData" XPath="/EmployeeData">
    <x:XData>
        <EmployeeData xmlns="">
            <EmployeeInfo>
                <EmployeeInfoData>Employee1</EmployeeInfoData>
                <Item Type="FirstName">Jesper</Item>
                <Item Type="LastName">Aaberg</Item>
                <Item Type="EmployeeNumber">12345</Item>
            </EmployeeInfo>
            <EmployeeInfo>
                <EmployeeInfoData>Employee2</EmployeeInfoData>
                <Item Type="FirstName">Dominik</Item>
                <Item Type="LastName">Paiha</Item>
                <Item Type="EmployeeNumber">98765</Item>
            </EmployeeInfo>
        </EmployeeData>
    </x:XData>
</XmlDataProvider>

```

```

<HierarchicalDataTemplate DataType="EmployeeInfo"
    ItemsSource = "{Binding XPath=Item}">
    <TextBlock Text="{Binding XPath=EmployeeInfoData}" />
</HierarchicalDataTemplate>

```

```

<TreeView ItemsSource="{Binding Source={StaticResource myEmployeeData},
    XPath=EmployeeInfo}"/>

```

The following example shows how to create a [TreeView](#) where the [TreeViewItem](#) content contains [DockPanel](#) controls that have embedded content.

```

<TreeView>
    <TreeViewItem Header="Animals">
        <TreeViewItem.Items>
            <DockPanel>
                <Image Source="data\fish.png"/>
                <TextBlock Margin="5" Foreground="Brown"
                    FontSize="12">Fish</TextBlock>
            </DockPanel>
            <DockPanel>
                <Image Source="data\dog.png"/>
                <TextBlock Margin="5" Foreground="Brown"
                    FontSize="12">Dog</TextBlock>
            </DockPanel>
            <DockPanel>
                <Image Source="data\cat.png"/>
                <TextBlock Margin="5" Foreground="Brown"
                    FontSize="12">Cat</TextBlock>
            </DockPanel>
        </TreeViewItem.Items>
    </TreeViewItem>
</TreeView>

```

## See also

- [TreeView](#)
- [TreeViewItem](#)
- [TreeView Overview](#)
- [How-to Topics](#)



# How to: Use SelectedValue, SelectedValuePath, and SelectedItem

2 minutes to read • [Edit Online](#)

This example shows how to use the [SelectedValue](#) and [SelectedValuePath](#) properties to specify a value for the [SelectedItem](#) of a [TreeView](#).

## Example

The [SelectedValuePath](#) property provides a way to specify a [SelectedValue](#) for the [SelectedItem](#) in a [TreeView](#). The [SelectedItem](#) represents an object in the [Items](#) collection and the [TreeView](#) displays the value of a single property of the selected item. The [SelectedValuePath](#) property specifies the path to the property that is used to determine the value of the [SelectedValue](#) property. The examples in this topic illustrate this concept.

The following example shows an [XmlDataProvider](#) that contains employee information.

```
<XmlDataProvider x:Key="myEmployeeData" XPath="/EmployeeData">
  <x:XData>
    <EmployeeData xmlns="">
      <EmployeeInfo>
        <EmployeeName>Jesper Aabergy</EmployeeName>
        <EmployeeWorkDay>Monday</EmployeeWorkDay>
        <EmployeeWorkDay>Wednesday</EmployeeWorkDay>
        <EmployeeWorkDay>Friday</EmployeeWorkDay>
        <EmployeeStartTime>8:00am</EmployeeStartTime>
        <EmployeeNumber>12345</EmployeeNumber>
      </EmployeeInfo>
      <EmployeeInfo>
        <EmployeeName>Dominik Paiha</EmployeeName>
        <EmployeeWorkDay>Monday</EmployeeWorkDay>
        <EmployeeWorkDay>Tuesday</EmployeeWorkDay>
        <EmployeeStartTime>6:30am</EmployeeStartTime>
        <EmployeeNumber>98765</EmployeeNumber>
      </EmployeeInfo>
    </EmployeeData>
  </x:XData>
</XmlDataProvider>
```

The following example defines a [HierarchicalDataTemplate](#) that displays the [EmployeeName](#) and [EmployeeWorkDay](#) of the [Employee](#). Note that the [HierarchicalDataTemplate](#) does not specify the [EmployeeNumber](#) as part of the template.

```
<HierarchicalDataTemplate x:Key="SampleTemplate" DataType="EmployeeInfo"
  ItemsSource ="{Binding XPath=EmployeeWorkDay}">
  <TextBlock Text="{Binding XPath=EmployeeName}" />
</HierarchicalDataTemplate>
```

The following example shows a [TreeView](#) that uses the previously defined [HierarchicalDataTemplate](#) and that sets the [SelectedValue](#) property to the [EmployeeNumber](#). When you select an [EmployeeName](#) in the [TreeView](#), the [SelectedItem](#) property returns the [EmployeeInfo](#) data item that corresponds to the selected [EmployeeName](#). However, because the [SelectedValuePath](#) of this [TreeView](#) is set to [EmployeeNumber](#), the [SelectedValue](#) is set to the [EmployeeNumber](#).

```
<TreeView ItemsSource="{Binding Source={StaticResource myEmployeeData},  
ItemTemplate={StaticResource SampleTemplate},  
XPath=EmployeeInfo}"  
Name="myTreeView"  
SelectedValuePath="EmployeeNumber"  
/>  
  
<TextBlock Margin="10">SelectedValuePath: </TextBlock>  
<TextBlock Margin="10,0,0,0"  
Text="{Binding ElementName=myTreeView,  
Path=SelectedValuePath}"  
Foreground="Blue"/>  
  
<TextBlock Margin="10">SelectedValue: </TextBlock>  
<TextBlock Margin="10,0,0,0"  
Text="{Binding ElementName=myTreeView,  
Path=SelectedValue}"  
Foreground="Blue"/>
```

## See also

- [TreeView](#)
- [TreeViewItem](#)
- [TreeView Overview](#)
- [How-to Topics](#)

# How to: Bind a TreeView to Data That Has an Indeterminable Depth

2 minutes to read • [Edit Online](#)

There might be times when you want to bind a [TreeView](#) to a data source whose depth is not known. This can occur when the data is recursive in nature, such as a file system, where folders can contain folders, or a company's organizational structure, where employees have other employees as direct reports.

The data source must have a hierarchical object model. For example, an [Employee](#) class might contain a collection of Employee objects that are the direct reports of an employee. If the data is represented in a way that is not hierarchical, you must build a hierarchical representation of the data.

When you set the [ItemsControl.ItemTemplate](#) property and if the [ItemsControl](#) generates an [ItemsControl](#) for each child item, then the child [ItemsControl](#) uses the same [ItemTemplate](#) as the parent. For example, if you set the [ItemTemplate](#) property on a data-bound [TreeView](#), each [TreeViewItem](#) that is generated uses the [DataTemplate](#) that was assigned to the [ItemTemplate](#) property of the [TreeView](#).

The [HierarchicalDataTemplate](#) enables you to specify the [ItemsSource](#) for a [TreeViewItem](#), or any [HeaderedItemsControl](#), on the data template. When you set the [HierarchicalDataTemplate.ItemsSource](#) property, that value is used when the [HierarchicalDataTemplate](#) is applied. By using a [HierarchicalDataTemplate](#), you can recursively set the [ItemsSource](#) for each [TreeViewItem](#) in the [TreeView](#).

## Example

The following example demonstrates how to bind a [TreeView](#) to hierarchical data and use a [HierarchicalDataTemplate](#) to specify the [ItemsSource](#) for each [TreeViewItem](#). The [TreeView](#) binds to XML data that represents the employees in a company. Each [Employee](#) element can contain other [Employee](#) elements to indicate who reports to whom. Because the data is recursive, the [HierarchicalDataTemplate](#) can be applied to each level.

```

<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Page.Resources>
        <XmlDataProvider x:Key="myCompany" XPath="Company/Employee">
            <x:XData>
                <Company xmlns="">
                    <Employee Name="Don Hall">
                        <Employee Name="Alice Ciccu">
                            <Employee Name="David Pelton">
                                <Employee Name="Vivian Atlas"/>
                            </Employee>
                            <Employee Name="Jeff Price"/>
                            <Employee Name="Andy Jacobs"/>
                        </Employee>
                    <Employee Name="Bill Malone">
                        <Employee Name="Maurice Taylor"/>
                        <Employee Name="Sunil Uppal"/>
                        <Employee Name="Qiang Wang"/>
                    </Employee>
                </Employee>
            </Company>
        </x:XData>
    </XmlDataProvider>

    <!-- Bind the HierarchicalDataTemplate.ItemsSource property to the employees under
         each Employee element. -->
    <HierarchicalDataTemplate x:Key="EmployeeTemplate"
        ItemsSource="{Binding XPath=Employee}">
        <TextBlock Text="{Binding XPath=@Name}" />
    </HierarchicalDataTemplate>

    <Style TargetType="TreeViewItem">
        <Setter Property="IsExpanded" Value="True"/>
    </Style>
</Page.Resources>

<Grid>
    <TreeView ItemsSource="{Binding Source={StaticResource myCompany}}"
        ItemTemplate="{StaticResource EmployeeTemplate}"/>
</Grid>
</Page>

```

## See also

- [Data Binding Overview](#)
- [Data Templating Overview](#)

# How to: Improve the Performance of a TreeView

2 minutes to read • [Edit Online](#)

If a [TreeView](#) contains many items, the amount of time it takes to load may cause a significant delay in the user interface. You can improve the load time by setting the `VirtualizingStackPanel.IsVirtualizing` attached property to `true`. The UI might also be slow to react when a user scrolls the [TreeView](#) by using the mouse wheel or dragging the thumb of a scrollbar. You can improve the performance of the [TreeView](#) when the user scrolls by setting the `VirtualizingStackPanel.VirtualizationMode` attached property to [VirtualizationMode.Recycling](#).

## Example

## Description

The following example creates a [TreeView](#) that sets the `VirtualizingStackPanel.IsVirtualizing` attached property to true and the `VirtualizingStackPanel.VirtualizationMode` attached property to [VirtualizationMode.Recycling](#) to optimize its performance.

## Code

```

<StackPanel>
    <StackPanel.Resources>
        <src:TreeViewData x:Key="dataItems"/>

        <HierarchicalDataTemplate DataType="{x:Type src:ItemsForTreeView}"
            ItemsSource="{Binding Path=SecondLevelItems}">

            <!--Display the TopLevelName property in the first level.-->
            <TextBlock Text="{Binding Path=TopLevelName}" />

            <!--Display each string in the SecondLevelItems property in
                the second level.-->
            <HierarchicalDataTemplate.ItemTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding}" />
                </DataTemplate>
            </HierarchicalDataTemplate.ItemTemplate>

            <!--Set the foreground of the items in the second level
                to Navy.-->
            <HierarchicalDataTemplate.ItemContainerStyle>
                <Style TargetType="TreeViewItem">
                    <Setter Property="Foreground" Value="Navy" />
                </Style>
            </HierarchicalDataTemplate.ItemContainerStyle>
        </HierarchicalDataTemplate>
    </StackPanel.Resources>

    <TreeView Height="200"
        ItemsSource="{Binding Source={StaticResource dataItems}}"
        VirtualizingStackPanel.IsVirtualizing="True"
        VirtualizingStackPanel.VirtualizationMode="Recycling">
        <TreeView.ItemContainerStyle>

            <!--Expand each TreeViewItem in the first level and
                set its foreground to Green.-->
            <Style TargetType="TreeViewItem">
                <Setter Property="IsExpanded" Value="True" />
                <Setter Property="Foreground" Value="Green" />
            </Style>
        </TreeView.ItemContainerStyle>
    </TreeView>
</StackPanel>

```

The following example shows the data that the previous example uses.

```
public class TreeViewData : ObservableCollection<ItemsForTreeView>
{
    public TreeViewData()
    {
        for (int i = 0; i < 100; ++i)
        {
            ItemsForTreeView item = new ItemsForTreeView();
            item.TopLevelName = "item " + i.ToString();
            Add(item);
        }
    }
}

public class ItemsForTreeView
{
    public string TopLevelName { get; set; }
    private ObservableCollection<string> level2Items;

    public ObservableCollection<string> SecondLevelItems
    {
        get
        {
            level2Items ??= new ObservableCollection<string>();
            return level2Items;
        }
    }

    public ItemsForTreeView()
    {
        for (int i = 0; i < 10; ++i)
        {
            SecondLevelItems.Add("Second Level " + i.ToString());
        }
    }
}
```

```

Public Class TreeViewData
    Inherits ObservableCollection(Of ItemsForTreeView)

    Public Sub New()
        For i As Integer = 0 To 99
            Dim item As New ItemsForTreeView()
            item.TopLevelName = "item " & i.ToString()
            Add(item)
        Next
    End Sub
End Class

Public Class ItemsForTreeView
    Private _TopLevelName As String
    Public Property TopLevelName() As String
        Get
            Return _TopLevelName
        End Get
        Set(ByVal value As String)
            _TopLevelName = value
        End Set
    End Property
    Private level2Items As ObservableCollection(Of String)

    Public ReadOnly Property SecondLevelItems() As ObservableCollection(Of String)
        Get
            If level2Items Is Nothing Then
                level2Items = New ObservableCollection(Of String)()
            End If
            Return level2Items
        End Get
    End Property

    Public Sub New()
        For i As Integer = 0 To 9
            SecondLevelItems.Add("Second Level " & i.ToString())
        Next
    End Sub
End Class

```

## See also

- [Controls](#)

# How to: Find a TreeViewItem in a TreeView

6 minutes to read • [Edit Online](#)

The [TreeView](#) control provides a convenient way to display hierarchical data. If your [TreeView](#) is bound to a data source, the [SelectedItem](#) property provides a convenient way for you to quickly retrieve the selected data object. It is typically best to work with the underlying data object, but sometimes you may need to programmatically manipulate the data's containing [TreeViewItem](#). For example, you may need to programmatically expand the [TreeViewItem](#), or select a different item in the [TreeView](#).

To find a [TreeViewItem](#) that contains a specific data object, you must traverse each level of the [TreeView](#). The items in a [TreeView](#) can also be virtualized to improve performance. In the case where items might be virtualized, you also must realize a [TreeViewItem](#) to check whether it contains the data object.

## Example

### Description

The following example searches a [TreeView](#) for a specific object and returns the object's containing [TreeViewItem](#). The example ensures that each [TreeViewItem](#) is instantiated so that its child items can be searched. This example also works if the [TreeView](#) does not use virtualized items.

#### NOTE

The following example works for any [TreeView](#), regardless of the underlying data model, and searches every [TreeViewItem](#) until the object is found. Another technique that has better performance is to search the data model for the specified object, keep track of its location within the data hierarchy, and then find the corresponding [TreeViewItem](#) in the [TreeView](#). However, the technique that has better performance requires knowledge of the data model and cannot be generalized for any given [TreeView](#).

## Code

```
/// <summary>
/// Recursively search for an item in this subtree.
/// </summary>
/// <param name="container">
///   The parent ItemsControl. This can be a TreeView or a TreeViewItem.
/// </param>
/// <param name="item">
///   The item to search for.
/// </param>
/// <returns>
///   The TreeViewItem that contains the specified item.
/// </returns>
private TreeViewItem GetTreeViewItem(ItemsControl container, object item)
{
    if (container != null)
    {
        if (container.DataContext == item)
        {
            return container as TreeViewItem;
        }

        // Expand the current container
        if (container is TreeViewItem && !((TreeViewItem)container).IsExpanded)
```

```

{
    container.SetValue(TreeViewItem.IsExpandedProperty, true);
}

// Try to generate the ItemsPresenter and the ItemsPanel.
// by calling ApplyTemplate. Note that in the
// virtualizing case even if the item is marked
// expanded we still need to do this step in order to
// regenerate the visuals because they may have been virtualized away.

container.ApplyTemplate();
ItemsPresenter itemsPresenter =
    (ItemsPresenter)container.Template.FindName("ItemsHost", container);
if (itemsPresenter != null)
{
    itemsPresenter.ApplyTemplate();
}
else
{
    // The Tree template has not named the ItemsPresenter,
    // so walk the descendants and find the child.
    itemsPresenter = FindVisualChild<ItemsPresenter>(container);
    if (itemsPresenter == null)
    {
        container.UpdateLayout();

        itemsPresenter = FindVisualChild<ItemsPresenter>(container);
    }
}

Panel itemsHostPanel = (Panel)VisualTreeHelper.GetChild(itemsPresenter, 0);

// Ensure that the generator for this panel has been created.
UIElementCollection children = itemsHostPanel.Children;

MyVirtualizingStackPanel virtualizingPanel =
    itemsHostPanel as MyVirtualizingStackPanel;

for (int i = 0, count = container.Items.Count; i < count; i++)
{
    TreeViewItem subContainer;
    if (virtualizingPanel != null)
    {
        // Bring the item into view so
        // that the container will be generated.
        virtualizingPanel.BringIntoView(i);

        subContainer =
            (TreeViewItem)container.ItemContainerGenerator.
                ContainerFromIndex(i);
    }
    else
    {
        subContainer =
            (TreeViewItem)container.ItemContainerGenerator.
                ContainerFromIndex(i);

        // Bring the item into view to maintain the
        // same behavior as with a virtualizing panel.
        subContainer.BringIntoView();
    }

    if (subContainer != null)
    {
        // Search the next level for the object.
        TreeViewItem resultContainer = GetTreeViewItem(subContainer, item);
        if (resultContainer != null)
        {
            return resultContainer;
        }
    }
}

```

```

        }
    else
    {
        // The object is not under this TreeViewItem
        // so collapse it.
        subContainer.IsExpanded = false;
    }
}

return null;
}

/// <summary>
/// Search for an element of a certain type in the visual tree.
/// </summary>
/// <typeparam name="T">The type of element to find.</typeparam>
/// <param name="visual">The parent element.</param>
/// <returns></returns>
private T FindVisualChild<T>(Visual visual) where T : Visual
{
    for (int i = 0; i < VisualTreeHelper.GetChildrenCount(visual); i++)
    {
        Visual child = (Visual)VisualTreeHelper.GetChild(visual, i);
        if (child != null)
        {
            T correctlyTyped = child as T;
            if (correctlyTyped != null)
            {
                return correctlyTyped;
            }

            T descendant = FindVisualChild<T>(child);
            if (descendant != null)
            {
                return descendant;
            }
        }
    }

    return null;
}

```

```

''' <summary>
''' Recursively search for an item in this subtree.
''' </summary>
''' <param name="container">
''' The parent ItemsControl. This can be a TreeView or a TreeViewItem.
''' </param>
''' <param name="item">
''' The item to search for.
''' </param>
''' <returns>
''' The TreeViewItem that contains the specified item.
''' </returns>
Private Function GetTreeViewItem(ByVal container As ItemsControl,
                                  ByVal item As Object) As TreeViewItem

If container IsNot Nothing Then
    If container.DataContext Is item Then
        Return TryCast(container, TreeViewItem)
    End If

    ' Expand the current container
    If TypeOf container Is TreeViewItem AndAlso
        Not DirectCast(container, TreeViewItem).IsExpanded Then

```

```

        container.SetValue(TreeViewItem.IsExpandedProperty, True)
End If

' Try to generate the ItemsPresenter and the ItemsPanel.
' by calling ApplyTemplate. Note that in the
' virtualizing case, even if IsExpanded = true,
' we still need to do this step in order to
' regenerate the visuals because they may have been virtualized away.
container.ApplyTemplate()

Dim itemsPresenter As ItemsPresenter =
    DirectCast(container.Template.FindName("ItemsHost", container), ItemsPresenter)

If itemsPresenter IsNot Nothing Then
    itemsPresenter.ApplyTemplate()
Else
    ' The Tree template has not named the ItemsPresenter,
    ' so walk the descendants and find the child.
    itemsPresenter = FindVisualChild(Of ItemsPresenter)(container)

    If itemsPresenter Is Nothing Then
        container.UpdateLayout()

        itemsPresenter = FindVisualChild(Of ItemsPresenter)(container)
    End If
End If

Dim itemsHostPanel As Panel =
    DirectCast(VisualTreeHelper.GetChild(itemsPresenter, 0), Panel)

' Do this to ensure that the generator for this panel has been created.
Dim children As UIElementCollection = itemsHostPanel.Children

Dim virtualizingPanel As MyVirtualizingStackPanel =
    TryCast(itemsHostPanel, MyVirtualizingStackPanel)

For index As Integer = 0 To container.Items.Count - 1

    Dim subContainer As TreeViewItem

    If virtualizingPanel IsNot Nothing Then

        ' Bring the item into view so
        ' that the container will be generated.
        virtualizingPanel.BringIntoView(index)

        subContainer =
            DirectCast(container.ItemContainerGenerator.ContainerFromIndex(index),
                      TreeViewItem)
    Else
        subContainer =
            DirectCast(container.ItemContainerGenerator.ContainerFromIndex(index),
                      TreeViewItem)

        ' Bring the item into view to maintain the
        ' same behavior as with a virtualizing panel.
        subContainer.BringIntoView()
    End If

    If subContainer IsNot Nothing Then

        ' Search the next level for the object.
        Dim resultContainer As TreeViewItem =
            GetTreeViewItem(subContainer, item)

        If resultContainer IsNot Nothing Then

```

```

    If resultContainer IsNot Nothing Then
        Return resultContainer
    Else
        ' The object is not under this TreeViewItem
        ' so collapse it.
        subContainer.IsExpanded = False
    End If
End If

Next
End If

Return Nothing
End Function

''' <summary>
''' Search for an element of a certain type in the visual tree.
''' </summary>
''' <typeparam name="T">The type of element to find.</typeparam>
''' <param name="visual">The parent element.</param>
''' <returns></returns>
Private Function FindVisualChild(Of T As Visual)(ByVal visual As Visual) As T

    For i As Integer = 0 To VisualTreeHelper.GetChildrenCount(visual) - 1

        Dim child As Visual = DirectCast(VisualTreeHelper.GetChild(visual, i), Visual)

        If child IsNot Nothing Then

            Dim correctlyTyped As T = TryCast(child, T)
            If correctlyTyped IsNot Nothing Then
                Return correctlyTyped
            End If

            Dim descendent As T = FindVisualChild(Of T)(child)
            If descendent IsNot Nothing Then
                Return descendent
            End If
        End If
    Next

    Return Nothing
End Function

```

The previous code relies on a custom [VirtualizingStackPanel](#) that exposes a method named `BringIntoView`. The following code defines the custom [VirtualizingStackPanel](#).

```

public class MyVirtualizingStackPanel : VirtualizingStackPanel
{
    /// <summary>
    /// Publically expose BringIndexIntoView.
    /// </summary>
    public void BringIndexIntoView(int index)
    {

        this.BringIndexIntoView(index);
    }
}

```

```
Public Class MyVirtualizingStackPanel
    Inherits VirtualizingStackPanel
    ''' <summary>
    ''' Publically expose BringIndexIntoView.
    ''' </summary>
    Public Overloads Sub BringIntoView(ByVal index As Integer)

        Me.BringIndexIntoView(index)
    End Sub
End Class
```

The following XAML shows how to create a [TreeView](#) that uses the custom [VirtualizingStackPanel](#).

```
<TreeView VirtualizingStackPanel.IsVirtualizing="True">

    <!--Use the custom class MyVirtualizingStackPanel
        as the ItemsPanel for the TreeView and
        TreeViewItem object.-->
    <TreeView.ItemsPanel>
        <ItemsPanelTemplate>
            <src:MyVirtualizingStackPanel/>
        </ItemsPanelTemplate>
    </TreeView.ItemsPanel>
    <TreeView.ItemContainerStyle>
        <Style TargetType="TreeViewItem">
            <Setter Property="ItemsPanel">
                <Setter.Value>
                    <ItemsPanelTemplate>
                        <src:MyVirtualizingStackPanel/>
                    </ItemsPanelTemplate>
                </Setter.Value>
            </Setter>
        </Style>
    </TreeView.ItemContainerStyle>
</TreeView>
```

## See also

- [Improve the Performance of a TreeView](#)

# WrapPanel

2 minutes to read • [Edit Online](#)

The [WrapPanel](#) element positions child elements in sequential position from left to right, breaking content to the next line at the edge of its containing box.

## Reference

[Panel](#)

[Canvas](#)

[DockPanel](#)

[Grid](#)

[StackPanel](#)

[VirtualizingStackPanel](#)

[WrapPanel](#)

## Related Sections

[Layout](#)

[Walkthrough: My first WPF desktop application](#)

[ScrollViewer Overview](#)

# Viewbox

2 minutes to read • [Edit Online](#)

The [Viewbox](#) control is used to stretch or scale a child element.

## In This Section

[Apply Stretch Properties to the Contents of a Viewbox](#)

## Reference

[Viewbox](#)

[Image](#)

## See also

- [WPF Controls Gallery Sample](#)

# How to: Apply Stretch Properties to the Contents of a Viewbox

2 minutes to read • [Edit Online](#)

## Example

This example shows how to change the value of the [StretchDirection](#) and [Stretch](#) properties of a [Viewbox](#).

The first example uses Extensible Application Markup Language (XAML) to define a [Viewbox](#) element. It assigns a [MaxWidth](#) and [MaxHeight](#) of 400. The example nests an [Image](#) element within the [Viewbox](#). [Button](#) elements that correspond to the property values for the [Stretch](#) and [StretchDirection](#) enumerations manipulate the stretching behavior of the nested [Image](#).

```
<StackPanel Margin="0,0,0,10" HorizontalAlignment="Center" Orientation="Horizontal" DockPanel.Dock="Top">
    <Button Name="btn1" Click="stretchNone">Stretch="None"</Button>
    <Button Name="btn2" Click="stretchFill">Stretch="Fill"</Button>
    <Button Name="btn3" Click="stretchUni">Stretch="Uniform"</Button>
    <Button Name="btn4" Click="stretchUniFill">Stretch="UniformToFill"</Button>
</StackPanel>

<StackPanel Margin="0,0,0,10" HorizontalAlignment="Center" Orientation="Horizontal" DockPanel.Dock="Top">
    <Button Name="btn5" Click="sdUpOnly">StretchDirection="UpOnly"</Button>
    <Button Name="btn6" Click="sdDownOnly">StretchDirection="DownOnly"</Button>
    <Button Name="btn7" Click="sdBoth">StretchDirection="Both"</Button>
</StackPanel>

<TextBlock DockPanel.Dock="Top" Name="txt1" />
<TextBlock DockPanel.Dock="Top" Name="txt2" />

<StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
    <Viewbox MaxWidth="500" MaxHeight="500" Name="vb1">
        <Image Source="tulip_farm.jpg"/>
    </Viewbox>
</StackPanel>
```

The following code-behind file handles the [Button Click](#) events that the previous XAML example defines.

```
private void stretchNone(object sender, RoutedEventArgs e)
{
    vb1.Stretch = System.Windows.Media.Stretch.None;
    txt1.Text = "Stretch is now set to None.";
}

private void stretchFill(object sender, RoutedEventArgs e)
{
    vb1.Stretch = System.Windows.Media.Stretch.Fill;
    txt1.Text = "Stretch is now set to Fill.";
}

private void stretchUni(object sender, RoutedEventArgs e)
{
    vb1.Stretch = System.Windows.Media.Stretch.Uniform;
    txt1.Text = "Stretch is now set to Uniform.";
}

private void stretchUniFill(object sender, RoutedEventArgs e)
{
    vb1.Stretch = System.Windows.Media.Stretch.UniformToFill;
    txt1.Text = "Stretch is now set to UniformToFill.";
}

private void sdUpOnly(object sender, RoutedEventArgs e)
{
    vb1.StretchDirection = System.Windows.Controls.StretchDirection.UpOnly;
    txt2.Text = "StretchDirection is now UpOnly.";
}

private void sdDownOnly(object sender, RoutedEventArgs e)
{
    vb1.StretchDirection = System.Windows.Controls.StretchDirection.DownOnly;
    txt2.Text = "StretchDirection is now DownOnly.";
}

private void sdBoth(object sender, RoutedEventArgs e)
{
    vb1.StretchDirection = System.Windows.Controls.StretchDirection.Both;
    txt2.Text = "StretchDirection is now Both.";
}
```

```
Private Sub stretchNone(ByVal sender As Object, ByVal args As RoutedEventArgs)

    vb1.Stretch = System.Windows.Media.Stretch.None
    txt1.Text = "Stretch is now set to None."
End Sub

Private Sub stretchFill(ByVal sender As Object, ByVal args As RoutedEventArgs)

    vb1.Stretch = System.Windows.Media.Stretch.Fill
    txt1.Text = "Stretch is now set to Fill."
End Sub

Private Sub stretchUni(ByVal sender As Object, ByVal args As RoutedEventArgs)

    vb1.Stretch = System.Windows.Media.Stretch.Uniform
    txt1.Text = "Stretch is now set to Uniform."
End Sub

Private Sub stretchUniFill(ByVal sender As Object, ByVal args As RoutedEventArgs)

    vb1.Stretch = System.Windows.Media.Stretch.UniformToFill
    txt1.Text = "Stretch is now set to UniformToFill."
End Sub

Private Sub sdUpOnly(ByVal sender As Object, ByVal args As RoutedEventArgs)

    vb1.StretchDirection = System.Windows.Controls.StretchDirection.UpOnly
    txt2.Text = "StretchDirection is now UpOnly."
End Sub

Private Sub sdDownOnly(ByVal sender As Object, ByVal args As RoutedEventArgs)

    vb1.StretchDirection = System.Windows.Controls.StretchDirection.DownOnly
    txt2.Text = "StretchDirection is now DownOnly."
End Sub

Private Sub sdBoth(ByVal sender As Object, ByVal args As RoutedEventArgs)

    vb1.StretchDirection = System.Windows.Controls.StretchDirection.Both
    txt2.Text = "StretchDirection is now Both."
End Sub
```

## See also

- [Viewbox](#)
- [Stretch](#)
- [StretchDirection](#)

# Styles and Templates

2 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) styling and templating refer to a suite of features (styles, templates, triggers, and storyboards) that allow an application, document, or user interface (UI) designer to create visually compelling applications and to standardize on a particular look for their product.

## In This Section

[Styling and Templating](#)

[How to: Find ControlTemplate-Generated Elements](#)

## Reference

[Style](#)

[ControlTemplate](#)

[DataTemplate](#)

## Related Sections

[Advanced](#)

[Control Customization](#)

[Graphics and Multimedia](#)

2 minutes to read

# Create a template for a control

7 minutes to read • [Edit Online](#)

With Windows Presentation Foundation (WPF), you can customize an existing control's visual structure and behavior with your own reusable template. Templates can be applied globally to your application, windows and pages, or directly to controls. Most scenarios that require you to create a new control can be covered by instead creating a new template for an existing control.

## IMPORTANT

The Desktop Guide documentation is under construction.

In this article, you'll explore creating a new [ControlTemplate](#) for the [Button](#) control.

## When to create a ControlTemplate

Controls have many properties, such as [Background](#), [Foreground](#), and [FontFamily](#). These properties control different aspects of the control's appearance, but the changes that you can make by setting these properties are limited. For example, you can set the [Foreground](#) property to blue and [FontStyle](#) to italic on a [CheckBox](#). When you want to customize the control's appearance beyond what setting the other properties on the control can do, you create a [ControlTemplate](#).

In most user interfaces, a button has the same general appearance: a rectangle with some text. If you wanted to create a rounded button, you could create a new control that inherits from the button or recreates the functionality of the button. In addition, the new user control would provide the circular visual.

You can avoid creating new controls by customizing the visual layout of an existing control. With a rounded button, you create a [ControlTemplate](#) with the desired visual layout.

On the other hand, if you need a control with new functionality, different properties, and new settings, you would create a new [UserControl](#).

## Prerequisites

Create a new WPF application and in *MainWindow.xaml* (or another window of your choice) set the following properties on the **<Window>** element:

<b>Title</b>	Template Intro Sample
<b>SizeToContent</b>	WidthAndHeight
<b>MinWidth</b>	250

Set the content of the **<Window>** element to the following XAML:

```

<StackPanel Margin="10">
    <Label>Unstyled Button</Label>
    <Button>Button 1</Button>
    <Label>Rounded Button</Label>
    <Button>Button 2</Button>
</StackPanel>

```

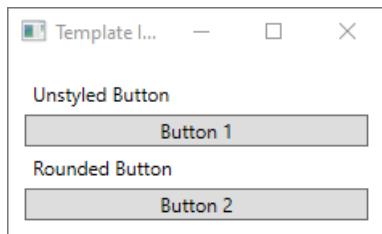
In the end, the *MainWindow.xaml* file should look similar to the following:

```

<Window x:Class="IntroToStylingAndTemplating.Window1"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:IntroToStylingAndTemplating"
        mc:Ignorable="d"
        Title="Template Intro Sample" SizeToContent="WidthAndHeight" MinWidth="250">
    <StackPanel Margin="10">
        <Label>Unstyled Button</Label>
        <Button>Button 1</Button>
        <Label>Rounded Button</Label>
        <Button>Button 2</Button>
    </StackPanel>
</Window>

```

If you run the application, it looks like the following:



## Create a ControlTemplate

The most common way to declare a [ControlTemplate](#) is as a resource in the `Resources` section in a XAML file. Because templates are resources, they obey the same scoping rules that apply to all resources. Put simply, where you declare a template affects where the template can be applied. For example, if you declare the template in the root element of your application definition XAML file, the template can be used anywhere in your application. If you define the template in a window, only the controls in that window can use the template.

To start with, add a `Window.Resources` element to your *MainWindow.xaml* file:

```

<Window x:Class="IntroToStylingAndTemplating.Window2"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:local="clr-namespace:IntroToStylingAndTemplating"
    mc:Ignorable="d"
    Title="Template Intro Sample" SizeToContent="WidthAndHeight" MinWidth="250">
    <Window.Resources>

        </Window.Resources>
        <StackPanel Margin="10">
            <Label>Unstyled Button</Label>
            <Button>Button 1</Button>
            <Label>Rounded Button</Label>
            <Button>Button 2</Button>
        </StackPanel>
    </Window>

```

Create a new **<ControlTemplate>** with the following properties set:

<b>x:Key</b>	roundbutton
<b>TargetType</b>	Button

This control template will be simple:

- a root element for the control, a **Grid**
- an **Ellipse** to draw the rounded appearance of the button
- a **ContentPresenter** to display the user-specified button content

```

<ControlTemplate x:Key="roundbutton" TargetType="Button">
    <Grid>
        <Ellipse Fill="{TemplateBinding Background}" Stroke="{TemplateBinding Foreground}" />
        <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center" />
    </Grid>
</ControlTemplate>

```

## TemplateBinding

When you create a new **ControlTemplate**, you still might want to use the public properties to change the control's appearance. The **TemplateBinding** markup extension binds a property of an element that is in the **ControlTemplate** to a public property that is defined by the control. When you use a **TemplateBinding**, you enable properties on the control to act as parameters to the template. That is, when a property on a control is set, that value is passed on to the element that has the **TemplateBinding** on it.

## Ellipse

Notice that the **Fill** and **Stroke** properties of the **<Ellipse>** element are bound to the control's **Foreground** and **Background** properties.

## ContentPresenter

A **<ContentPresenter>** element is also added to the template. Because this template is designed for a button, take into consideration that the button inherits from **ContentControl**. The button presents the content of the element. You can set anything inside of the button, such as plain text or even another control. Both of the following are valid buttons:

```

<Button>My Text</Button>

<!-- and -->

<Button>
    <CheckBox>Checkbox in a button</CheckBox>
</Button>

```

In both of the previous examples, the text and the checkbox are set as the `Button.Content` property. Whatever is set as the content can be presented through a `<ContentPresenter>`, which is what the template does.

If the `ControlTemplate` is applied to a `ContentControl` type, such as a `Button`, a `ContentPresenter` is searched for in the element tree. If the `ContentPresenter` is found, the template automatically binds the control's `Content` property to the `ContentPresenter`.

## Use the template

Find the buttons that were declared at the start of this article.

```

<StackPanel Margin="10">
    <Label>Unstyled Button</Label>
    <Button>Button 1</Button>
    <Label>Rounded Button</Label>
    <Button>Button 2</Button>
</StackPanel>

```

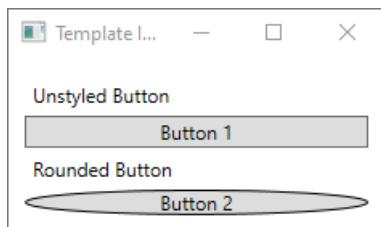
Set the second button's `Template` property to the `roundbutton` resource:

```

<StackPanel Margin="10">
    <Label>Unstyled Button</Label>
    <Button>Button 1</Button>
    <Label>Rounded Button</Label>
    <Button Template="{StaticResource roundbutton}">Button 2</Button>
</StackPanel>

```

If you run the project and look at the result, you'll see that the button has a rounded background.

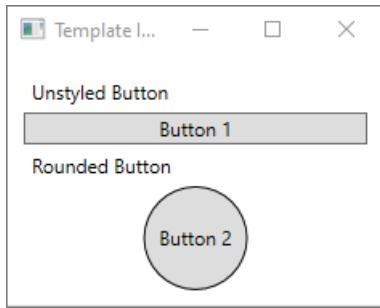


You may have noticed that the button isn't a circle but is skewed. Because of the way the `<Ellipse>` element works, it always expands to fill the available space. Make the circle uniform by changing the button's `width` and `height` properties to the same value:

```

<StackPanel Margin="10">
    <Label>Unstyled Button</Label>
    <Button>Button 1</Button>
    <Label>Rounded Button</Label>
    <Button Template="{StaticResource roundbutton}" Width="65" Height="65">Button 2</Button>
</StackPanel>

```



## Add a Trigger

Even though a button with a template applied looks different, it behaves the same as any other button. If you press the button, the [Click](#) event fires. However, you may have noticed that when you move your mouse over the button, the button's visuals don't change. These visual interactions are all defined by the template.

With the dynamic event and property systems that WPF provides, you can watch a specific property for a value and then restyle the template when appropriate. In this example, you'll watch the button's [IsMouseOver](#) property. When the mouse is over the control, style the [<Ellipse>](#) with a new color. This type of trigger is known as a *PropertyTrigger*.

For this to work, you'll need to add a name to the [<Ellipse>](#) that you can reference. Give it the name of **backgroundElement**.

```
<Ellipse x:Name="backgroundElement" Fill="{TemplateBinding Background}" Stroke="{TemplateBinding Foreground}" />
```

Next, add a new [Trigger](#) to the [ControlTemplate.Triggers](#) collection. The trigger will watch the [IsMouseOver](#) event for the value `true`.

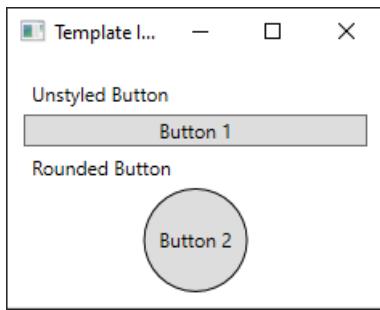
```
<ControlTemplate x:Key="roundbutton" TargetType="Button">
    <Grid>
        <Ellipse x:Name="backgroundElement" Fill="{TemplateBinding Background}" Stroke="{TemplateBinding Foreground}" />
        <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center" />
    </Grid>
    <ControlTemplate.Triggers>
        <Trigger Property="IsMouseOver" Value="true">

        </Trigger>
    </ControlTemplate.Triggers>
</ControlTemplate>
```

Next, add a [<Setter>](#) to the [<Trigger>](#) that changes the [Fill](#) property of the [<Ellipse>](#) to a new color.

```
<Trigger Property="IsMouseOver" Value="true">
    <Setter Property="Fill" TargetName="backgroundElement" Value="AliceBlue"/>
</Trigger>
```

Run the project. Notice that when you move the mouse over the button, the color of the [<Ellipse>](#) changes.



## Use a VisualState

Visual states are defined and triggered by a control. For example, when the mouse is moved on top of the control, the `CommonStates.MouseOver` state is triggered. You can animate property changes based on the current state of the control. In the previous section, a `<PropertyTrigger>` was used to change the foreground of the button to `AliceBlue` when the `IsMouseOver` property was `true`. Instead, create a visual state that animates the change of this color, providing a smooth transition. For more information about `VisualStates`, see [Styles and templates in WPF](#).

To convert the `<PropertyTrigger>` to an animated visual state, First, remove the `<ControlTemplate.Triggers>` element from your template.

```
<ControlTemplate x:Key="roundbutton" TargetType="Button">
    <Grid>
        <Ellipse x:Name="backgroundElement" Fill="{TemplateBinding Background}" Stroke="{TemplateBinding Foreground}" />
        <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center" />
    </Grid>
</ControlTemplate>
```

Next, in the `<Grid>` root of the control template, add the `<VisualStateManager.VisualStateGroups>` element with a `<VisualStateGroup>` for `CommonStates`. Define two states, `Normal` and `MouseOver`.

```
<ControlTemplate x:Key="roundbutton" TargetType="Button">
    <Grid>
        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup Name="CommonStates">
                <VisualState Name="Normal">
                </VisualState>
                <VisualState Name="MouseOver">
                </VisualState>
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>
        <Ellipse x:Name="backgroundElement" Fill="{TemplateBinding Background}" Stroke="{TemplateBinding Foreground}" />
        <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center" />
    </Grid>
</ControlTemplate>
```

Any animations defined in a `<VisualState>` are applied when that state is triggered. Create animations for each state. Animations are put inside of a `<Storyboard>` element. For more information about storyboards, see [Storyboards Overview](#).

- Normal

This state animates the ellipse fill, restoring it to the control's `Background` color.

```

<Storyboard>
    <ColorAnimation Storyboard.TargetName="backgroundElement"
        Storyboard.TargetProperty="(Shape.Fill).(SolidColorBrush.Color)"
        To="{TemplateBinding Background}"
        Duration="0:0:0.3"/>
</Storyboard>

```

- MouseOver

This state animates the ellipse `Background` color to a new color: `Yellow`.

```

<Storyboard>
    <ColorAnimation Storyboard.TargetName="backgroundElement"
        Storyboard.TargetProperty="(Shape.Fill).(SolidColorBrush.Color)"
        To="Yellow"
        Duration="0:0:0.3"/>
</Storyboard>

```

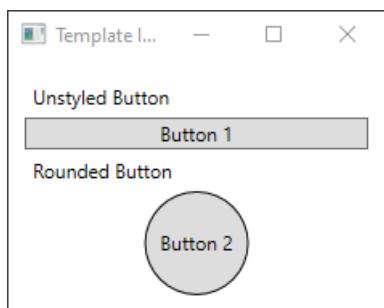
The `<ControlTemplate>` should now look like the following.

```

<ControlTemplate x:Key="roundbutton" TargetType="Button">
    <Grid>
        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup Name="CommonStates">
                <VisualState Name="Normal">
                    <Storyboard>
                        <ColorAnimation Storyboard.TargetName="backgroundElement"
                            Storyboard.TargetProperty="(Shape.Fill).(SolidColorBrush.Color)"
                            To="{TemplateBinding Background}"
                            Duration="0:0:0.3"/>
                    </Storyboard>
                </VisualState>
                <VisualState Name="MouseOver">
                    <Storyboard>
                        <ColorAnimation Storyboard.TargetName="backgroundElement"
                            Storyboard.TargetProperty="(Shape.Fill).(SolidColorBrush.Color)"
                            To="Yellow"
                            Duration="0:0:0.3"/>
                    </Storyboard>
                </VisualState>
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>
        <Ellipse Name="backgroundElement" Fill="{TemplateBinding Background}" Stroke="{TemplateBinding Foreground}" />
        <ContentPresenter x:Name="contentPresenter" HorizontalAlignment="Center" VerticalAlignment="Center" />
    </Grid>
</ControlTemplate>

```

Run the project. Notice that when you move the mouse over the button, the color of the `<Ellipse>` animates.



## Next steps

- [Create a style for a control in WPF](#)
- [Styles and templates in WPF](#)
- [Overview of XAML Resources](#)

# How to: Find ControlTemplate-Generated Elements

2 minutes to read • [Edit Online](#)

This example shows how to find elements that are generated by a [ControlTemplate](#).

## Example

The following example shows a style that creates a simple [ControlTemplate](#) for the [Button](#) class:

```
<Style TargetType="{x:Type Button}">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Button}">
                <Grid Margin="5" Name="grid">
                    <Ellipse Stroke="DarkBlue" StrokeThickness="2">
                        <Ellipse.Fill>
                            <RadialGradientBrush Center="0.3,0.2" RadiusX="0.5" RadiusY="0.5">
                                <GradientStop Color="Azure" Offset="0.1" />
                                <GradientStop Color="CornflowerBlue" Offset="1.1" />
                            </RadialGradientBrush>
                        </Ellipse.Fill>
                    </Ellipse>
                    <ContentPresenter Name="content" Margin="10"
                        HorizontalAlignment="Center" VerticalAlignment="Center"/>
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

To find an element within the template after the template has been applied, you can call the [FindName](#) method of the [Template](#). The following example creates a message box that shows the actual width value of the [Grid](#) within the control template:

```
// Finding the grid that is generated by the ControlTemplate of the Button
Grid gridInTemplate = (Grid)myButton1.Template.FindName("grid", myButton1);

// Do something to the ControlTemplate-generated grid
MessageBox.Show("The actual width of the grid in the ControlTemplate: "
    + gridInTemplate.GetValue(Grid.ActualWidthProperty).ToString());
```

```
' Finding the grid that is generated by the ControlTemplate of the Button
Dim gridInTemplate As Grid = CType(myButton1.Template.FindName("grid", myButton1), Grid)

' Do something to the ControlTemplate-generated grid
MessageBox.Show("The actual width of the grid in the ControlTemplate: " &
    gridInTemplate.GetValue(Grid.ActualWidthProperty).ToString())
```

## See also

- [Find DataTemplate-Generated Elements](#)
- [Styling and Templating](#)
- [WPF XAML Namescopes](#)

- [Trees in WPF](#)

# Control Customization

2 minutes to read • [Edit Online](#)

This category covers the various base classes, interfaces and other elements and concepts used in creating a fully functional Windows Presentation Foundation (WPF) control.

## In This Section

[Control Authoring Overview](#)

[Guidelines for Designing Stylistic Controls](#)

[Adorners](#)

[Control Styles and Templates](#)

[UI Automation of a WPF Custom Control](#)

## See also

- [Styling and Templating](#)

# Control Authoring Overview

21 minutes to read • [Edit Online](#)

The extensibility of the Windows Presentation Foundation (WPF) control model greatly reduces the need to create a new control. However, in certain cases you may still need to create a custom control. This topic discusses the features that minimize your need to create a custom control and the different control authoring models in Windows Presentation Foundation (WPF). This topic also demonstrates how to create a new control.

## Alternatives to Writing a New Control

Historically, if you wanted to get a customized experience from an existing control, you were limited to changing the standard properties of the control, such as background color, border width, and font size. If you wished to extend the appearance or behavior of a control beyond these predefined parameters, you would need to create a new control, usually by inheriting from an existing control and overriding the method responsible for drawing the control. Although that is still an option, WPF enables you to customize existing controls by using its rich content model, styles, templates, and triggers. The following list gives examples of how these features can be used to create custom and consistent experiences without having to create a new control.

- **Rich Content.** Many of the standard WPF controls support rich content. For example, the `Content` property of a [Button](#) is of type [Object](#), so theoretically anything can be displayed on a [Button](#). To have a button display an image and text, you can add an image and a [TextBlock](#) to a [StackPanel](#) and assign the [StackPanel](#) to the `Content` property. Because the controls can display WPF visual elements and arbitrary data, there is less need to create a new control or to modify an existing control to support a complex visualization. For more information about the content model for [Button](#) and other content models in WPF, see [WPF Content Model](#).
- **Styles.** A [Style](#) is a collection of values that represent properties for a control. By using styles, you can create a reusable representation of a desired control appearance and behavior without writing a new control. For example, assume that you want all of your [TextBlock](#) controls to have red, Arial font with a font size of 14. You can create a style as a resource and set the appropriate properties accordingly. Then every [TextBlock](#) that you add to your application will have the same appearance.
- **Data Templates.** A [DataTemplate](#) enables you to customize how data is displayed on a control. For example, a [DataTemplate](#) can be used to specify how data is displayed in a [ListBox](#). For an example of this, see [Data Templating Overview](#). In addition to customizing the appearance of data, a [DataTemplate](#) can include UI elements, which gives you a lot of flexibility in custom UIs. For example, by using a [DataTemplate](#), you can create a [ComboBox](#) in which each item contains a check box.
- **Control Templates.** Many controls in WPF use a [ControlTemplate](#) to define the control's structure and appearance, which separates the appearance of a control from the functionality of the control. You can drastically change the appearance of a control by redefining its [ControlTemplate](#). For example, suppose you want a control that looks like a stoplight. This control has a simple user interface and functionality. The control is three circles, only one of which can be lit up at a time. After some reflection, you might realize that a [RadioButton](#) offers the functionality of only one being selected at a time, but the default appearance of the [RadioButton](#) looks nothing like the lights on a stoplight. Because the [RadioButton](#) uses a control template to define its appearance, it is easy to redefine the [ControlTemplate](#) to fit the requirements of the control, and use radio buttons to make your stoplight.

#### **NOTE**

Although a [RadioButton](#) can use a [DataTemplate](#), a [DataTemplate](#) is not sufficient in this example. The [DataTemplate](#) defines the appearance of the content of a control. In the case of a [RadioButton](#), the content is whatever appears to the right of the circle that indicates whether the [RadioButton](#) is selected. In the example of the stoplight, the radio button needs just be a circle that can "light up." Because the appearance requirement for the stoplight is so different than the default appearance of the [RadioButton](#), it is necessary to redefine the [ControlTemplate](#). In general a [DataTemplate](#) is used for defining the content (or data) of a control, and a [ControlTemplate](#) is used for defining how a control is structured.

- **Triggers.** A [Trigger](#) allows you to dynamically change the appearance and behavior of a control without creating a new control. For example, suppose you have multiple [ListBox](#) controls in your application and want the items in each [ListBox](#) to be bold and red when they are selected. Your first instinct might be to create a class that inherits from [ListBox](#) and override the [OnSelectionChanged](#) method to change the appearance of the selected item, but a better approach is to add a trigger to a style of a [ListBoxItem](#) that changes the appearance of the selected item. A trigger enables you to change property values or take actions based on the value of a property. An [EventTrigger](#) enables you to take actions when an event occurs.

For more information about styles, templates, and triggers, see [Styling and Templating](#).

In general, if your control mirrors the functionality of an existing control, but you want the control to look different, you should first consider whether you can use any of the methods discussed in this section to change the existing control's appearance.

## Models for Control Authoring

The rich content model, styles, templates, and triggers minimize the need for you to create a new control. However, if you do need to create a new control, it is important to understand the different control authoring models in WPF. WPF provides three general models for creating a control, each of which provides a different set of features and level of flexibility. The base classes for the three models are [UserControl](#), [Control](#), and [FrameworkElement](#).

### Deriving from [UserControl](#)

The simplest way to create a control in WPF is to derive from [UserControl](#). When you build a control that inherits from [UserControl](#), you add existing components to the [UserControl](#), name the components, and reference event handlers in Extensible Application Markup Language (XAML). You can then reference the named elements and define the event handlers in code. This development model is very similar to the model used for application development in WPF.

If built correctly, a [UserControl](#) can take advantage of the benefits of rich content, styles, and triggers. However, if your control inherits from [UserControl](#), people who use your control will not be able to use a [DataTemplate](#) or [ControlTemplate](#) to customize its appearance. It is necessary to derive from the [Control](#) class or one of its derived classes (other than [UserControl](#)) to create a custom control that supports templates.

#### Benefits of Deriving from [UserControl](#)

Consider deriving from [UserControl](#) if all of the following apply:

- You want to build your control similarly to how you build an application.
- Your control consists only of existing components.
- You don't need to support complex customization.

### Deriving from [Control](#)

Deriving from the [Control](#) class is the model used by most of the existing WPF controls. When you create a control that inherits from the [Control](#) class, you define its appearance by using templates. By doing so, you separate the operational logic from the visual representation. You can also ensure the decoupling of the UI and

logic by using commands and bindings instead of events and avoiding referencing elements in the [ControlTemplate](#) whenever possible. If the UI and logic of your control are properly decoupled, a user of your control can redefine the control's [ControlTemplate](#) to customize its appearance. Although building a custom [Control](#) is not as simple as building a [UserControl](#), a custom [Control](#) provides the most flexibility.

#### **Benefits of Deriving from Control**

Consider deriving from [Control](#) instead of using the [UserControl](#) class if any of the following apply:

- You want the appearance of your control to be customizable via the [ControlTemplate](#).
- You want your control to support different themes.

#### **Deriving from FrameworkElement**

Controls that derive from [UserControl](#) or [Control](#) rely upon composing existing elements. For many scenarios, this is an acceptable solution, because any object that inherits from [FrameworkElement](#) can be in a [ControlTemplate](#).

However, there are times when a control's appearance requires more than the functionality of simple element composition. For these scenarios, basing a component on [FrameworkElement](#) is the right choice.

There are two standard methods for building [FrameworkElement](#)-based components: direct rendering and custom element composition. Direct rendering involves overriding the [OnRender](#) method of [FrameworkElement](#) and providing [DrawingContext](#) operations that explicitly define the component visuals. This is the method used by [Image](#) and [Border](#). Custom element composition involves using objects of type [Visual](#) to compose the appearance of your component. For an example, see [Using DrawingVisual Objects](#). [Track](#) is an example of a control in WPF that uses custom element composition. It is also possible to mix direct rendering and custom element composition in the same control.

#### **Benefits of Deriving from FrameworkElement**

Consider deriving from [FrameworkElement](#) if any of the following apply:

- You want to have precise control over the appearance of your control beyond what is provided by simple element composition.
- You want to define the appearance of your control by defining your own render logic.
- You want to compose existing elements in novel ways that go beyond what is possible with [UserControl](#) and [Control](#).

## **Control Authoring Basics**

As discussed earlier, one of the most powerful features of WPF is the ability to go beyond setting basic properties of a control to change its appearance and behavior, yet still not needing to create a custom control. The styling, data binding, and trigger features are made possible by the WPF property system and the WPF event system. The following sections describe some practices that you should follow, regardless of the model you use to create the custom control, so that users of your custom control can use these features just as they would for a control that is included with WPF.

#### **Use Dependency Properties**

When a property is a dependency property, it is possible to do the following:

- Set the property in a style.
- Bind the property to a data source.
- Use a dynamic resource as the property's value.
- Animate the property.

If you want a property of your control to support any of this functionality, you should implement it as a

dependency property. The following example defines a dependency property named `Value` by doing the following:

- Define a [DependencyProperty](#) identifier named `ValueProperty` as a `public static readonly` field.
- Register the property name with the property system, by calling [DependencyProperty.Register](#), to specify the following:
  - The name of the property.
  - The type of the property.
  - The type that owns the property.
  - The metadata for the property. The metadata contains the property's default value, a [CoerceValueCallback](#) and a [PropertyChangedCallback](#).
- Define a CLR wrapper property named `Value`, which is the same name that is used to register the dependency property, by implementing the property's `get` and `set` accessors. Note that the `get` and `set` accessors only call [GetValue](#) and [SetValue](#) respectively. It is recommended that the accessors of dependency properties not contain additional logic because clients and WPF can bypass the accessors and call [GetValue](#) and [SetValue](#) directly. For example, when a property is bound to a data source, the property's `set` accessor is not called. Instead of adding additional logic to the get and set accessors, use the [ValidateValueCallback](#), [CoerceValueCallback](#), and [PropertyChangedCallback](#) delegates to respond to or check the value when it changes. For more information on these callbacks, see [Dependency Property Callbacks and Validation](#).
- Define a method for the [CoerceValueCallback](#) named `CoerceValue`. `CoerceValue` ensures that `Value` is greater or equal to `MinValue` and less than or equal to `MaxValue`.
- Define a method for the [PropertyChangedCallback](#), named `OnValueChanged`. `OnValueChanged` creates a [RoutedPropertyChangedEventArgs<T>](#) object and prepares to raise the `valueChanged` routed event. Routed events are discussed in the next section.

```
/// <summary>
/// Identifies the Value dependency property.
/// </summary>
public static readonly DependencyProperty ValueProperty =
    DependencyProperty.Register(
        "Value", typeof(decimal), typeof(NumericUpDown),
        new FrameworkPropertyMetadata(MinValue, new PropertyChangedCallback(OnValueChanged),
            new CoerceValueCallback(CoerceValue)));


/// <summary>
/// Gets or sets the value assigned to the control.
/// </summary>
public decimal Value
{
    get { return (decimal)GetValue(ValueProperty); }
    set { SetValue(ValueProperty, value); }
}

private static object CoerceValue(DependencyObject element, object value)
{
    decimal newValue = (decimal)value;
    NumericUpDown control = (NumericUpDown)element;

    newValue = Math.Max(MinValue, Math.Min.MaxValue, newValue));

    return newValue;
}

private static void OnValueChanged(DependencyObject obj, DependencyPropertyChangedEventArgs args)
{
    NumericUpDown control = (NumericUpDown)obj;

    RoutedPropertyChangedEventArgs<decimal> e = new RoutedPropertyChangedEventArgs<decimal>(
        (decimal)args.OldValue, (decimal)args.NewValue, ValueChangedEvent);
    control.OnValueChanged(e);
}
```

```

''' <summary>
''' Identifies the Value dependency property.
''' </summary>
Public Shared ReadOnly ValueProperty As DependencyProperty = DependencyProperty.Register("Value",
GetType(Decimal), GetType(NumericUpDown), New FrameworkPropertyMetadata(MinValue, New
PropertyChangedCallback(AddressOf OnValueChanged), New CoerceValueCallback(AddressOf CoerceValue)))

''' <summary>
''' Gets or sets the value assigned to the control.
''' </summary>
Public Property Value() As Decimal
    Get
        Return CDecGetValue(ValueProperty)
    End Get
    Set(ByVal value As Decimal)
        SetValue(ValueProperty, value)
    End Set
End Property

Private Shared Overloads Function CoerceValue(ByVal element As DependencyObject, ByVal value As Object) As
Object
    Dim newValue As Decimal = CDec(value)
    Dim control As NumericUpDown = CType(element, NumericUpDown)

    newValue = Math.Max(MinValue, Math.Min.MaxValue, newValue)

    Return newValue
End Function

Private Shared Sub OnValueChanged(ByVal obj As DependencyObject, ByVal args As
DependencyPropertyChangedEventArgs)
    Dim control As NumericUpDown = CType(obj, NumericUpDown)

    Dim e As New RoutedPropertyChangedEventArgs(Of Decimal)(CDec(args.OldValue), CDec(args.NewValue),
ValueChangedEventArgs)
    control.OnValueChanged(e)
End Sub

```

For more information, see [Custom Dependency Properties](#).

## Use Routed Events

Just as dependency properties extend the notion of CLR properties with additional functionality, routed events extend the notion of standard CLR events. When you create a new WPF control, it is also good practice to implement your event as a routed event because a routed event supports the following behavior:

- Events can be handled on a parent of multiple controls. If an event is a bubbling event, a single parent in the element tree can subscribe to the event. Then application authors can use one handler to respond to the event of multiple controls. For example, if your control is a part of each item in a [ListBox](#) (because it is included in a [DataTemplate](#)), the application developer can define the event handler for your control's event on the [ListBox](#). Whenever the event occurs on any of the controls, the event handler is called.
- Routed events can be used in an [EventSetter](#), which enables application developers to specify the handler of an event within a style.
- Routed events can be used in an [EventTrigger](#), which is useful for animating properties by using XAML. For more information, see [Animation Overview](#).

The following example defines a routed event by doing the following:

- Define a [RoutedEvent](#) identifier named `ValueChangedEvent` as a `public static readonly` field.
- Register the routed event by calling the [EventManager.RegisterRoutedEventArgs](#) method. The example specifies

the following information when it calls `RegisterRoutedEvent`:

- The name of the event is `ValueChanged`.
  - The routing strategy is `Bubble`, which means that an event handler on the source (the object that raises the event) is called first, and then event handlers on the source's parent elements are called in succession, starting with the event handler on the closest parent element.
  - The type of the event handler is `RoutedPropertyChangedEventHandler<T>`, constructed with a `Decimal` type.
  - The owning type of the event is `NumericUpDown`.
- Declare a public event named `ValueChanged` and includes event-accessor declarations. The example calls `AddHandler` in the `add` accessor declaration and `RemoveHandler` in the `remove` accessor declaration to use the WPF event services.
  - Create a protected, virtual method named `OnValueChanged` that raises the `ValueChanged` event.

```
/// <summary>
/// Identifies the ValueChanged routed event.
/// </summary>
public static readonly RoutedEvent ValueChangedEvent = EventManager.RegisterRoutedEvent(
    "ValueChanged", RoutingStrategy.Bubble,
    typeof(RoutedPropertyChangedEventHandler<decimal>), typeof(NumericUpDown));

/// <summary>
/// Occurs when the Value property changes.
/// </summary>
public event RoutedPropertyChangedEventHandler<decimal> ValueChanged
{
    add { AddHandler(ValueChangedEvent, value); }
    remove { RemoveHandler(ValueChangedEvent, value); }
}

/// <summary>
/// Raises the ValueChanged event.
/// </summary>
/// <param name="args">Arguments associated with the ValueChanged event.</param>
protected virtual void OnValueChanged(RoutedEventArgs<decimal> args)
{
    RaiseEvent(args);
}
```

```

''' <summary>
''' Identifies the ValueChanged routed event.
''' </summary>
Public Shared ReadOnly ValueChangedEvent As RoutedEvent = EventManager.RegisterRoutedEvent("ValueChanged",
RoutingStrategy.Bubble, GetType(RoutedPropertyChangedEventHandler(Of Decimal)), GetType(NumericUpDown))

''' <summary>
''' Occurs when the Value property changes.
''' </summary>
Public Custom Event ValueChanged As RoutedPropertyChangedEventHandler(Of Decimal)
    AddHandler(ByVal value As RoutedPropertyChangedEventHandler(Of Decimal))
        MyBase.AddHandler(ValueChangedEvent, value)
    End AddHandler
    RemoveHandler(ByVal value As RoutedPropertyChangedEventHandler(Of Decimal))
        MyBase.RemoveHandler(ValueChangedEvent, value)
    End RemoveHandler
    RaiseEvent(ByVal sender As System.Object, ByVal e As RoutedPropertyChangedEventArgs(Of Decimal))
    End RaiseEvent
End Event

''' <summary>
''' Raises the ValueChanged event.
''' </summary>
''' <param name="args">Arguments associated with the ValueChanged event.</param>
Protected Overridable Sub OnValueChanged(ByVal args As RoutedPropertyChangedEventArgs(Of Decimal))
    MyBase.RaiseEvent(args)
End Sub

```

For more information, see [Routed Events Overview](#) and [Create a Custom Routed Event](#).

## Use Binding

To decouple the UI of your control from its logic, consider using data binding. This is particularly important if you define the appearance of your control by using a [ControlTemplate](#). When you use data binding, you might be able to eliminate the need to reference specific parts of the UI from the code. It's a good idea to avoid referencing elements that are in the [ControlTemplate](#) because when the code references elements that are in the [ControlTemplate](#) and the [ControlTemplate](#) is changed, the referenced element needs to be included in the new [ControlTemplate](#).

The following example updates the [TextBlock](#) of the [NumericUpDown](#) control, assigning a name to it and referencing the textbox by name in code.

```

<Border BorderThickness="1" BorderBrush="Gray" Margin="2"
        Grid.RowSpan="2" VerticalAlignment="Center" HorizontalAlignment="Stretch">
    <TextBlock Name="valueText" Width="60" TextAlignment="Right" Padding="5"/>
</Border>

```

```

private void UpdateTextBlock()
{
    valueText.Text = Value.ToString();
}

```

```

Private Sub UpdateTextBlock()
    valueText.Text = Value.ToString()
End Sub

```

The following example uses binding to accomplish the same thing.

```

<Border BorderThickness="1" BorderBrush="Gray" Margin="2"
        Grid.RowSpan="2" VerticalAlignment="Center" HorizontalAlignment="Stretch">

    <!--Bind the TextBlock to the Value property-->
    <TextBlock
        Width="60" TextAlignment="Right" Padding="5"
        Text="{Binding RelativeSource={RelativeSource FindAncestor,
            AncestorType={x:Type local:NumericUpDown}},
        Path=Value}"/>

</Border>

```

For more information about data binding, see [Data Binding Overview](#).

## Design for Designers

To receive support for custom WPF controls in the WPF Designer for Visual Studio (for example, property editing with the Properties window), follow these guidelines. For more information on developing for the WPF Designer, see [Design XAML in Visual Studio](#).

### Dependency Properties

Be sure to implement CLR `get` and `set` accessors as described earlier, in "Use Dependency Properties."

Designers may use the wrapper to detect the presence of a dependency property, but they, like WPF and clients of the control, are not required to call the accessors when getting or setting the property.

### Attached Properties

You should implement attached properties on custom controls using the following guidelines:

- Have a `public static readonly DependencyProperty` of the form `PropertyName Property` that was created using the `RegisterAttached` method. The property name that is passed to `RegisterAttached` must match `PropertyName`.
- Implement a pair of `public static` CLR methods named `SetPropertyName` and `GetPropertyName`. Both methods should accept a class derived from `DependencyProperty` as their first argument. The `SetPropertyName` method also accepts an argument whose type matches the registered data type for the property. The `GetPropertyName` method should return a value of the same type. If the `SetPropertyName` method is missing, the property is marked read-only.
- `SetPropertyName` and `GetPropertyName` must route directly to the `GetValue` and `SetValue` methods on the target dependency object, respectively. Designers may access the attached property by calling through the method wrapper or making a direct call to the target dependency object.

For more information on attached properties, see [Attached Properties Overview](#).

## Define and Use Shared Resources

You can include your control in the same assembly as your application, or you can package your control in a separate assembly that can be used in multiple applications. For the most part, the information discussed in this topic applies regardless of the method you use. There is one difference worth noting, however. When you put a control in the same assembly as an application, you are free to add global resources to the App.xaml file. But an assembly that contains only controls does not have an `Application` object associated with it, so an App.xaml file is not available.

When an application looks for a resource, it looks at three levels in the following order:

1. The element level.

The system starts with the element that references the resource and then searches resources of the logical parent and so forth until the root element is reached.

## 2. The application level.

Resources defined by the [Application](#) object.

## 3. The theme level.

Theme-level dictionaries are stored in a subfolder named Themes. The files in the Themes folder correspond to themes. For example, you might have Aero.NormalColor.xaml, Luna.NormalColor.xaml, Royale.NormalColor.xaml, and so on. You can also have a file named generic.xaml. When the system looks for a resource at the themes level, it first looks for it in the theme-specific file and then looks for it in generic.xaml.

When your control is in an assembly that is separate from the application, you must put your global resources at the element level or at the theme level. Both methods have their advantages.

### Defining Resources at the Element Level

You can define shared resources at the element level by creating a custom resource dictionary and merging it with your control's resource dictionary. When you use this method, you can name your resource file anything you want, and it can be in the same folder as your controls. Resources at the element level can also use simple strings as keys. The following example creates a [LinearGradientBrush](#) resource file named Dictionary1.xaml.

```
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <LinearGradientBrush
        x:Key="myBrush"
        StartPoint="0,0" EndPoint="1,1">
        <GradientStop Color="Red" Offset="0.25" />
        <GradientStop Color="Blue" Offset="0.75" />
    </LinearGradientBrush>
</ResourceDictionary>
```

Once you have defined your dictionary, you need to merge it with your control's resource dictionary. You can do this by using XAML or code.

The following example merges a resource dictionary by using XAML.

```
<UserControl.Resources>
    <ResourceDictionary>
        <ResourceDictionary.MergedDictionaries>
            <ResourceDictionary Source="Dictionary1.xaml"/>
        </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
</UserControl.Resources>
```

The disadvantage to this approach is that a [ResourceDictionary](#) object is created each time you reference it. For example, if you have 10 custom controls in your library and merge the shared resource dictionaries for each control by using XAML, you create 10 identical [ResourceDictionary](#) objects. You can avoid this by creating a static class that merges the resources in code and returns the resulting [ResourceDictionary](#).

The following example creates a class that returns a shared [ResourceDictionary](#).

```

internal static class SharedDictionaryManager
{
    internal static ResourceDictionary SharedDictionary
    {
        get
        {
            if (_sharedDictionary == null)
            {
                System.Uri resourceLocater =
                    new System.Uri("/ElementResourcesCustomControlLibrary;component/Dictionary1.xaml",
                        System.UriKind.Relative);

                _sharedDictionary =
                    (ResourceDictionary)Application.LoadComponent(resourceLocater);
            }

            return _sharedDictionary;
        }
    }

    private static ResourceDictionary _sharedDictionary;
}

```

The following example merges the shared resource with the resources of a custom control in the control's constructor before it calls `InitializeComponent`. Because the `SharedDictionaryManager.SharedDictionary` is a static property, the `ResourceDictionary` is created only once. Because the resource dictionary was merged before `InitializeComponent` was called, the resources are available to the control in its XAML file.

```

public NumericUpDown()
{
    this.Resources.MergedDictionaries.Add(SharedDictionaryManager.SharedDictionary);
    InitializeComponent();
}

```

#### Defining Resources at the Theme Level

WPF enables you to create resources for different Windows themes. As a control author, you can define a resource for a specific theme to change your control's appearance depending on what theme is in use. For example, the appearance of a `Button` in the Windows Classic theme (the default theme for Windows 2000) differs from a `Button` in the Windows Luna theme (the default theme for Windows XP) because the `Button` uses a different `ControlTemplate` for each theme.

Resources that are specific to a theme are kept in a resource dictionary with a specific file name. These files must be in a folder named `Themes` that is a subfolder of the folder that contains the control. The following table lists the resource dictionary files and the theme that is associated with each file:

RESOURCE DICTIONARY FILE NAME	WINDOWS THEME
<code>Classic.xaml</code>	Classic Windows 9x/2000 look on Windows XP
<code>Luna.NormalColor.xaml</code>	Default blue theme on Windows XP
<code>Luna.Homestead.xaml</code>	Olive theme on Windows XP
<code>Luna.Metallic.xaml</code>	Silver theme on Windows XP
<code>Royale.NormalColor.xaml</code>	Default theme on Windows XP Media Center Edition

RESOURCE DICTIONARY FILE NAME	WINDOWS THEME
Aero.NormalColor.xaml	Default theme on Windows Vista

You do not need to define a resource for every theme. If a resource is not defined for a specific theme, then the control checks `classic.xaml` for the resource. If the resource is not defined in the file that corresponds to the current theme or in `Classic.xaml`, the control uses the generic resource, which is in a resource dictionary file named `generic.xaml`. The `generic.xaml` file is located in the same folder as the theme-specific resource dictionary files. Although `generic.xaml` does not correspond to a specific Windows theme, it is still a theme-level dictionary.

The [C#](#) or [Visual Basic](#) NumericUpDown custom control with theme and UI automation support sample contains two resource dictionaries for the `NumericUpDown` control: one is in `generic.xaml`, and the other is in `Luna.NormalColor.xaml`.

When you put a [ControlTemplate](#) in any of the theme-specific resource dictionary files, you must create a static constructor for your control and call the [OverrideMetadata\(Type, PropertyMetadata\)](#) method on the [DefaultStyleKey](#), as shown in the following example.

```
static NumericUpDown()
{
    DefaultStyleKeyProperty.OverrideMetadata(typeof(NumericUpDown),
        new FrameworkPropertyMetadata(typeof(NumericUpDown)));
}
```

```
Shared Sub New()
    DefaultStyleKeyProperty.OverrideMetadata(GetType(NumericUpDown), New
FrameworkPropertyMetadata(GetType(NumericUpDown)))
End Sub
```

#### Defining and Referencing Keys for Theme Resources

When you define a resource at the element level, you can assign a string as its key and access the resource via the string. When you define a resource at the theme level, you must use a [ComponentResourceKey](#) as the key. The following example defines a resource in `generic.xaml`.

```
<LinearGradientBrush
    x:Key="{ComponentResourceKey TypeInTargetAssembly={x:Type local:Painter},
        ResourceId=MyEllipseBrush}"
    StartPoint="0,0" EndPoint="1,0">
    <GradientStop Color="Blue" Offset="0" />
    <GradientStop Color="Red" Offset="0.5" />
    <GradientStop Color="Green" Offset="1"/>
</LinearGradientBrush>
```

The following example references the resource by specifying the [ComponentResourceKey](#) as the key.

```

<RepeatButton
    Grid.Column="1" Grid.Row="0"
    Background="{StaticResource {ComponentResourceKey
        TypeInTargetAssembly={x:Type local:NumericUpDown},
        ResourceId=ButtonBrush}}">
    Up
</RepeatButton>
<RepeatButton
    Grid.Column="1" Grid.Row="1"
    Background="{StaticResource {ComponentResourceKey
        TypeInTargetAssembly={x:Type local:NumericUpDown},
        ResourceId=ButtonBrush}}">
    Down
</RepeatButton>

```

#### **Specifying the Location of Theme Resources**

To find the resources for a control, the hosting application needs to know that the assembly contains control-specific resources. You can accomplish that by adding the [ThemeInfoAttribute](#) to the assembly that contains the control. The [ThemeInfoAttribute](#) has a [GenericDictionaryLocation](#) property that specifies the location of generic resources, and a [ThemeDictionaryLocation](#) property that specifies the location of the theme-specific resources.

The following example sets the [GenericDictionaryLocation](#) and [ThemeDictionaryLocation](#) properties to [SourceAssembly](#), to specify that the generic and theme-specific resources are in the same assembly as the control.

```
[assembly: ThemeInfo(ResourceDictionaryLocation.SourceAssembly,
    ResourceDictionaryLocation.SourceAssembly)]
```

```
<Assembly: ThemeInfo(ResourceDictionaryLocation.SourceAssembly, ResourceDictionaryLocation.SourceAssembly)>
```

## See also

- [Design XAML in Visual Studio](#)
- [Pack URIs in WPF](#)
- [Control Customization](#)

# Creating a Control That Has a Customizable Appearance

24 minutes to read • [Edit Online](#)

Windows Presentation Foundation (WPF) gives you the ability to create a control whose appearance can be customized. For example, you can change the appearance of a [CheckBox](#) beyond what setting properties will do by creating a new [ControlTemplate](#). The following illustration shows a [CheckBox](#) that uses a default [ControlTemplate](#) and a [CheckBox](#) that uses a custom [ControlTemplate](#).

**CheckBox1** A CheckBox that uses the default control template

**CheckBox2**



A CheckBox that uses a custom control template

If you follow the parts and states model when you create a control, your control's appearance will be customizable. Designer tools such as Blend for Visual Studio support the parts and states model, so when you follow this model your control will be customizable in those types of applications. This topic discusses the parts and states model and how to follow it when you create your own control. This topic uses an example of a custom control, [NumericUpDown](#), to illustrate the philosophy of this model. The [NumericUpDown](#) control displays a numeric value, which a user can increase or decrease by clicking on the control's buttons. The following illustration shows the [NumericUpDown](#) control that is discussed in this topic.



A custom NumericUpDown control

This topic contains the following sections:

- [Prerequisites](#)
- [Parts and States Model](#)
- [Defining the Visual Structure and Visual Behavior of a Control in a ControlTemplate](#)
- [Using Parts of the ControlTemplate in Code](#)
- [Providing the Control Contract](#)
- [Complete Example](#)

## Prerequisites

This topic assumes that you know how to create a new [ControlTemplate](#) for an existing control, are familiar with what the elements on a control contract are, and understand the concepts discussed in [Create a template for a control](#).

### NOTE

To create a control that can have its appearance customized, you must create a control that inherits from the [Control](#) class or one of its subclasses other than [UserControl](#). A control that inherits from [UserControl](#) is a control that can be quickly created, but it does not use a [ControlTemplate](#) and you cannot customize its appearance.

## Parts and States Model

The parts and states model specifies how to define the visual structure and visual behavior of a control. To follow the parts and states model, you should do the following:

- Define the visual structure and visual behavior in the [ControlTemplate](#) of a control.
- Follow certain best practices when your control's logic interacts with parts of the control template.
- Provide a control contract to specify what should be included in the [ControlTemplate](#).

When you define the visual structure and visual behavior in the [ControlTemplate](#) of a control, application authors can change the visual structure and visual behavior of your control by creating a new [ControlTemplate](#) instead of writing code. You must provide a control contract that tells application authors which [FrameworkElement](#) objects and states should be defined in the [ControlTemplate](#). You should follow some best practices when you interact with the parts in the [ControlTemplate](#) so that your control properly handles an incomplete [ControlTemplate](#). If you follow these three principles, application authors will be able to create a [ControlTemplate](#) for your control just as easily as they can for the controls that ship with WPF. The following section explains each of these recommendations in detail.

## Defining the Visual Structure and Visual Behavior of a Control in a ControlTemplate

When you create your custom control by using the parts and states model, you define the control's visual structure and visual behavior in its [ControlTemplate](#) instead of in its logic. The visual structure of a control is the composite of [FrameworkElement](#) objects that make up the control. The visual behavior is the way the control appears when it is in a certain state. For more information about creating a [ControlTemplate](#) that specifies the visual structure and visual behavior of a control, see [Create a template for a control](#).

In the example of the `NumericUpDown` control, the visual structure includes two [RepeatButton](#) controls and a [TextBlock](#). If you add these controls in the code of the `NumericUpDown` control--in its constructor, for example--the positions of those controls would be unalterable. Instead of defining the control's visual structure and visual behavior in its code, you should define it in the [ControlTemplate](#). Then an application developer can customize the position of the buttons and [TextBlock](#) and specify what behavior occurs when `Value` is negative because the [ControlTemplate](#) can be replaced.

The following example shows the visual structure of the `NumericUpDown` control, which includes a [RepeatButton](#) to increase `Value`, a [RepeatButton](#) to decrease `Value`, and a [TextBlock](#) to display `Value`.

```

<ControlTemplate TargetType="src:NumericUpDown">
    <Grid Margin="3"
        Background="{TemplateBinding Background}">
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition/>
                <RowDefinition/>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>

            <Border BorderThickness="1" BorderBrush="Gray"
                Margin="7,2,2,2" Grid.RowSpan="2"
                Background="#E0FFFFFF"
                VerticalAlignment="Center"
                HorizontalAlignment="Stretch">

                <!--Bind the TextBlock to the Value property-->
                <TextBlock Name="TextBlock"
                    Width="60" TextAlignment="Right" Padding="5"
                    Text="{Binding RelativeSource={RelativeSource FindAncestor,
                        AncestorType={x:Type src:NumericUpDown}},
                        Path=Value}"/>
            </Border>

            <RepeatButton Content="Up" Margin="2,5,5,0"
                Name="UpButton"
                Grid.Column="1" Grid.Row="0"/>
            <RepeatButton Content="Down" Margin="2,0,5,5"
                Name="DownButton"
                Grid.Column="1" Grid.Row="1"/>

            <Rectangle Name="FocusVisual" Grid.ColumnSpan="2" Grid.RowSpan="2"
                Stroke="Black" StrokeThickness="1"
                Visibility="Collapsed"/>
        </Grid>

    </Grid>
</ControlTemplate>

```

A visual behavior of the `NumericUpDown` control is that the value is in a red font if it is negative. If you change the `Foreground` of the `TextBlock` in code when the `value` is negative, the `NumericUpDown` will always show a red negative value. You specify the visual behavior of the control in the `ControlTemplate` by adding `VisualState` objects to the `ControlTemplate`. The following example shows the `VisualState` objects for the `Positive` and `Negative` states. `Positive` and `Negative` are mutually exclusive (the control is always in exactly one of the two), so the example puts the `VisualState` objects into a single `VisualStyleGroup`. When the control goes into the `Negative` state, the `Foreground` of the `TextBlock` turns red. When the control is in the `Positive` state, the `Foreground` returns to its original value. Defining `VisualStyle` objects in a `ControlTemplate` is further discussed in [Create a template for a control](#).

#### NOTE

Be sure to set the `VisualStyleGroups` attached property on the root `FrameworkElement` of the `ControlTemplate`.

```

<ControlTemplate TargetType="local:NumericUpDown">
    <Grid Margin="3"
        Background="{TemplateBinding Background}">

        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup Name="ValueStates">

                <!--Make the Value property red when it is negative.-->
                <VisualState Name="Negative">
                    <Storyboard>
                        <ColorAnimation To="Red"
                            Storyboard.TargetName="TextBlock"
                            Storyboard.TargetProperty="(Foreground).(Color)"/>
                    </Storyboard>
                </VisualState>

                <!--Return the TextBlock's Foreground to its
                    original color.-->
                <VisualState Name="Positive"/>
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>
    </Grid>
</ControlTemplate>

```

## Using Parts of the ControlTemplate in Code

A [ControlTemplate](#) author might omit [FrameworkElement](#) or [VisualState](#) objects, either purposefully or by mistake, but your control's logic might need those parts to function properly. The parts and states model specifies that your control should be resilient to a [ControlTemplate](#) that is missing [FrameworkElement](#) or [VisualState](#) objects. Your control should not throw an exception or report an error if a [FrameworkElement](#), [VisualState](#), or [VisualStateGroup](#) is missing from the [ControlTemplate](#). This section describes the recommended practices for interacting with [FrameworkElement](#) objects and managing states.

### Anticipate Missing FrameworkElement Objects

When you define [FrameworkElement](#) objects in the [ControlTemplate](#), your control's logic might need to interact with some of them. For example, the `NumericUpDown` control subscribes to the buttons' [Click](#) event to increase or decrease `Value` and sets the `Text` property of the `TextBlock` to `Value`. If a custom [ControlTemplate](#) omits the `TextBlock` or buttons, it is acceptable that the control loses some of its functionality, but you should be sure that your control does not cause an error. For example, if a [ControlTemplate](#) does not contain the buttons to change `Value`, the `NumericUpDown` loses that functionality, but an application that uses the [ControlTemplate](#) will continue to run.

The following practices will ensure that your control responds properly to missing [FrameworkElement](#) objects:

1. Set the `x:Name` attribute for each [FrameworkElement](#) that you need to reference in code.
2. Define private properties for each [FrameworkElement](#) that you need to interact with.
3. Subscribe to and unsubscribe from any events that your control handles in the [FrameworkElement](#) property's set accessor.
4. Set the [FrameworkElement](#) properties that you defined in step 2 in the [OnApplyTemplate](#) method. This is the earliest that the [FrameworkElement](#) in the [ControlTemplate](#) is available to the control. Use the `x:Name` of the [FrameworkElement](#) to get it from the [ControlTemplate](#).
5. Check that the [FrameworkElement](#) is not `null` before accessing its members. If it is `null`, do not report an error.

The following examples show how the `NumericUpDown` control interacts with `FrameworkElement` objects in accordance with the recommendations in the preceding list.

In the example that defines the visual structure of the `NumericUpDown` control in the `ControlTemplate`, the `RepeatButton` that increases `Value` has its `x:Name` attribute set to `upButton`. The following example declares a property called `UpButtonElement` that represents the `RepeatButton` that is declared in the `ControlTemplate`. The `set` accessor first unsubscribes to the button's `Click` event if `UpDownElement` is not `null`, then it sets the property, and then it subscribes to the `Click` event. There is also a property defined, but not shown here, for the other `RepeatButton`, called `DownButtonElement`.

```
private RepeatButton upButtonElement;

private RepeatButton UpButtonElement
{
    get
    {
        return upButtonElement;
    }

    set
    {
        if (upButtonElement != null)
        {
            upButtonElement.Click -=
                new RoutedEventHandler(upButtonElement_Click);
        }
        upButtonElement = value;

        if (upButtonElement != null)
        {
            upButtonElement.Click +=
                new RoutedEventHandler(upButtonElement_Click);
        }
    }
}
```

```
Private m_upButtonElement As RepeatButton

Private Property UpButtonElement() As RepeatButton
    Get
        Return m_upButtonElement
    End Get

    Set(ByVal value As RepeatButton)
        If m_upButtonElement IsNot Nothing Then
            RemoveHandler m_upButtonElement.Click, AddressOf upButtonElement_Click
        End If
        m_upButtonElement = value

        If m_upButtonElement IsNot Nothing Then
            AddHandler m_upButtonElement.Click, AddressOf upButtonElement_Click
        End If
    End Set
End Property
```

The following example shows the `OnApplyTemplate` for the `NumericUpDown` control. The example uses the `GetTemplateChild` method to get the `FrameworkElement` objects from the `ControlTemplate`. Notice that the example guards against cases where `GetTemplateChild` finds a `FrameworkElement` with the specified name that is not of the expected type. It is also a best practice to ignore elements that have the specified `x:Name` but are of the wrong type.

```

public override void OnApplyTemplate()
{
    UpButtonElement = GetTemplateChild("UpButton") as RepeatButton;
    DownButtonElement = GetTemplateChild("DownButton") as RepeatButton;
    //TextElement = GetTemplateChild("TextBlock") as TextBlock;

    UpdateStates(false);
}

```

```

Public Overloads Overrides Sub OnApplyTemplate()

    UpButtonElement = TryCast(GetTemplateChild("UpButton"), RepeatButton)
    DownButtonElement = TryCast(GetTemplateChild("DownButton"), RepeatButton)

    UpdateStates(False)
End Sub

```

By following the practices that are shown in the previous examples, you ensure that your control will continue to run when the [ControlTemplate](#) is missing a [FrameworkElement](#).

### Use the [VisualStateManager](#) to Manage States

The [VisualStateManager](#) keeps track of the states of a control and performs the logic necessary to transition between states. When you add [VisualState](#) objects to the [ControlTemplate](#), you add them to a [VisualStateGroup](#) and add the [VisualStateGroup](#) to the [VisualStateGroups](#) attached property so that the [VisualStateManager](#) has access to them.

The following example repeats the previous example that shows the [VisualState](#) objects that correspond to the [Positive](#) and [Negative](#) states of the control. The [Storyboard](#) in the [Negative](#) [VisualState](#) turns the [Foreground](#) of the [TextBlock](#) red. When the [NumericUpDown](#) control is in the [Negative](#) state, the storyboard in the [Negative](#) state begins. Then the [Storyboard](#) in the [Negative](#) state stops when the control returns to the [Positive](#) state. The [Positive](#) [VisualState](#) does not need to contain a [Storyboard](#) because when the [Storyboard](#) for the [Negative](#) stops, the [Foreground](#) returns to its original color.

```

<ControlTemplate TargetType="local:NumericUpDown">
    <Grid Margin="3"
        Background="{TemplateBinding Background}">

        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup Name="ValueStates">

                <!--Make the Value property red when it is negative.-->
                <VisualState Name="Negative">
                    <Storyboard>
                        <ColorAnimation To="Red"
                            Storyboard.TargetName="TextBlock"
                            Storyboard.TargetProperty="(Foreground).(Color)"/>
                    </Storyboard>
                </VisualState>

                <!--Return the TextBlock's Foreground to its
                    original color.-->
                <VisualState Name="Positive"/>
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>
    </Grid>
</ControlTemplate>

```

Note that the [TextBlock](#) is given a name, but the [TextBlock](#) is not in the control contract for [NumericUpDown](#) because

the control's logic never references the [TextBlock](#). Elements that are referenced in the [ControlTemplate](#) have names, but do not need to be part of the control contract because a new [ControlTemplate](#) for the control might not need to reference that element. For example, someone who creates a new [ControlTemplate](#) for [NumericUpDown](#) might decide to not indicate that [Value](#) is negative by changing the [Foreground](#). In that case, neither the code nor the [ControlTemplate](#) references the [TextBlock](#) by name.

The control's logic is responsible for changing the control's state. The following example shows that the [NumericUpDown](#) control calls the [GoToState](#) method to go into the [Positive](#) state when [Value](#) is 0 or greater, and the [Negative](#) state when [Value](#) is less than 0.

```
if (Value >= 0)
{
    VisualStateManager.GoToState(this, "Positive", useTransitions);
}
else
{
    VisualStateManager.GoToState(this, "Negative", useTransitions);
}
```

```
If Value >= 0 Then
    VisualStateManager.GoToState(Me, "Positive", useTransitions)
Else
    VisualStateManager.GoToState(Me, "Negative", useTransitions)
End If
```

The [GoToState](#) method performs the logic necessary to start and stop the storyboards appropriately. When a control calls [GoToState](#) to change its state, the [VisualStateManager](#) does the following:

- If the [VisualState](#) that the control is going to has a [Storyboard](#), the storyboard begins. Then, if the [VisualState](#) that the control is coming from has a [Storyboard](#), the storyboard ends.
- If the control is already in the state that is specified, [GoToState](#) takes no action and returns [true](#).
- If state that is specified doesn't exist in the [ControlTemplate](#) of [control](#), [GoToState](#) takes no action and returns [false](#).

#### Best Practices for Working with the VisualStateManager

It is recommended that you do the following to maintain your control's states:

- Use properties to track its state.
- Create a helper method to transition between states.

The [NumericUpDown](#) control uses its [Value](#) property to track whether it is in the [Positive](#) or [Negative](#) state. The [NumericUpDown](#) control also defines the [Focused](#) and [UnFocused](#) states, which tracks the [IsFocused](#) property. If you use states that do not naturally correspond to a property of the control, you can define a private property to track the state.

A single method that updates all the states centralizes calls to the [VisualStateManager](#) and keeps your code manageable. The following example shows the [NumericUpDown](#) control's helper method, [UpdateStates](#). When [Value](#) is greater than or equal to 0, the [Control](#) is in the [Positive](#) state. When [Value](#) is less than 0, the control is in the [Negative](#) state. When [IsFocused](#) is [true](#), the control is in the [Focused](#) state; otherwise, it is in the [Unfocused](#) state. The control can call [UpdateStates](#) whenever it needs to change its state, regardless of what state changes.

```

private void UpdateStates(bool useTransitions)
{
    if (Value >= 0)
    {
        VisualStateManager.GoToState(this, "Positive", useTransitions);
    }
    else
    {
        VisualStateManager.GoToState(this, "Negative", useTransitions);
    }

    if (IsFocused)
    {
        VisualStateManager.GoToState(this, "Focused", useTransitions);
    }
    else
    {
        VisualStateManager.GoToState(this, "Unfocused", useTransitions);
    }
}

```

```

Private Sub UpdateStates(ByVal useTransitions As Boolean)

    If Value >= 0 Then
        VisualStateManager.GoToState(Me, "Positive", useTransitions)
    Else
        VisualStateManager.GoToState(Me, "Negative", useTransitions)
    End If

    If IsFocused Then
        VisualStateManager.GoToState(Me, "Focused", useTransitions)
    Else
        VisualStateManager.GoToState(Me, "Unfocused", useTransitions)

    End If
End Sub

```

If you pass a state name to [GoToState](#) when the control is already in that state, [GoToState](#) does nothing, so you don't need to check for the control's current state. For example, if `value` changes from one negative number to another negative number, the storyboard for the `Negative` state is not interrupted and the user will not see a change in the control.

The [VisualStateManager](#) uses [VisualStateGroup](#) objects to determine which state to exit when you call [GoToState](#). The control is always in one state for each [VisualStateGroup](#) that is defined in its [ControlTemplate](#) and only leaves a state when it goes into another state from the same [VisualStateGroup](#). For example, the [ControlTemplate](#) of the `NumericUpDown` control defines the `Positive` and `Negative` [VisualState](#) objects in one [VisualStateGroup](#) and the `Focused` and `Unfocused` [VisualState](#) objects in another. (You can see the `Focused` and `Unfocused` [VisualState](#) defined in the [Complete Example](#) section in this topic.) When the control goes from the `Positive` state to the `Negative` state, or vice versa, the control remains in either the `Focused` or `Unfocused` state.

There are three typical places where the state of a control might change:

- When the [ControlTemplate](#) is applied to the [Control](#).
- When a property changes.
- When an event occurs.

The following examples demonstrate updating the state of the `NumericUpDown` control in these cases.

You should update the state of the control in the [OnApplyTemplate](#) method so that the control appears in the

correct state when the `ControlTemplate` is applied. The following example calls `UpdateStates` in `OnApplyTemplate` to ensure that the control is in the appropriate states. For example, suppose that you create a `NumericUpDown` control, and then set its `Foreground` to green and `Value` to -5. If you do not call `UpdateStates` when the `ControlTemplate` is applied to the `NumericUpDown` control, the control is not in the `Negative` state and the value is green instead of red. You must call `UpdateStates` to put the control in the `Negative` state.

```
public override void OnApplyTemplate()
{
    UpButtonElement = GetTemplateChild("UpButton") as RepeatButton;
    DownButtonElement = GetTemplateChild("DownButton") as RepeatButton;
    //TextElement = GetTemplateChild("TextBlock") as TextBlock;

    UpdateStates(false);
}
```

```
Public Overloads Overrides Sub OnApplyTemplate()

    UpButtonElement = TryCast(GetTemplateChild("UpButton"), RepeatButton)
    DownButtonElement = TryCast(GetTemplateChild("DownButton"), RepeatButton)

    UpdateStates(False)
End Sub
```

You often need to update the states of a control when a property changes. The following example shows the entire `ValueChangedCallback` method. Because `ValueChangedCallback` is called when `Value` changes, the method calls `UpdateStates` in case `Value` changed from positive to negative or vice versa. It is acceptable to call `UpdateStates` when `Value` changes but remains positive or negative because in that case, the control will not change states.

```
private static void ValueChangedCallback(DependencyObject obj,
    DependencyPropertyChangedEventArgs args)
{
    NumericUpDown ctl = (NumericUpDown)obj;
    int newValue = (int)args.NewValue;

    // Call UpdateStates because the Value might have caused the
    // control to change ValueStates.
    ctl.UpdateStates(true);

    // Call OnValueChanged to raise the ValueChanged event.
    ctl.OnValueChanged(
        new ValueChangedEventArgs(NumericUpDown.ValueChangedEvent,
            newValue));
}
```

```
Private Shared Sub ValueChangedCallback(ByVal obj As DependencyObject,
    ByVal args As DependencyPropertyChangedEventArgs)

    Dim ctl As NumericUpDown = DirectCast(obj, NumericUpDown)
    Dim newValue As Integer = CInt(args.NewValue)

    ' Call UpdateStates because the Value might have caused the
    ' control to change ValueStates.
    ctl.UpdateStates(True)

    ' Call OnValueChanged to raise the ValueChanged event.
    ctl.OnValueChanged(New ValueChangedEventArgs(NumericUpDown.ValueChangedEvent, newValue))
End Sub
```

You might also need to update states when an event occurs. The following example shows that the `NumericUpDown` calls `UpdateStates` on the `Control` to handle the `GotFocus` event.

```
protected override void OnGotFocus(RoutedEventArgs e)
{
    base.OnGotFocus(e);
    UpdateStates(true);
}
```

```
Protected Overloads Overrides Sub OnGotFocus(ByVal e As RoutedEventArgs)
    MyBase.OnGotFocus(e)
    UpdateStates(True)
End Sub
```

The `VisualStateManager` helps you manage your control's states. By using the `VisualStateManager`, you ensure that your control correctly transitions between states. If you follow the recommendations described in this section for working with the `VisualStateManager`, your control's code will remain readable and maintainable.

## Providing the Control Contract

You provide a control contract so that `ControlTemplate` authors will know what to put in the template. A control contract has three elements:

- The visual elements that the control's logic uses.
- The states of the control and the group each state belongs to.
- The public properties that visually affect the control.

Someone that creates a new `ControlTemplate` needs to know what `FrameworkElement` objects the control's logic uses, what type each object is, and what its name is. A `ControlTemplate` author also needs to know the name of each possible state the control can be in, and which `VisualStateGroup` the state is in.

Returning to the `NumericUpDown` example, the control expects the `ControlTemplate` to have the following `FrameworkElement` objects:

- A `RepeatButton` called `upButton`.
- A `RepeatButton` called `DownButton`.

The control can be in the following states:

- In the `ValueStates` `VisualStateGroup`
  - `Positive`
  - `Negative`
- In the `FocusStates` `VisualStateGroup`
  - `Focused`
  - `Unfocused`

To specify what `FrameworkElement` objects the control expects, you use the `TemplatePartAttribute`, which specifies the name and type of the expected elements. To specify the possible states of a control, you use the `TemplateVisualStateAttribute`, which specifies the state's name and which `VisualStateGroup` it belongs to. Put the `TemplatePartAttribute` and `TemplateVisualStateAttribute` on the class definition of the control.

Any public property that affects the appearance of your control is also a part of the control contract.

The following example specifies the [FrameworkElement](#) object and states for the [NumericUpDown](#) control.

```
[TemplatePart(Name = "UpButtonElement", Type = typeof(RepeatButton))]
[TemplatePart(Name = "DownButtonElement", Type = typeof(RepeatButton))]
[TemplateVisualState(Name = "Positive", GroupName = "ValueStates")]
[TemplateVisualState(Name = "Negative", GroupName = "ValueStates")]
[TemplateVisualState(Name = "Focused", GroupName = "FocusedStates")]
[TemplateVisualState(Name = "Unfocused", GroupName = "FocusedStates")]
public class NumericUpDown : Control
{
    public static readonly DependencyProperty BackgroundProperty;
    public static readonly DependencyProperty BorderBrushProperty;
    public static readonly DependencyProperty BorderThicknessProperty;
    public static readonly DependencyProperty FontFamilyProperty;
    public static readonly DependencyProperty FontSizeProperty;
    public static readonly DependencyProperty FontStretchProperty;
    public static readonly DependencyProperty FontStyleProperty;
    public static readonly DependencyProperty FontWeightProperty;
    public static readonly DependencyProperty ForegroundProperty;
    public static readonly DependencyProperty HorizontalContentAlignmentProperty;
    public static readonly DependencyProperty PaddingProperty;
    public static readonly DependencyProperty TextAlignementProperty;
    public static readonly DependencyProperty TextDecorationsProperty;
    public static readonly DependencyProperty TextWrappingProperty;
    public static readonly DependencyProperty VerticalContentAlignmentProperty;

    public Brush Background { get; set; }
    public Brush BorderBrush { get; set; }
    public Thickness BorderThickness { get; set; }
    public FontFamily FontFamily { get; set; }
    public double FontSize { get; set; }
    public FontStretch FontStretch { get; set; }
    public FontStyle FontStyle { get; set; }
    public FontWeight FontWeight { get; set; }
    public Brush Foreground { get; set; }
    public HorizontalAlignment HorizontalContentAlignment { get; set; }
    public Thickness Padding { get; set; }
    public TextAlignment TextAlignement { get; set; }
    public TextDecorationCollection TextDecorations { get; set; }
    public TextWrapping TextWrapping { get; set; }
    public VerticalAlignment VerticalContentAlignment { get; set; }
}
```

```
<TemplatePart(Name:="UpButtonElement", Type:=GetType(RepeatButton))> _
<TemplatePart(Name:="DownButtonElement", Type:=GetType(RepeatButton))> _
<TemplateVisualState(Name:="Positive", GroupName:="ValueStates")> _
<TemplateVisualState(Name:="Negative", GroupName:="ValueStates")> _
<TemplateVisualState(Name:="Focused", GroupName:="FocusedStates")> _
<TemplateVisualState(Name:="Unfocused", GroupName:="FocusedStates")> _
Public Class NumericUpDown
    Inherits Control
    Public Shared ReadOnly BackgroundProperty As DependencyProperty
    Public Shared ReadOnly BorderBrushProperty As DependencyProperty
    Public Shared ReadOnly BorderThicknessProperty As DependencyProperty
    Public Shared ReadOnly FontFamilyProperty As DependencyProperty
    Public Shared ReadOnly FontSizeProperty As DependencyProperty
    Public Shared ReadOnly FontStretchProperty As DependencyProperty
    Public Shared ReadOnly FontStyleProperty As DependencyProperty
    Public Shared ReadOnly FontWeightProperty As DependencyProperty
    Public Shared ReadOnly ForegroundProperty As DependencyProperty
    Public Shared ReadOnly HorizontalContentAlignmentProperty As DependencyProperty
    Public Shared ReadOnly PaddingProperty As DependencyProperty
    Public Shared ReadOnly TextAlignementProperty As DependencyProperty
    Public Shared ReadOnly TextDecorationsProperty As DependencyProperty
```

```

Public Shared ReadOnly TextWrappingProperty As DependencyProperty
Public Shared ReadOnly VerticalContentAlignmentProperty As DependencyProperty


Private _Background As Brush
Public Property Background() As Brush
    Get
        Return _Background
    End Get
    Set(ByVal value As Brush)
        _Background = value
    End Set
End Property

Private _BorderBrush As Brush
Public Property BorderBrush() As Brush
    Get
        Return _BorderBrush
    End Get
    Set(ByVal value As Brush)
        _BorderBrush = value
    End Set
End Property

Private _BorderThickness As Thickness
Public Property BorderThickness() As Thickness
    Get
        Return _BorderThickness
    End Get
    Set(ByVal value As Thickness)
        _BorderThickness = value
    End Set
End Property

Private _FontFamily As FontFamily
Public Property FontFamily() As FontFamily
    Get
        Return _FontFamily
    End Get
    Set(ByVal value As FontFamily)
        _FontFamily = value
    End Set
End Property

Private _FontSize As Double
Public Property FontSize() As Double
    Get
        Return _FontSize
    End Get
    Set(ByVal value As Double)
        _FontSize = value
    End Set
End Property

Private _FontStretch As FontStretch
Public Property FontStretch() As FontStretch
    Get
        Return _FontStretch
    End Get
    Set(ByVal value As FontStretch)
        _FontStretch = value
    End Set
End Property

Private _FontStyle As FontStyle
Public Property FontStyle() As FontStyle
    Get
        Return _FontStyle
    End Get

```

```

        Set(ByVal value As FontStyle)
            _FontStyle = value
        End Set
    End Property

    Private _FontWeight As FontWeight
    Public Property FontWeight() As FontWeight
        Get
            Return _FontWeight
        End Get
        Set(ByVal value As FontWeight)
            _FontWeight = value
        End Set
    End Property

    Private _Foreground As Brush
    Public Property Foreground() As Brush
        Get
            Return _Foreground
        End Get
        Set(ByVal value As Brush)
            _Foreground = value
        End Set
    End Property

    Private _HorizontalContentAlignment As HorizontalAlignment
    Public Property HorizontalContentAlignment() As HorizontalAlignment
        Get
            Return _HorizontalContentAlignment
        End Get
        Set(ByVal value As HorizontalAlignment)
            _HorizontalContentAlignment = value
        End Set
    End Property

    Private _Padding As Thickness
    Public Property Padding() As Thickness
        Get
            Return _Padding
        End Get
        Set(ByVal value As Thickness)
            _Padding = value
        End Set
    End Property

    Private _TextAlignment As TextAlign
    Public Property TextAlign() As TextAlign
        Get
            Return _TextAlignment
        End Get
        Set(ByVal value As TextAlign)
            _TextAlignment = value
        End Set
    End Property

    Private _TextDecorations As TextDecorationCollection
    Public Property TextDecorations() As TextDecorationCollection
        Get
            Return _TextDecorations
        End Get
        Set(ByVal value As TextDecorationCollection)
            _TextDecorations = value
        End Set
    End Property

    Private _TextWrapping As TextWrapping
    Public Property TextWrapping() As TextWrapping
        Get
            Return _TextWrapping
        End Get
    End Property

```

```
    Return _TextWrapping
End Get
Set(ByVal value As TextWrapping)
    _TextWrapping = value
End Set
End Property

Private _VerticalContentAlignment As VerticalAlignment
Public Property VerticalContentAlignment() As VerticalAlignment
    Get
        Return _VerticalContentAlignment
    End Get
    Set(ByVal value As VerticalAlignment)
        _VerticalContentAlignment = value
    End Set
End Property
End Class
```

## Complete Example

The following example is the entire [ControlTemplate](#) for the `NumericUpDown` control.

```

        <DiscreteObjectKeyFrame.Value>
            <Visibility>Visible</Visibility>
        </DiscreteObjectKeyFrame.Value>
    </DiscreteObjectKeyFrame>
</Storyboard>
</VisualState>

        <!--Return the control to its initial state by
            hiding the focus rectangle.-->
<VisualState Name="Unfocused"/>
</VisualStateGroup>

</VisualStateManager.VisualStateGroups>

<Grid>
    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>

    <Border BorderThickness="1" BorderBrush="Gray"
        Margin="7,2,2,2" Grid.RowSpan="2"
        Background="#E0FFFFFF"
        VerticalAlignment="Center"
        HorizontalAlignment="Stretch">
        <!--Bind the TextBlock to the Value property-->
        <TextBlock Name="TextBlock"
            Width="60" TextAlignment="Right" Padding="5"
            Text="{Binding RelativeSource={RelativeSource FindAncestor,
                AncestorType={x:Type local:NumericUpDown}},
                Path=Value}"/>
    </Border>

    <RepeatButton Content="Up" Margin="2,5,5,0"
        Name="UpButton"
        Grid.Column="1" Grid.Row="0"/>
    <RepeatButton Content="Down" Margin="2,0,5,5"
        Name="DownButton"
        Grid.Column="1" Grid.Row="1"/>

    <Rectangle Name="FocusVisual" Grid.ColumnSpan="2" Grid.RowSpan="2"
        Stroke="Black" StrokeThickness="1"
        Visibility="Collapsed"/>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</ResourceDictionary>

```

The following example shows the logic for the `NumericUpDown`.

```

using System.Windows;
using System.Windows.Controls;
using System.Windows.Controls.Primitives;
using System.Windows.Input;
using System.Windows.Media;

namespace VSMCustomControl
{
    [TemplatePart(Name = "UpInputElement", Type = typeof(RepeatButton))]

```

```

[TemplatePart(Name = "UpButtonElement", Type = typeof(RepeatButton))]
[TemplatePart(Name = "DownButtonElement", Type = typeof(RepeatButton))]
[TemplateVisualState(Name = "Positive", GroupName = "ValueStates")]
[TemplateVisualState(Name = "Negative", GroupName = "ValueStates")]
[TemplateVisualState(Name = "Focused", GroupName = "FocusedStates")]
[TemplateVisualState(Name = "Unfocused", GroupName = "FocusedStates")]
public class NumericUpDown : Control
{
    public NumericUpDown()
    {
        DefaultStyleKey = typeof(NumericUpDown);
        this.IsTabStop = true;
    }

    public static readonly DependencyProperty ValueProperty =
        DependencyProperty.Register(
            "Value", typeof(int), typeof(NumericUpDown),
            new PropertyMetadata(
                new PropertyChangedCallback(ValueChangedCallback)));

    public int Value
    {
        get
        {
            return (int)GetValue(ValueProperty);
        }

        set
        {
            SetValue(ValueProperty, value);
        }
    }

    private static void ValueChangedCallback(DependencyObject obj,
        DependencyPropertyChangedEventArgs args)
    {
        NumericUpDown ctl = (NumericUpDown)obj;
        int newValue = (int)args.NewValue;

        // Call UpdateStates because the Value might have caused the
        // control to change ValueStates.
        ctl.UpdateStates(true);

        // Call OnValueChanged to raise the ValueChanged event.
        ctl.OnValueChanged(
            new ValueChangedEventArgs(NumericUpDown.ValueChangedEvent,
                newValue));
    }

    public static readonly RoutedEvent ValueChangedEvent =
        EventManager.RegisterRoutedEvent("ValueChanged", RoutingStrategy.Direct,
            typeof(ValueChangedEventHandler), typeof(NumericUpDown));

    public event ValueChangedEventHandler ValueChanged
    {
        add { AddHandler(ValueChangedEvent, value); }
        remove { RemoveHandler(ValueChangedEvent, value); }
    }

    protected virtual void OnValueChanged(ValueChangedEventArgs e)
    {
        // Raise the ValueChanged event so applications can be alerted
        // when Value changes.
        RaiseEvent(e);
    }

    private void UpdateStates(bool useTransitions)
    {
        if (Value >= 0)
        {
            if (useTransitions)
                VisualStateManager.GoToState(this, "Positive");
            else
                SetCurrentValue(ValueProperty, 0);
        }
        else
        {
            if (useTransitions)
                VisualStateManager.GoToState(this, "Negative");
            else
                SetCurrentValue(ValueProperty, 0);
        }
    }
}

```

```

        {
            VisualStateManager.GoToState(this, "Positive", useTransitions);
        }
        else
        {
            VisualStateManager.GoToState(this, "Negative", useTransitions);
        }

        if (IsFocused)
        {
            VisualStateManager.GoToState(this, "Focused", useTransitions);
        }
        else
        {
            VisualStateManager.GoToState(this, "Unfocused", useTransitions);
        }
    }

    public override void OnApplyTemplate()
    {
        UpButtonElement = GetTemplateChild("UpButton") as RepeatButton;
        DownButtonElement = GetTemplateChild("DownButton") as RepeatButton;
        //TextElement = GetTemplateChild("TextBlock") as TextBlock;

        UpdateStates(false);
    }

    private RepeatButton downButtonElement;

    private RepeatButton DownButtonElement
    {
        get
        {
            return downButtonElement;
        }

        set
        {
            if (downButtonElement != null)
            {
                downButtonElement.Click -=
                    new RoutedEventHandler(downButtonElement_Click);
            }
            downButtonElement = value;

            if (downButtonElement != null)
            {
                downButtonElement.Click +=
                    new RoutedEventHandler(downButtonElement_Click);
            }
        }
    }

    void downButtonElement_Click(object sender, RoutedEventArgs e)
    {
        Value--;
    }

    private RepeatButton upButtonElement;

    private RepeatButton UpButtonElement
    {
        get
        {
            return upButtonElement;
        }

        set
        {
    
```

```

        if (upButtonElement != null)
        {
            upButtonElement.Click -=
                new RoutedEventHandler(upButtonElement_Click);
        }
        upButtonElement = value;

        if (upButtonElement != null)
        {
            upButtonElement.Click +=
                new RoutedEventHandler(upButtonElement_Click);
        }
    }

    void upButtonElement_Click(object sender, RoutedEventArgs e)
    {
        Value++;
    }

    protected override void OnMouseLeftButtonDown(MouseButtonEventArgs e)
    {
        base.OnMouseLeftButtonDown(e);
        Focus();
    }

    protected override void OnGotFocus(RoutedEventArgs e)
    {
        base.OnGotFocus(e);
        UpdateStates(true);
    }

    protected override void OnLostFocus(RoutedEventArgs e)
    {
        base.OnLostFocus(e);
        UpdateStates(true);
    }
}

public delegate void ValueChangedEventHandler(object sender, ValueChangedEventArgs e);

public class ValueChangedEventArgs : RoutedEventArgs
{
    private int _value;

    public ValueChangedEventArgs(RoutedEventArgs id, int num)
    {
        _value = num;
        RoutedEvent = id;
    }

    public int Value
    {
        get { return _value; }
    }
}
}

```

```

Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Controls.Primitives
Imports System.Windows.Input
Imports System.Windows.Media

<TemplatePart(Name:="UpButtonElement", Type:=GetType(RepeatButton))> _
<TemplatePart(Name:="DownButtonElement", Type:=GetType(RepeatButton))> _

```

```

<TemplateVisualState(Name:="Positive", GroupName:="ValueStates")> _
<TemplateVisualState(Name:="Negative", GroupName:="ValueStates")> _
<TemplateVisualState(Name:="Focused", GroupName:="FocusedStates")> _
<TemplateVisualState(Name:="Unfocused", GroupName:="FocusedStates")> _

Public Class NumericUpDown
    Inherits Control

    Public Sub New()
        DefaultStyleKeyProperty.OverrideMetadata(GetType(NumericUpDown), New
FrameworkPropertyMetadata(GetType(NumericUpDown)))
        Me.IsTabStop = True
    End Sub

    Public Shared ReadOnly ValueProperty As DependencyProperty =
        DependencyProperty.Register("Value", GetType(Integer), GetType(NumericUpDown),
            New PropertyMetadata(New PropertyChangedCallback(AddressOf ValueChangedCallback)))

    Public Property Value() As Integer

        Get
            Return CIntGetValue(ValueProperty)
        End Get

        Set(ByVal value As Integer)

            SetValue(ValueProperty, value)
        End Set
    End Property

    Private Shared Sub ValueChangedCallback(ByVal obj As DependencyObject,
        ByVal args As DependencyPropertyChangedEventArgs)

        Dim ctl As NumericUpDown = DirectCast(obj, NumericUpDown)
        Dim newValue As Integer = CInt(args.NewValue)

        ' Call UpdateStates because the Value might have caused the
        ' control to change ValueStates.
        ctl.UpdateStates(True)

        ' Call OnValueChanged to raise the ValueChanged event.
        ctl.OnValueChanged(New ValueChangedEventArgs(NumericUpDown.ValueChangedEvent, newValue))
    End Sub

    Public Shared ReadOnly ValueChangedEvent As RoutedEvent =
        EventManager.RegisterRoutedEvent("ValueChanged", RoutingStrategy.Direct,
            GetType(ValueChangedEventHandler), GetType(NumericUpDown))

    Public Custom Event ValueChanged As ValueChangedEventHandler

        AddHandler(ByVal value As ValueChangedEventHandler)
            Me.AddHandler(ValueChangedEvent, value)
        End AddHandler

        RemoveHandler(ByVal value As ValueChangedEventHandler)
            Me.RemoveHandler(ValueChangedEvent, value)
        End RemoveHandler

        RaiseEvent(ByVal sender As Object, ByVal e As RoutedEventArgs)
            Me.RaiseEvent(e)
        End RaiseEvent

    End Event

    Protected Overridable Sub OnValueChanged(ByVal e As ValueChangedEventArgs)
        ' Raise the ValueChanged event so applications can be alerted
        ' when Value changes.
        MyBase.RaiseEvent(e)
    End Sub

```

```

#Region "NUDCode"
    Private Sub UpdateStates(ByVal useTransitions As Boolean)

        If Value >= 0 Then
            VisualStateManager.GoToState(Me, "Positive", useTransitions)
        Else
            VisualStateManager.GoToState(Me, "Negative", useTransitions)
        End If

        If IsFocused Then
            VisualStateManager.GoToState(Me, "Focused", useTransitions)
        Else
            VisualStateManager.GoToState(Me, "Unfocused", useTransitions)
        End If
    End Sub

    Public Overloads Overrides Sub OnApplyTemplate()

        UpButtonElement = TryCast(GetTemplateChild("UpButton"), RepeatButton)
        DownButtonElement = TryCast(GetTemplateChild("DownButton"), RepeatButton)

        UpdateStates(False)
    End Sub

    Private m_downButtonElement As RepeatButton

    Private Property DownButtonElement() As RepeatButton
        Get
            Return m_downButtonElement
        End Get

        Set(ByVal value As RepeatButton)

            If m_downButtonElement IsNot Nothing Then
                RemoveHandler m_downButtonElement.Click, AddressOf downButtonElement_Click
            End If
            m_downButtonElement = value

            If m_downButtonElement IsNot Nothing Then
                AddHandler m_downButtonElement.Click, AddressOf downButtonElement_Click
            End If
        End Set
    End Property

    Private Sub downButtonElement_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
        Value -= 1
    End Sub

    Private m_upButtonElement As RepeatButton

    Private Property UpButtonElement() As RepeatButton
        Get
            Return m_upButtonElement
        End Get

        Set(ByVal value As RepeatButton)
            If m_upButtonElement IsNot Nothing Then
                RemoveHandler m_upButtonElement.Click, AddressOf upButtonElement_Click
            End If
            m_upButtonElement = value

            If m_upButtonElement IsNot Nothing Then
                AddHandler m_upButtonElement.Click, AddressOf upButtonElement_Click
            End If
        End Set
    End Property

```

```

Private Sub upButtonElement_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    Value += 1
End Sub

Protected Overloads Overrides Sub OnMouseLeftButtonDown(ByVal e As MouseButtonEventArgs)
    MyBase.OnMouseLeftButtonDown(e)
    Focus()
End Sub

Protected Overloads Overrides Sub OnGotFocus(ByVal e As RoutedEventArgs)
    MyBase.OnGotFocus(e)
    UpdateStates(True)
End Sub

Protected Overloads Overrides Sub OnLostFocus(ByVal e As RoutedEventArgs)
    MyBase.OnLostFocus(e)
    UpdateStates(True)
End Sub
#End Region
End Class

Public Delegate Sub ValueChangedEventHandler(ByVal sender As Object,
                                            ByVal e As ValueChangedEventArgs)

Public Class ValueChangedEventArgs
    Inherits RoutedEventArgs
    Private _value As Integer

    Public Sub New(ByVal id As RoutedEvent,
                  ByVal num As Integer)
        _value = num
        RoutedEvent = id
    End Sub

    Public ReadOnly Property Value() As Integer
        Get
            Return _value
        End Get
    End Property
End Class

```

## See also

- [Create a template for a control](#)
- [Control Customization](#)

# Guidelines for Designing Stylistable Controls

8 minutes to read • [Edit Online](#)

This document summarizes a set of best practices to consider when designing a control which you intend to be easily stylistable and templatable. We came to this set of best practices through a lot of trial and error while working on the theme control styles for the built-in WPF control set. We learned that successful styling is as much a function of a well-designed object model as it is of the style itself. The intended audience for this document is the control author, not the style author.

## Terminology

"Styling and templating" refer to the suite of technologies that enable a control author to defer the visual aspects of the control to the style and template of the control. This suite of technologies includes:

- Styles (including property setters, triggers, and storyboards).
- Resources.
- Control templates.
- Data templates.

For an introduction to styling and templating, see [Styling and Templating](#).

## Before You Start: Understanding Your Control

Before you jump into these guidelines, it is important to understand and have defined the common usage of your control. Styling exposes an often unruly set of possibilities. Controls that are written to be used broadly (in many applications, by many developers) face the challenge that styling can be used to make far-reaching changes to the visual appearance of the control. In fact, the styled control may not even resemble the control author's intentions. Since the flexibility offered by styling is essentially boundless, you can use the idea of common usage to help you scope your decisions.

To understand your control's common usage, it's good to think about the value proposition of the control. What does your control bring to the table that no other control can offer? Common usage does not imply any specific visual appearance, but rather the philosophy of the control and a reasonable set of expectations about its usage. This understanding allows you to make some assumptions about the composition model and the style-defined behaviors of the control in the common case. In the case of [ComboBox](#), for example, understanding the common usage won't give you any insight about whether a particular [ComboBox](#) has rounded corners, but it will give you insight into the fact that the [ComboBox](#) probably needs a pop-up window and some way of toggling whether it is open.

## General Guidelines

- **Do not strictly enforce template contracts.** The template contract of a control might consist of elements, commands, bindings, triggers, or even property settings that are required or expected for a control to function properly.
  - Minimize contracts as much as possible.
  - Design around the expectation that during design time (that is, when using a design tool) it is common for a control template to be in an incomplete state. WPF does not offer a "composing" state

infrastructure, so controls have to be built with the expectation that such a state might be valid.

- Do not throw exceptions when any aspect of a template contract is not followed. Along these lines, panels should not throw exceptions if they have too many or too few children.
- **Factor peripheral functionality into template helper elements.** Each control should be focused on its core functionality and true value proposition and defined by the control's common usage. To that end, use composition and helper elements within the template to enable peripheral behaviors and visualizations, that is, those behaviors and visualizations that do not contribute to the core functionality of the control. Helper elements fall into three categories:
  - **Standalone** helper types are public and reusable controls or primitives that are used "anonymously" in a template, meaning that neither the helper element nor the styled control is aware of the other. Technically, any element can be an anonymous type, but in this context the term describes those types that encapsulate specialized functionality to enable targeted scenarios.
  - **Type-based** helper elements are new types that encapsulate specialized functionality. These elements are typically designed with a narrower range of functionality than common controls or primitives. Unlike standalone helper elements, type-based helper elements are aware of the context in which they are used and typically must share data with the control to whose template they belong.
  - **Named** helper elements are common controls or primitives that a control expects to find within its template by name. These elements are given a well-known name within the template, making it possible for a control to find the element and interact with it programmatically. There can only be one element with a given name in any template.

The following table shows helper elements employed by control styles today (this list is not exhaustive):

ELEMENT	TYPE	USED BY
ContentPresenter	Type-based	Button, CheckBox, RadioButton, Frame, and so on (all ContentControl types)
ItemsPresenter	Type-based	ListBox, ComboBox, Menu, and so on (all ItemsControl types)
ToolBarOverflowPanel	Named	ToolBar
Popup	Standalone	ComboBox, ToolBar, Menu, ToolTip, and so on
RepeatButton	Named	Slider, ScrollBar, and so on
ScrollBar	Named	ScrollViewer
ScrollViewer	Standalone	ListBox, ComboBox, Menu, Frame, and so on
TabPanel	Standalone	TabControl
TextBox	Named	ComboBox
TickBar	Type-based	Slider

- **Minimize required user-specified bindings or property settings on helper elements.** It is common

for a helper element to require certain bindings or property settings in order to function properly within the control template. The helper element and templated control should, as much as possible, establish these settings. When setting properties or establishing bindings, care should be taken to not override values set by the user. Specific best practices are as follows:

- Named helper elements should be identified by the parent and the parent should establish any required settings on the helper element.
  - Type-based helper elements should establish any required settings directly on themselves. Doing this may require the helper element to query for information context in which it is being used, including its `TemplatedParent` (the control type of the template in which it is being used). For example, `ContentPresenter` automatically binds the `Content` property of its `TemplatedParent` to its `Content` property when used in a `ContentControl` derived type.
  - Standalone helper elements cannot be optimized in this way because, by definition, neither the helper element nor the parent knows about the other.
- **Use the `Name` property to flag elements within a template.** A control that needs to find an element in its style in order to access it programmatically should do so using the `Name` property and the `FindName` paradigm. A control should not throw an exception when an element is not found, but silently and gracefully disable the functionality which required that element.
  - **Use best practices for expressing control state and behavior in a style.** The following is an ordered list of best practices for expressing control state changes and behavior in a style. You should use the first item on the list that enables your scenario.
    1. Property binding. Example: binding between `ComboBox.IsDropDownOpen` and `ToggleButton.IsChecked`.
    2. Triggered property changes or property animations. Example: the hover state of a `Button`.
    3. Command. Example: `LineUpCommand` / `LineDownCommand` in `ScrollBar`.
    4. Standalone helper elements. Example: `TabPanel` in `TabControl`.
    5. Type-based helper types. Example: `ContentPresenter` in `Button`, `TickBar` in `Slider`.
    6. Named helper elements. Example: `TextBox` in `ComboBox`.
    7. Bubbled events from named helper types. If you listen for bubbled events from a style element, you should require that the element generating the event can be uniquely identified. Example: `Thumb` in `ToolBar`.
    8. Custom `OnRender` behavior. Example: `ButtonChrome` in `Button`.
  - **Use style triggers (as opposed to template triggers) sparingly.** Triggers that affect properties on elements in the template must be declared in the template. Triggers that affect properties on the control (no `TargetName`) may be declared in the style unless you know that changing the template should also destroy the trigger.
  - **Be consistent with existing styling patterns.** Many times there are multiple ways to solve a problem. Be aware of and, when possible, consistent with existing control styling patterns. This is especially important for controls that derive from the same base type (for example, `ContentControl`, `ItemsControl`, `RangeBase`, and so on).
  - **Expose properties to enable common customization scenarios without retemplating.** WPF does not support pluggable/customizable parts, so a control user is left with only two methods of customization: setting properties directly or setting properties using styles. With that in mind, it is appropriate to surface a limited number of properties targeted at very common, high-priority customization scenarios which would

otherwise require the retemplating. Here are best practices for when and how to enable customization scenarios:

- Very common customizations should be exposed as properties on the control and consumed by the template.
- Less common (though not rare) customizations should be exposed as attached properties and consumed by the template.
- It is acceptable for known but rare customizations to require retemplating.

## Theme Considerations

- **Theme styles should attempt to have consistent property semantics across all themes, but make no guarantee.** As part of its documentation, your control should have a document describing the control's property semantics, that is, the "meaning" of a property for a control. For example, the [ComboBox](#) control should define the meaning of the [Background](#) property within [ComboBox](#). The default styles for your control should attempt to follow the semantics defined in that document across all themes. Control users, on the other hand, should be aware that property semantics can change from theme to theme. In certain cases, a given property may not be expressible under the visual constraints required by a particular theme. (The Classic theme, for example, does not have a single border to which [Thickness](#) can be applied for many controls.)
- **Theme styles do not need to have consistent trigger semantics across all themes.** The behavior exposed by a control style through triggers or animations may vary from theme to theme. Control users should be aware that a control will not necessarily employ the same mechanism to achieve a particular behavior across all themes. One theme, for example, may use an animation to express hover behavior where another theme uses a trigger. This can result in inconsistencies in behavior preservation on customized controls. (Changing the background property, for example, might not affect the hover state of the control if that state is expressed using a trigger. However, if the hover state is implemented using an animation, changing to background could irreparably break the animation and therefore the state transition.)
- **Theme styles do not need to have consistent "layout" semantics across all themes.** For example, the default style does not need to guarantee that a control will occupy the same amount of size in all themes or guarantee that a control will have the same content margins / padding across all themes.

## See also

- [Styling and Templating](#)
- [Control Authoring Overview](#)

# Adorners

2 minutes to read • [Edit Online](#)

This section provides information about Adorners and the Windows Presentation Foundation (WPF) Adorner framework.

## In This Section

[Adorners Overview](#)

[How-to Topics](#)

## Reference

[AdornedElementPlaceholder](#)

[Adorner](#)

[AdornerDecorator](#)

[AdornerHitTestResult](#)

[AdornerLayer](#)

## Related Sections

# Adorners Overview

4 minutes to read • [Edit Online](#)

Adorners are a special type of [FrameworkElement](#), used to provide visual cues to a user. Among other uses, Adorners can be used to add functional handles to elements or provide state information about a control.

## About Adorners

An [Adorner](#) is a custom [FrameworkElement](#) that is bound to a [UIElement](#). Adorners are rendered in an [AdornerLayer](#), which is a rendering surface that is always on top of the adorned element or a collection of adorned elements. Rendering of an adorner is independent from rendering of the [UIElement](#) that the adorner is bound to. An adorner is typically positioned relative to the element to which it is bound, using the standard 2-D coordinate origin located at the upper-left of the adorned element.

Common applications for adorners include:

- Adding functional handles to a [UIElement](#) that enable a user to manipulate the element in some way (resize, rotate, reposition, etc.).
- Provide visual feedback to indicate various states, or in response to various events.
- Overlay visual decorations on a [UIElement](#).
- Visually mask or override part or all of a [UIElement](#).

Windows Presentation Foundation (WPF) provides a basic framework for adorning visual elements. The following table lists the primary types used when adorning objects, and their purpose. Several usage examples follow:

<a href="#">Adorner</a>	An abstract base class from which all concrete adorner implementations inherit.
<a href="#">AdornerLayer</a>	A class representing a rendering layer for the adorner(s) of one or more adorned elements.
<a href="#">AdornerDecorator</a>	A class that enables an adorner layer to be associated with a collection of elements.

## Implementing a Custom Adorner

The adorners framework provided by Windows Presentation Foundation (WPF) is intended primarily to support the creation of custom adorners. A custom adorner is created by implementing a class that inherits from the abstract [Adorner](#) class.

### NOTE

The parent of an [Adorner](#) is the [AdornerLayer](#) that renders the [Adorner](#), not the element being adorned.

The following example shows a class that implements a simple adorner. The example adorner simply adorns the corners of a [UIElement](#) with circles.

```

// Adorners must subclass the abstract base class Adorner.
public class SimpleCircleAdorner : Adorner
{
    // Be sure to call the base class constructor.
    public SimpleCircleAdorner(UIElement adornedElement)
        : base(adornedElement)
    {
    }

    // A common way to implement an adorer's rendering behavior is to override the OnRender
    // method, which is called by the layout system as part of a rendering pass.
    protected override void OnRender(DrawingContext drawingContext)
    {
        Rect adornedElementRect = new Rect(this.AdornedElement.DesiredSize);

        // Some arbitrary drawing implements.
        SolidColorBrush renderBrush = new SolidColorBrush(Colors.Green);
        renderBrush.Opacity = 0.2;
        Pen renderPen = new Pen(new SolidColorBrush(Colors.Navy), 1.5);
        double renderRadius = 5.0;

        // Draw a circle at each corner.
        drawingContext.DrawEllipse(renderBrush, renderPen, adornedElementRect.TopLeft, renderRadius,
        renderRadius);
        drawingContext.DrawEllipse(renderBrush, renderPen, adornedElementRect.TopRight, renderRadius,
        renderRadius);
        drawingContext.DrawEllipse(renderBrush, renderPen, adornedElementRect.BottomLeft, renderRadius,
        renderRadius);
        drawingContext.DrawEllipse(renderBrush, renderPen, adornedElementRect.BottomRight, renderRadius,
        renderRadius);
    }
}

```

```

Public Class SimpleCircleAdorner
    Inherits Adorner
    Sub New(ByVal adornedElement As UIElement)
        MyBase.New(adornedElement)
    End Sub

    Protected Overrides Sub OnRender(ByVal drawingContext As System.Windows.Media.DrawingContext)
        MyBase.OnRender(drawingContext)
        Dim adornedElementRect As New Rect(AdornedElement.DesiredSize)
        Dim renderBrush As New SolidColorBrush(Colors.Green)
        renderBrush.Opacity = 0.2
        Dim renderPen As New Pen(New SolidColorBrush(Colors.Navy), 1.5)
        Dim renderRadius As Double
        renderRadius = 5.0

        'Draw a circle at each corner.
        drawingContext.DrawEllipse(renderBrush, renderPen, adornedElementRect.TopLeft, renderRadius,
        renderRadius)
        drawingContext.DrawEllipse(renderBrush, renderPen, adornedElementRect.TopRight, renderRadius,
        renderRadius)
        drawingContext.DrawEllipse(renderBrush, renderPen, adornedElementRect.BottomLeft, renderRadius,
        renderRadius)
        drawingContext.DrawEllipse(renderBrush, renderPen, adornedElementRect.BottomRight, renderRadius,
        renderRadius)
    End Sub
End Class

```

The following image shows the SimpleCircleAdorner applied to a [TextBox](#):



## Rendering Behavior for Adorners

It is important to note that adorners do not include any inherent rendering behavior; ensuring that an adorner renders is the responsibility of the adorner implementer. A common way of implementing rendering behavior is to override the [OnRender](#) method and use one or more [DrawingContext](#) objects to render the adorner's visuals as needed (as shown in the example above).

### NOTE

Anything placed in the adorner layer is rendered on top of the rest of any styles you have set. In other words, adorners are always visually on top and cannot be overridden using z-order.

## Events and Hit Testing

Adorners receive input events just like any other [FrameworkElement](#). Because an adorner always has a higher z-order than the element it adorns, the adorner receives input events (such as [Drop](#) or [MouseMove](#)) that may be intended for the underlying adorned element. An adorner can listen for certain input events and pass these on to the underlying adorned element by re-raising the event.

To enable pass-through hit testing of elements under an adorner, set the hit test [IsHitTestVisible](#) property to **false** on the adorner. For more information about hit testing, see [Hit Testing in the Visual Layer](#).

## Adorning a Single UIElement

To bind an adorner to a particular [UIElement](#), follow these steps:

1. Call the static method [GetAdornerLayer](#) to get an [AdornerLayer](#) object for the [UIElement](#) to be adorned. [GetAdornerLayer](#) walks up the visual tree, starting at the specified [UIElement](#), and returns the first adorner layer it finds. (If no adorner layers are found, the method returns null.)
2. Call the [Add](#) method to bind the adorner to the target [UIElement](#).

The following example binds a [SimpleCircleAdorner](#) (shown above) to a [TextBox](#) named *myTextBox*:

```
myAdornerLayer = AdornerLayer.GetAdornerLayer(myTextBox);
myAdornerLayer.Add(new SimpleCircleAdorner(myTextBox));
```

```
myAdornerLayer = AdornerLayer.GetAdornerLayer(myTextBox)
myAdornerLayer.Add(New SimpleCircleAdorner(myTextBox))
```

### NOTE

Using Extensible Application Markup Language (XAML) to bind an adorner to another element is currently not supported.

## Adorning the Children of a Panel

To bind an adorner to the children of a [Panel](#), follow these steps:

1. Call the [static](#) method [GetAdornerLayer](#) to find an adorner layer for the element whose children are to

be adorned.

2. Enumerate through the children of the parent element and call the [Add](#) method to bind an adorer to each child element.

The following example binds a [SimpleCircleAdorner](#) (shown above) to the children of a [StackPanel](#) named *myStackPanel*:

```
foreach (UIElement toAdorn in myStackPanel.Children)
    myAdornerLayer.Add(new SimpleCircleAdorner(toAdorn));
```

```
For Each toAdorn As UIElement In myStackPanel.Children
    myAdornerLayer.Add(New SimpleCircleAdorner(toAdorn))
Next
```

## See also

- [AdornerHitTestResult](#)
- [Shapes and Basic Drawing in WPF Overview](#)
- [Painting with Images, Drawings, and Visuals](#)
- [Drawing Objects Overview](#)
- [How-to Topics](#)

# Adorners How-to Topics

2 minutes to read • [Edit Online](#)

The following examples demonstrate how to accomplish common tasks using the Windows Presentation Foundation (WPF) adorner framework.

## In This Section

[Implement an Adorner](#)

[Bind an Adorner to an Element](#)

[Adorn the Children of a Panel](#)

[Remove an Adorner from an Element](#)

[Remove all Adorners from an Element](#)

## Reference

[AdornedElementPlaceholder](#)

[Adorner](#)

[AdornerDecorator](#)

[AdornerHitTestResult](#)

[AdornerLayer](#)

## Related Sections

# How to: Implement an Adorner

2 minutes to read • [Edit Online](#)

This example shows a minimal adorner implementation.

## Notes for Implementers

It is important to note that adorners do not include any inherent rendering behavior; ensuring that an adorner renders is the responsibility of the adorner implementer. A common way of implementing rendering behavior is to override the [OnRender](#) method and use one or more [DrawingContext](#) objects to render the adorner's visuals as needed (as shown in this example).

## Example

### Description

A custom adorner is created by implementing a class that inherits from the abstract [Adorner](#) class. The example adorner simply adorns the corners of a [UIElement](#) with circles by overriding the [OnRender](#) method.

### Code

```
// Adorners must subclass the abstract base class Adorner.
public class SimpleCircleAdorner : Adorner
{
    // Be sure to call the base class constructor.
    public SimpleCircleAdorner(UIElement adornedElement)
        : base(adornedElement)
    {
    }

    // A common way to implement an adorner's rendering behavior is to override the OnRender
    // method, which is called by the layout system as part of a rendering pass.
    protected override void OnRender(DrawingContext drawingContext)
    {
        Rect adornedElementRect = new Rect(this.AdornedElement.DesiredSize);

        // Some arbitrary drawing implements.
        SolidColorBrush renderBrush = new SolidColorBrush(Colors.Green);
        renderBrush.Opacity = 0.2;
        Pen renderPen = new Pen(new SolidColorBrush(Colors.Navy), 1.5);
        double renderRadius = 5.0;

        // Draw a circle at each corner.
        drawingContext.DrawEllipse(renderBrush, renderPen, adornedElementRect.TopLeft, renderRadius,
        renderRadius);
        drawingContext.DrawEllipse(renderBrush, renderPen, adornedElementRect.TopRight, renderRadius,
        renderRadius);
        drawingContext.DrawEllipse(renderBrush, renderPen, adornedElementRect.BottomLeft, renderRadius,
        renderRadius);
        drawingContext.DrawEllipse(renderBrush, renderPen, adornedElementRect.BottomRight, renderRadius,
        renderRadius);
    }
}
```

```
Public Class SimpleCircleAdorner
    Inherits Adorner
    Sub New(ByVal adornedElement As UIElement)
        MyBase.New(adornedElement)
    End Sub

    Protected Overrides Sub OnRender(ByVal drawingContext As System.Windows.Media.DrawingContext)
        MyBase.OnRender(drawingContext)
        Dim adornedElementRect As New Rect(AdornedElement.DesiredSize)
        Dim renderBrush As New SolidColorBrush(Colors.Green)
        renderBrush.Opacity = 0.2
        Dim renderPen As New Pen(New SolidColorBrush(Colors.Navy), 1.5)
        Dim renderRadius As Double
        renderRadius = 5.0

        'Draw a circle at each corner.
        drawingContext.DrawEllipse(renderBrush, renderPen, adornedElementRect.TopLeft, renderRadius,
        renderRadius)
        drawingContext.DrawEllipse(renderBrush, renderPen, adornedElementRect.TopRight, renderRadius,
        renderRadius)
        drawingContext.DrawEllipse(renderBrush, renderPen, adornedElementRect.BottomLeft, renderRadius,
        renderRadius)
        drawingContext.DrawEllipse(renderBrush, renderPen, adornedElementRect.BottomRight, renderRadius,
        renderRadius)
    End Sub
End Class
```

## See also

- [Adorners Overview](#)

# How to: Bind an Adorner to an Element

2 minutes to read • [Edit Online](#)

This example shows how to programmatically bind an adorner to a specified **UIElement**.

## Example

To bind an adorner to a particular **UIElement**, follow these steps:

1. Call the `static` method `GetAdornerLayer` to get an `AdornerLayer` object for the **UIElement** to be adorned. `GetAdornerLayer` walks up the visual tree, starting at the specified **UIElement**, and returns the first adorner layer it finds. (If no adorner layers are found, the method returns null.)
2. Call the `Add` method to bind the adorner to the target **UIElement**.

The following example binds a `SimpleCircleAdorner` (shown above) to a `TextBox` named `myTextBox`.

```
myAdornerLayer = AdornerLayer.GetAdornerLayer(myTextBox);
myAdornerLayer.Add(new SimpleCircleAdorner(myTextBox));
```

```
myAdornerLayer = AdornerLayer.GetAdornerLayer(myTextBox)
myAdornerLayer.Add(New SimpleCircleAdorner(myTextBox))
```

### NOTE

Using Extensible Application Markup Language (XAML) to bind an adorner to another element is currently not supported.

## See also

- [Adorners Overview](#)

# How to: Adorn the Children of a Panel

2 minutes to read • [Edit Online](#)

This example shows how to programmatically bind an adorner to the children of a specified [Panel](#).

## Example

To bind an adorner to the children of a [Panel](#), follow these steps:

1. Declare a new [AdornerLayer](#) object and call the `static GetAdornerLayer` method to find an adorner layer for the element whose children are to be adorned.
2. Enumerate through the children of the parent element and call the [Add](#) method to bind an adorner to each child element.

The following example binds a [SimpleCircleAdorner](#) (shown above) to the children of a [StackPanel](#) named *myStackPanel*.

```
foreach (UIElement toAdorn in myStackPanel.Children)
    myAdornerLayer.Add(new SimpleCircleAdorner(toAdorn));
```

```
For Each toAdorn As UIElement In myStackPanel.Children
    myAdornerLayer.Add(New SimpleCircleAdorner(toAdorn))
Next
```

### NOTE

Using Extensible Application Markup Language (XAML) to bind an adorner to another element is currently not supported.

## See also

- [Adorners Overview](#)

# How to: Remove an Adorner from an Element

2 minutes to read • [Edit Online](#)

This example shows how to programmatically remove a specific adorer from a specified [UIElement](#).

## Example

This verbose code example removes the first adorer in the array of adorners returned by [GetAdorners](#). This example happens to retrieve the adorners on a [UIElement](#) named *myTextBox*. If the element specified in the call to [GetAdorners](#) has no adorners, `null` is returned. This code explicitly checks for a null array, and is best suited for applications where a null array is expected to be relatively common.

```
Adorner[] toRemoveArray = myAdornerLayer.GetAdorners(myTextBox);
Adorner toRemove;
if (toRemoveArray != null)
{
    toRemove = toRemoveArray[0];
    myAdornerLayer.Remove(toRemove);
}
```

```
Dim toRemoveArray() As Adorner = myAdornerLayer.GetAdorners(myTextBox)
Dim toRemove As Adorner
If toRemoveArray IsNot Nothing Then
    toRemove = toRemoveArray(0)
    myAdornerLayer.Remove(toRemove)
End If
```

## Example

This condensed code example is functionally equivalent to the verbose example shown above. This code does not explicitly check for a null array, so it is possible that a [NullReferenceException](#) exception may be raised. This code is best suited for applications where a null array is expected to be rare.

```
try { myAdornerLayer.Remove((myAdornerLayer.GetAdorners(myTextBox))[0]); } catch { }
```

```
Try
    myAdornerLayer.Remove((myAdornerLayer.GetAdorners(myTextBox))(0))
Catch
End Try
```

## See also

- [Adorners Overview](#)

# How to: Remove all Adorners from an Element

2 minutes to read • [Edit Online](#)

This example shows how to programmatically remove all adorners from a specified [UIElement](#).

## Example

This verbose code example removes all of the adorners in the array of adorners returned by [GetAdorners](#). This example happens to retrieve the adorners on a [UIElement](#) named *myTextBox*. If the element specified in the call to [GetAdorners](#) has no adorners, `null` is returned. This code explicitly checks for a null array, and is best suited for applications where a null array is expected to be relatively common.

```
Adorner[] toRemoveArray = myAdornerLayer.GetAdorners(myTextBox);
if (toRemoveArray != null)
{
    for (int x = 0; x < toRemoveArray.Length; x++)
    {
        myAdornerLayer.Remove(toRemoveArray[x]);
    }
}
```

```
toRemoveArray = myAdornerLayer.GetAdorners(myTextBox)
If toRemoveArray IsNot Nothing Then
    For x As Integer = 0 To toRemoveArray.Length - 1
        myAdornerLayer.Remove(toRemoveArray(x))
    Next x
End If
```

## Example

This condensed code example is functionally equivalent to the verbose example shown above. This code does not explicitly check for a null array, so it is possible that a [NullReferenceException](#) exception may be raised. This code is best suited for applications where a null array is expected to be rare.

```
try { foreach (Adorner toRemove in myAdornerLayer.GetAdorners(myTextBox)) myAdornerLayer.Remove(toRemove); }
catch { }
```

```
Try
    For Each toRemove In myAdornerLayer.GetAdorners(myTextBox)
        myAdornerLayer.Remove(toRemove)
    Next toRemove
Catch
End Try
```

## See also

- [Adorners Overview](#)

# Control Styles and Templates

2 minutes to read • [Edit Online](#)

Controls in Windows Presentation Foundation (WPF) have a [ControlTemplate](#) that contains the visual tree of that control. You can change the structure and appearance of a control by modifying the [ControlTemplate](#) of that control. There is no way to replace only part of the visual tree of a control; to change the visual tree of a control you must set the [Template](#) property of the control to its new and complete [ControlTemplate](#).

The desktop themes determine which resource dictionary is used. To get the resource dictionaries for the desktop themes, see [Default WPF Themes](#).

The following table describes the resource dictionary file names and their corresponding desktop themes.

THEME FILE	DESKTOP THEME
Classic.xaml	Classic Windows look (from Windows 95, Windows 98, and Windows 2000) on the Windows XP operating system..
Luna.NormalColor.xaml	Default blue theme on Windows XP.
Luna.Homestead.xaml	Olive theme on Windows XP.
Luna.Metallic.xaml	Silver theme on Windows XP.
Royale.NormalColor.xaml	Default theme on the Windows XP Media Center Edition operating system.
Aero.NormalColor.xaml	Default theme on the Windows Vista operating system.

## In This Section

- [Button Styles and Templates](#)
- [Calendar Styles and Templates](#)
- [CheckBox Styles and Templates](#)
- [ComboBox Styles and Templates](#)
- [ContextMenu Styles and Templates](#)
- [DataGrid Styles and Templates](#)
- [DatePicker Styles and Templates](#)
- [DocumentViewer Styles and Templates](#)
- [Expander Styles and Templates](#)
- [Frame Styles and Templates](#)
- [GroupBox Styles and Templates](#)
- [Label Styles and Templates](#)
- [ListBox Styles and Templates](#)
- [ListView Styles and Templates](#)
- [Menu Styles and Templates](#)
- [NavigationWindow Styles and Templates](#)
- [PasswordBox Styles and Templates](#)
- [ProgressBar Styles and Templates](#)
- [RadioButton Styles and Templates](#)

[RepeatButton Styles and Templates](#)  
[ScrollBar Styles and Templates](#)  
[ScrollViewer Styles and Templates](#)  
[Slider Styles and Templates](#)  
[StatusBar Styles and Templates](#)  
[TabControl Styles and Templates](#)  
[TextBox Styles and Templates](#)  
[Thumb Styles and Templates](#)  
[ToggleButton Styles and Templates](#)  
[ToolBar Styles and Templates](#)  
[ToolTip Styles and Templates](#)  
[TreeView Styles and Templates](#)  
[Window Styles and Templates](#)

## Reference

[System.Windows.Controls](#)

[ControlTemplate](#)

## Related Sections

[Control Authoring Overview](#)

[Styling and Templating](#)

# Button Styles and Templates

2 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [Button](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## Button Parts

The [Button](#) control does not have any named parts.

## Button States

The following table lists the visual states for the [Button](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
MouseOver	CommonStates	The mouse pointer is positioned over the control.
Pressed	CommonStates	The control is pressed.
Disabled	CommonStates	The control is disabled.
Focused	FocusStates	The control has focus.
Unfocused	FocusStates	The control does not have focus.
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> and the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> and the control does not have focus.

## Button ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [Button](#) control.

```
<!-- FocusVisual -->

<Style x:Key="ButtonFocusVisual">
  <Setter Property="Control.Template">
    <Setter.Value>
      <ControlTemplate>
        <Border>
          <Rectangle Margin="2"
```

```

        StrokeThickness="1"
        Stroke="#60000000"
        StrokeDashArray="1 2" />
    </Border>
</ControlTemplate>
<Setter.Value>
</Setter>
</Style>

<!-- Button -->
<Style TargetType="Button">
    <Setter Property="SnapsToDevicePixels"
        Value="true" />
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="FocusVisualStyle"
        Value="{StaticResource ButtonFocusVisual}" />
    <Setter Property="MinHeight"
        Value="23" />
    <Setter Property="MinWidth"
        Value="75" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="Button">
                <Border TextBlock.Foreground="{TemplateBinding Foreground}"
                    x:Name="Border"
                    CornerRadius="2"
                    BorderThickness="1">
                    <Border.BorderBrush>
                        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1">
                            <LinearGradientBrush.GradientStops>
                                <GradientStopCollection>
                                    <GradientStop Color="{DynamicResource BorderLightColor}"
                                        Offset="0.0" />
                                    <GradientStop Color="{DynamicResource BorderDarkColor}"
                                        Offset="1.0" />
                                </GradientStopCollection>
                            </LinearGradientBrush.GradientStops>
                        </LinearGradientBrush>
                    </Border.BorderBrush>
                    <Border.Background>
                        <LinearGradientBrush EndPoint="0.5,1"
                            StartPoint="0.5,0">
                            <GradientStop Color="{DynamicResource ControlLightColor}"
                                Offset="0" />
                            <GradientStop Color="{DynamicResource ControlMediumColor}"
                                Offset="1" />
                        </LinearGradientBrush>
                    </Border.Background>
                <VisualStateManager.VisualStateGroups>
                    <VisualStateGroup x:Name="CommonStates">
                        <VisualStateGroup.Transitions>
                            <VisualTransition GeneratedDuration="0:0:0.5" />
                            <VisualTransition GeneratedDuration="0"
                                To="Pressed" />
                        </VisualStateGroup.Transitions>
                    <VisualState x:Name="Normal" />
                    <VisualState x:Name="MouseOver">
                        <Storyboard>
                            <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Panel.Background)."
                                (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                                <Storyboard.TargetName="Border">
                                    <EasingColorKeyFrame KeyTime="0"
                                        Value="{StaticResource ControlMouseOverColor}" />
                                </ColorAnimationUsingKeyFrames>
                            </Storyboard>
                        </VisualState>
                    </VisualStateGroup>
                </VisualStateManager.VisualStateGroups>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

```

<VisualState x:Name="Pressed">
    <Storyboard>
        <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Panel.Background).(
            GradientBrush.GradientStops)[1].(GradientStop.Color)"
            Storyboard.TargetName="Border">
            <EasingColorKeyFrame KeyTime="0"
                Value="{StaticResource ControlPressedColor}" />
        </ColorAnimationUsingKeyFrames>
        <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Border.BorderBrush).(
            GradientBrush.GradientStops)[0].(GradientStop.Color)"
            Storyboard.TargetName="Border">
            <EasingColorKeyFrame KeyTime="0"
                Value="{StaticResource PressedBorderDarkColor}" />
        </ColorAnimationUsingKeyFrames>
        <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Border.BorderBrush).(
            GradientBrush.GradientStops)[1].(GradientStop.Color)"
            Storyboard.TargetName="Border">
            <EasingColorKeyFrame KeyTime="0"
                Value="{StaticResource PressedBorderLightColor}" />
        </ColorAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
<VisualState x:Name="Disabled">
    <Storyboard>
        <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Panel.Background).(
            GradientBrush.GradientStops)[1].(GradientStop.Color)"
            Storyboard.TargetName="Border">
            <EasingColorKeyFrame KeyTime="0"
                Value="{StaticResource DisabledControlDarkColor}" />
        </ColorAnimationUsingKeyFrames>
        <ColorAnimationUsingKeyFrames
            Storyboard.TargetProperty="(TextBlock.Foreground).(SolidColorBrush.Color)"
            Storyboard.TargetName="Border">
            <EasingColorKeyFrame KeyTime="0"
                Value="{StaticResource DisabledForegroundColor}" />
        </ColorAnimationUsingKeyFrames>
        <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Border.BorderBrush).(
            GradientBrush.GradientStops)[1].(GradientStop.Color)"
            Storyboard.TargetName="Border">
            <EasingColorKeyFrame KeyTime="0"
                Value="{StaticResource DisabledBorderDarkColor}" />
        </ColorAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<ContentPresenter Margin="2"
    HorizontalAlignment="Center"
    VerticalAlignment="Center"
    RecognizesAccessKey="True" />
</Border>
<ControlTemplate.Triggers>
    <Trigger Property="IsDefault"
        Value="true">

        <Setter TargetName="Border"
            Property="BorderBrush">
            <Setter.Value>
                <LinearGradientBrush StartPoint="0,0"
                    EndPoint="0,1">
                    <GradientBrush.GradientStops>
                        <GradientStopCollection>
                            <GradientStop Color="{DynamicResource DefaultBorderBrushLightBrush}"
                                Offset="0.0" />
                            <GradientStop Color="{DynamicResource DefaultBorderBrushDarkColor}"
                                Offset="1.0" />
                        </GradientStopCollection>
                    </GradientBrush.GradientStops>
                </LinearGradientBrush>
            </Setter.Value>
        </Setter>
    </Trigger>
</ControlTemplate.Triggers>

```

```

        </Setter.Value>
    </Setter>
    <Trigger>
        </ControlTemplate.Triggers>
    </ControlTemplate>
    </Setter.Value>
</Setter>
</Style>

```

The preceding example uses one or more of the following resources.

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FCFFFFFF</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"

```

```
    StartPoint="0,0"
    EndPoint="1,0">
<LinearGradientBrush.GradientStops>
    <GradientStopCollection>
        <GradientStop Color="#000000FF"
            Offset="0" />
        <GradientStop Color="#600000FF"
            Offset="0.4" />
        <GradientStop Color="#600000FF"
            Offset="0.6" />
        <GradientStop Color="#000000FF"
            Offset="1" />
    </GradientStopCollection>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# Calendar Styles and Templates

9 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [Calendar](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## Calendar Parts

The following table lists the named parts for the [Calendar](#) control.

PART	TYPE	DESCRIPTION
PART_CalendarItem	<a href="#">CalendarItem</a>	The currently displayed month or year on the <a href="#">Calendar</a> .
PART_Root	<a href="#">Panel</a>	The panel that contains the <a href="#">CalendarItem</a> .

## Calendar States

The following table lists the visual states for the [Calendar](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## CalendarItem Parts

The following table lists the named parts for the [CalendarItem](#) control.

PART	TYPE	DESCRIPTION
PART_Root	<a href="#">FrameworkElement</a>	The root of the control.
PART_PreviousButton	<a href="#">Button</a>	The button that displays the previous page of the calendar when it is clicked.
PART_NextButton	<a href="#">Button</a>	The button that displays the next page of the calendar when it is clicked.

PART	TYPE	DESCRIPTION
PART_HeaderButton	Button	The button that allows switching between month mode, year mode, and decade mode.
PART_MonthView	Grid	Hosts the content when in month mode.
PART_YearView	Grid	Hosts the content when in year or decade mode.
PART_DisabledVisual	FrameworkElement	The overlay for the disabled state.
DayTitleTemplate	DataTemplate	The <a href="#">DataTemplate</a> that describes the visual structure.

## CalendarItem States

The following table lists the visual states for the [CalendarItem](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal State	CommonStates	The default state.
Disabled State	CommonStates	The state of the calendar when the <a href="#">IsEnabled</a> property is <code>false</code> .
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## CalendarDayButton Parts

The [CalendarDayButton](#) control does not have any named parts.

## CalendarDayButton States

The following table lists the visual states for the [CalendarDayButton](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
Disabled	CommonStates	The <a href="#">CalendarDayButton</a> is disabled.
MouseOver	CommonStates	The mouse pointer is positioned over the <a href="#">CalendarDayButton</a> .
Pressed	CommonStates	The <a href="#">CalendarDayButton</a> is pressed.
Selected	SelectionStates	The button is selected.
Unselected	SelectionStates	The button is not selected.
CalendarButtonFocused	CalendarButtonFocusStates	The button has focus.
CalendarButtonUnfocused	CalendarButtonFocusStates	The button does not have focus.
Focused	FocusStates	The button has focus.
Unfocused	FocusStates	The button does not have focus.
Active	ActiveStates	The button is active.
Inactive	ActiveStates	The button is inactive.
RegularDay	DayStates	The button does not represent <a href="#">DateTime.Today</a> .
Today	DayStates	The button represents <a href="#">DateTime.Today</a> .
NormalDay	BlackoutDayStates	The button represents a day that can be selected.
BlackoutDay	BlackoutDayStates	The button represents a day that cannot be selected.
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## CalendarButton Parts

The [CalendarButton](#) control does not have any named parts.

## CalendarButton States

The following table lists the visual states for the [CalendarButton](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
Disabled	CommonStates	The <a href="#">CalendarButton</a> is disabled.
MouseOver	CommonStates	The mouse pointer is positioned over the <a href="#">CalendarButton</a> .
Pressed	CommonStates	The <a href="#">CalendarButton</a> is pressed.
Selected	SelectionStates	The button is selected.
Unselected	SelectionStates	The button is not selected.
CalendarButtonFocused	CalendarButtonFocusStates	The button has focus.
CalendarButtonUnfocused	CalendarButtonFocusStates	The button does not have focus.
Focused	FocusStates	The button has focus.
Unfocused	FocusStates	The button does not have focus.
Active	ActiveStates	The button is active.
Inactive	ActiveStates	The button is inactive.
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## Calendar ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [Calendar](#) control and associated types.

```
<!--Style for the days of a month.-->
<Style TargetType="CalendarDayButton"
      x:Key="CalendarDayButtonStyle">
    <Setter Property="MinWidth"
           Value="5" />
    <Setter Property="MinHeight"
           Value="5" />
    <Style.Triggers>
      <Trigger Property="IsMouseOver" Value="true">
        <Setter Property="Background" Value="LightBlue" />
      </Trigger>
      <Trigger Property="IsSelected" Value="true">
        <Setter Property="Background" Value="DarkBlue" />
        <Setter Property="Foreground" Value="White" />
      </Trigger>
    </Style.Triggers>
  </Style>
```

```

<Setter Property="FontSize"
       Value="10" />
<Setter Property="HorizontalContentAlignment"
       Value="Center" />
<Setter Property="VerticalContentAlignment"
       Value="Center" />
<Setter Property="Template">
    <Setter.Value>
        <ControlTemplate TargetType="CalendarDayButton">
            <Grid>
                <VisualStateManager.VisualStateGroups>
                    <VisualStateGroup Name="CommonStates">
                        <VisualStateGroup.Transitions>
                            <VisualTransition GeneratedDuration="0:0:0.1" />
                        </VisualStateGroup.Transitions>
                        <VisualState Name="Normal" />
                        <VisualState Name="MouseOver">
                            <Storyboard>
                                <DoubleAnimation Storyboard.TargetName="HighlightBackground"
                                                Storyboard.TargetProperty="Opacity"
                                                To="0.5"
                                                Duration="0" />
                            </Storyboard>
                        </VisualState>
                        <VisualState Name="Pressed">
                            <Storyboard>
                                <DoubleAnimation Storyboard.TargetName="HighlightBackground"
                                                Storyboard.TargetProperty="Opacity"
                                                To="0.5"
                                                Duration="0" />
                            </Storyboard>
                        </VisualState>
                        <VisualState Name="Disabled">
                            <Storyboard>
                                <DoubleAnimation Storyboard.TargetName="HighlightBackground"
                                                Storyboard.TargetProperty="Opacity"
                                                To="0"
                                                Duration="0" />
                                <DoubleAnimation Storyboard.TargetName="NormalText"
                                                Storyboard.TargetProperty="Opacity"
                                                To=".35"
                                                Duration="0" />
                            </Storyboard>
                        </VisualState>
                    </VisualStateGroup>
                    <VisualStateGroup Name="SelectionStates">
                        <VisualStateGroup.Transitions>
                            <VisualTransition GeneratedDuration="0" />
                        </VisualStateGroup.Transitions>
                        <VisualState Name="Unselected" />
                        <VisualState Name="Selected">
                            <Storyboard>
                                <DoubleAnimation Storyboard.TargetName="SelectedBackground"
                                                Storyboard.TargetProperty="Opacity"
                                                To=".75"
                                                Duration="0" />
                            </Storyboard>
                        </VisualState>
                    </VisualStateGroup>
                    <VisualStateGroup Name="CalendarButtonFocusStates">
                        <VisualStateGroup.Transitions>
                            <VisualTransition GeneratedDuration="0" />
                        </VisualStateGroup.Transitions>
                        <VisualState Name="CalendarButtonFocused">
                            <Storyboard>
                                <ObjectAnimationUsingKeyFrames Storyboard.TargetName="DayButtonFocusVisual"
                                                               Storyboard.TargetProperty="Visibility"
                                                               Duration="0">
                                    <DiscreteObjectKeyFrame KeyTime="0">

```

```

        <DiscreteObjectKeyFrame.Value>
            <Visibility>Visible</Visibility>
        </DiscreteObjectKeyFrame.Value>
    </DiscreteObjectKeyFrame>
</ObjectAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState Name="CalendarButtonUnfocused">
    <Storyboard>
        <ObjectAnimationUsingKeyFrames Storyboard.TargetName="DayButtonFocusVisual"
            Storyboard.TargetProperty="Visibility"
            Duration="0">
            <DiscreteObjectKeyFrame KeyTime="0">
                <DiscreteObjectKeyFrame.Value>
                    <Visibility>Collapsed</Visibility>
                </DiscreteObjectKeyFrame.Value>
            </DiscreteObjectKeyFrame>
        </ObjectAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup Name="ActiveStates">
    <VisualStateGroup.Transitions>
        <VisualTransition GeneratedDuration="0" />
    </VisualStateGroup.Transitions>
    <VisualState Name="Active" />
    <VisualState Name="Inactive">
        <Storyboard>
            <ColorAnimation Duration="0"
                Storyboard.TargetName="NormalText"
                Storyboard.TargetProperty="(TextElement.Foreground).(
                    SolidColorBrush.Color)"
                To="#FF777777" />
        </Storyboard>
    </VisualState>
</VisualStateGroup>
<VisualStateGroup Name="DayStates">
    <VisualStateGroup.Transitions>
        <VisualTransition GeneratedDuration="0" />
    </VisualStateGroup.Transitions>
    <VisualState Name="RegularDay" />
    <VisualState Name="Today">
        <Storyboard>
            <DoubleAnimation Storyboard.TargetName="TodayBackground"
                Storyboard.TargetProperty="Opacity"
                To="1"
                Duration="0" />
            <ColorAnimation Duration="0"
                Storyboard.TargetName="NormalText"
                Storyboard.TargetProperty="(TextElement.Foreground).(
                    SolidColorBrush.Color)"
                To="#FFFFFF" />
        </Storyboard>
    </VisualState>
</VisualStateGroup>
<VisualStateGroup Name="BlackoutDayStates">
    <VisualStateGroup.Transitions>
        <VisualTransition GeneratedDuration="0" />
    </VisualStateGroup.Transitions>
    <VisualState Name="NormalDay" />
    <VisualState Name="BlackoutDay">
        <Storyboard>
            <DoubleAnimation Duration="0"
                Storyboard.TargetName="Blackout"
                Storyboard.TargetProperty="Opacity"
                To=".2" />
        </Storyboard>
    </VisualState>
</VisualStateGroup>

```

```

</VisualStateManager.VisualStateGroups>
<Rectangle x:Name="TodayBackground"
           RadiusX="1"
           RadiusY="1"
           Opacity="0">
    <Rectangle.Fill>
        <SolidColorBrush Color="{DynamicResource SelectedBackgroundColor}" />
    </Rectangle.Fill>
</Rectangle>
<Rectangle x:Name="SelectedBackground"
           RadiusX="1"
           RadiusY="1"
           Opacity="0">
    <Rectangle.Fill>
        <SolidColorBrush Color="{DynamicResource SelectedBackgroundColor}" />
    </Rectangle.Fill>
</Rectangle>
<Border Background="{TemplateBinding Background}"
        BorderThickness="{TemplateBinding BorderThickness}"
        BorderBrush="{TemplateBinding BorderBrush}" />
<Rectangle x:Name="HighlightBackground"
           RadiusX="1"
           RadiusY="1"
           Opacity="0">
    <Rectangle.Fill>
        <SolidColorBrush Color="{DynamicResource ControlMouseOverColor}" />
    </Rectangle.Fill>
</Rectangle>
<ContentPresenter x:Name="NormalText"
                  HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"
                  VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
                  Margin="5,1,5,1">
    <TextElement.Foreground>
        <SolidColorBrush Color="#FF333333" />
    </TextElement.Foreground>
</ContentPresenter>
<Path x:Name="Blackout"
      Opacity="0"
      Margin="3"
      HorizontalAlignment="Stretch"
      VerticalAlignment="Stretch"
      RenderTransformOrigin="0.5,0.5"
      Fill="#FF000000"
      Stretch="Fill"
      Data="M8.1772461,11.029181 L10.433105,
            11.029181 L11.700684,12.801641 L12.973633,
            11.029181 L15.191895,11.029181 L12.844727,
            13.999395 L15.21875,17.060919 L12.962891,
            17.060919 L11.673828,15.256231 L10.352539,
            17.060919 L8.1396484,17.060919 L10.519043,
            14.042364 z" />
<Rectangle x:Name="DayButtonFocusVisual"
           Visibility="Collapsed"
           IsHitTestVisible="false"
           RadiusX="1"
           RadiusY="1">
    <Rectangle.Stroke>
        <SolidColorBrush Color="{DynamicResource SelectedBackgroundColor}" />
    </Rectangle.Stroke>
</Rectangle>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<!--Style for the months of a year and years of a decade.-->
<Style TargetType="CalendarButton"
      x:Key="CalendarButtonStyle">

```

```

<Setter Property="MinWidth"
       Value="40" />
<Setter Property="MinHeight"
       Value="42" />
<Setter Property="FontSize"
       Value="10" />
<Setter Property="HorizontalContentAlignment"
       Value="Center" />
<Setter Property="VerticalContentAlignment"
       Value="Center" />
<Setter Property="Template">
    <Setter.Value>
        <ControlTemplate TargetType="CalendarButton">
            <Grid>
                <VisualStateManager.VisualStateGroups>
                    <VisualStateGroup Name="CommonStates">
                        <VisualStateGroup.Transitions>
                            <VisualTransition GeneratedDuration="0:0:0.1" />
                        </VisualStateGroup.Transitions>
                        <VisualState Name="Normal" />
                        <VisualState Name="MouseOver">
                            <Storyboard>
                                <DoubleAnimation Storyboard.TargetName="Background"
                                                Storyboard.TargetProperty="Opacity"
                                                To=".5"
                                                Duration="0" />
                            </Storyboard>
                        </VisualState>
                        <VisualState Name="Pressed">
                            <Storyboard>
                                <DoubleAnimation Storyboard.TargetName="Background"
                                                Storyboard.TargetProperty="Opacity"
                                                To=".5"
                                                Duration="0" />
                            </Storyboard>
                        </VisualState>
                    </VisualStateGroup>
                    <VisualStateGroup Name="SelectionStates">
                        <VisualStateGroup.Transitions>
                            <VisualTransition GeneratedDuration="0" />
                        </VisualStateGroup.Transitions>
                        <VisualState Name="Unselected" />
                        <VisualState Name="Selected">
                            <Storyboard>
                                <DoubleAnimation Storyboard.TargetName="SelectedBackground"
                                                Storyboard.TargetProperty="Opacity"
                                                To=".75"
                                                Duration="0" />
                            </Storyboard>
                        </VisualState>
                    </VisualStateGroup>
                    <VisualStateGroup Name="ActiveStates">
                        <VisualStateGroup.Transitions>
                            <VisualTransition GeneratedDuration="0" />
                        </VisualStateGroup.Transitions>
                        <VisualState Name="Active" />
                        <VisualState Name="Inactive">
                            <Storyboard>
                                <ColorAnimation Duration="0"
                                                Storyboard.TargetName="NormalText"
                                                Storyboard.TargetProperty="(TextElement.Foreground)."
                                                (SolidColorBrush.Color)"
                                                To="#FF777777" />
                            </Storyboard>
                        </VisualState>
                    </VisualStateGroup>
                    <VisualStateGroup Name="CalendarButtonFocusStates">
                        <VisualStateGroup.Transitions>
                            <VisualTransition GeneratedDuration="0" />

```

```

        </VisualStateGroup.Transitions>
        <VisualState Name="CalendarButtonFocused">
            <Storyboard>
                <ObjectAnimationUsingKeyFrames Duration="0"
                    Storyboard.TargetName="CalendarButtonFocusVisual"
                    Storyboard.TargetProperty="Visibility">
                    <DiscreteObjectKeyFrame KeyTime="0">
                        <DiscreteObjectKeyFrame.Value>
                            <Visibility>Visible</Visibility>
                        </DiscreteObjectKeyFrame.Value>
                    </DiscreteObjectKeyFrame>
                </ObjectAnimationUsingKeyFrames>
            </Storyboard>
        </VisualState>
        <VisualState Name="CalendarButtonUnfocused" />
    </VisualStateManager.VisualStateGroups>
    <Rectangle x:Name="SelectedBackground">
        RadiusX="1"
        RadiusY="1"
        Opacity="0">
        <Rectangle.Fill>
            <SolidColorBrush Color="{DynamicResource SelectedBackgroundColor}" />
        </Rectangle.Fill>
    </Rectangle>
    <Rectangle x:Name="Background">
        RadiusX="1"
        RadiusY="1"
        Opacity="0">
        <Rectangle.Fill>
            <SolidColorBrush Color="{DynamicResource SelectedBackgroundColor}" />
        </Rectangle.Fill>
    </Rectangle>
    <ContentPresenter x:Name="NormalText">
        HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"
        VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
        Margin="1,0,1,1">
        <TextElement.Foreground>
            <SolidColorBrush Color="#FF333333" />
        </TextElement.Foreground>
    </ContentPresenter>
    <Rectangle x:Name="CalendarButtonFocusVisual">
        Visibility="Collapsed"
        IsHitTestVisible="false"
        RadiusX="1"
        RadiusY="1">
        <Rectangle.Stroke>
            <SolidColorBrush Color="{DynamicResource SelectedBackgroundColor}" />
        </Rectangle.Stroke>
    </Rectangle>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
<Setter Property="Background">
    <Setter.Value>
        <SolidColorBrush Color="{DynamicResource ControlMediumColor}" />
    </Setter.Value>
</Setter>
</Style>

<!--Button to go to the previous month or year.--&gt;
&lt;ControlTemplate x:Key="PreviousButtonTemplate"
    TargetType="{x:Type Button}"&gt;
    &lt;Grid Cursor="Hand"&gt;
        &lt;VisualStateManager.VisualStateGroups&gt;
            &lt;VisualStateGroup x:Name="CommonStates"&gt;
                &lt;VisualState x:Name="Normal" /&gt;
                &lt;VisualState x:Name="MouseOver"&gt;
</pre>

```

```

<VisualState x:Name="MouseOver" >
    <Storyboard>
        <ColorAnimation Duration="0"
            Storyboard.TargetName="path"
            Storyboard.TargetProperty="(Shape.Fill).(SolidColorBrush.Color)"
            To="{DynamicResource GlyphMouseOver}" />
    </Storyboard>
</VisualState>
<VisualState x:Name="Disabled">
    <Storyboard>
        <DoubleAnimation Duration="0"
            To=".5"
            Storyboard.TargetProperty="(Shape.Fill).(Brush.Opacity)"
            Storyboard.TargetName="path" />
    </Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<!--<Rectangle Fill="Transparent" Opacity="1" Stretch="Fill"/>-->
<Grid Background="Transparent">
    <Path x:Name="path"
        Margin="14,-6,0,0"
        Stretch="Fill"
        HorizontalAlignment="Left"
        Height="10"
        VerticalAlignment="Center"
        Width="6"
        Data="M288.75,232.25 L288.75,240.625 L283,236.625 z">
        <Path.Fill>
            <SolidColorBrush Color="{DynamicResource GlyphColor}" />
        </Path.Fill>
    </Path>
</Grid>
</Grid>
</ControlTemplate>

<!--Button to go to the next month or year.-->
<ControlTemplate x:Key="NextButtonTemplate"
    TargetType="{x:Type Button}">
    <Grid Cursor="Hand">
        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="CommonStates">
                <VisualState x:Name="Normal" />
                <VisualState x:Name="MouseOver">
                    <Storyboard>
                        <ColorAnimation Duration="0"
                            To="{StaticResource GlyphMouseOver}"
                            Storyboard.TargetProperty="(Shape.Fill).(SolidColorBrush.Color)"
                            Storyboard.TargetName="path" />
                    </Storyboard>
                </VisualState>
                <VisualState x:Name="Disabled">
                    <Storyboard>
                        <DoubleAnimation Duration="0"
                            To=".5"
                            Storyboard.TargetProperty="(Shape.Fill).(Brush.Opacity)"
                            Storyboard.TargetName="path" />
                    </Storyboard>
                </VisualState>
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>
        <!--<Rectangle Fill="#11E5EBF1" Opacity="1" Stretch="Fill"/>-->
        <Grid Background="Transparent">
            <Path x:Name="path"
                Data="M282.875,231.875 L282.875,240.375 L288.625,236 z"
                HorizontalAlignment="Right"
                Height="10"
                Margin="0,-6,14,0"
                Stretch="Fill"

```

```

        VerticalAlignment="Center"
        Width="6">
    <Path.Fill>
        <SolidColorBrush Color="{DynamicResource GlyphColor}" />
    </Path.Fill>
</Path>
</Grid>
</Grid>
</ControlTemplate>

<!--Button to go up a level to the year or decade.--&gt;
&lt;ControlTemplate x:Key="HeaderButtonTemplate"&gt;
    &lt;TargetType="{x:Type Button}"&gt;
        &lt;Grid Cursor="Hand"&gt;
            &lt;VisualStateManager.VisualStateGroups&gt;
                &lt;VisualStateGroup x:Name="CommonStates"&gt;
                    &lt;VisualState x:Name="Normal" /&gt;
                    &lt;VisualState x:Name="MouseOver"&gt;
                        &lt;Storyboard&gt;
                            &lt;ColorAnimation Duration="0"
                                To="{DynamicResource GlyphMouseOver}"
                                Storyboard.TargetProperty="(TextElement.Foreground).Color"
                                Storyboard.TargetName="buttonContent" /&gt;
                        &lt;/Storyboard&gt;
                    &lt;/VisualState&gt;
                    &lt;VisualState x:Name="Disabled"&gt;
                        &lt;Storyboard&gt;
                            &lt;DoubleAnimation Duration="0"
                                To=".5"
                                Storyboard.TargetProperty="Opacity"
                                Storyboard.TargetName="buttonContent" /&gt;
                        &lt;/Storyboard&gt;
                    &lt;/VisualState&gt;
                &lt;/VisualStateGroup&gt;
            &lt;/VisualStateManager.VisualStateGroups&gt;
            &lt;ContentPresenter x:Name="buttonContent"
                Margin="1,4,1,9"
                ContentTemplate="{TemplateBinding ContentTemplate}"
                Content="{TemplateBinding Content}"
                TextElement.Foreground="#FF333333"
                HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"
                VerticalAlignment="{TemplateBinding VerticalContentAlignment}" /&gt;
        &lt;/Grid&gt;
    &lt;/ControlTemplate&gt;

&lt;Style x:Key="CalendarItemStyle" TargetType="{x:Type CalendarItem}"&gt;
    &lt;Setter Property="Margin"
        Value="0,3,0,3" /&gt;
    &lt;Setter Property="Template"&gt;
        &lt;Setter.Value&gt;
            &lt;ControlTemplate TargetType="{x:Type CalendarItem}"&gt;
                &lt;ControlTemplate.Resources&gt;
                    &lt;DataTemplate x:Key="{x:Static CalendarItem.DayTitleTemplateResourceKey}"&gt;
                        &lt;TextBlock Foreground="#FF333333"
                            FontWeight="Bold"
                            FontSize="9.5"
                            fontFamily="Verdana"
                            Margin="0,6,0,6"
                            Text="{Binding}"
                            HorizontalAlignment="Center"
                            VerticalAlignment="Center" /&gt;
                    &lt;/DataTemplate&gt;
                &lt;/ControlTemplate.Resources&gt;
                &lt;Grid x:Name="PART_Root"&gt;
                    &lt;Grid.Resources&gt;
                        &lt;SolidColorBrush x:Key="DisabledColor"
                            Color="#A5FFFFFF" /&gt;
                    &lt;/Grid.Resources&gt;
</pre>

```

```

<VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="CommonStates">
        <VisualState x:Name="Normal" />
        <VisualState x:Name="Disabled">
            <Storyboard>
                <DoubleAnimation Duration="0"
                    To="1"
                    Storyboard.TargetProperty="Opacity"
                    Storyboard.TargetName="PART_DisabledVisual" />
            </Storyboard>
        </VisualState>
    </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Border BorderBrush="{TemplateBinding BorderBrush}"
    BorderThickness="{TemplateBinding BorderThickness}"
    Background="{TemplateBinding Background}"
    CornerRadius="1">
    <Border BorderBrush="#FFFFFF"
        BorderThickness="2"
        CornerRadius="1">
        <Grid>
            <Grid.Resources>
            </Grid.Resources>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="Auto" />
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto" />
                <RowDefinition Height="*" />
            </Grid.RowDefinitions>
            <Button x:Name="PART_PreviousButton"
                Template="{StaticResource PreviousButtonTemplate}"
                Focusable="False"
                HorizontalAlignment="Left"
                Grid.Column="0"
                Grid.Row="0"
                Height="20"
                Width="28" />
            <Button x:Name="PART_HeaderButton"
                FontWeight="Bold"
                Focusable="False"
                FontSize="10.5"
                HorizontalAlignment="Center"
                VerticalAlignment="Center"
                Grid.Column="1"
                Grid.Row="0"
                Template="{StaticResource HeaderButtonTemplate}" />
            <Button x:Name="PART_NextButton"
                Focusable="False"
                HorizontalAlignment="Right"
                Grid.Column="2"
                Grid.Row="0"
                Template="{StaticResource NextButtonTemplate}"
                Height="20"
                Width="28" />
            <Grid x:Name="PART_MonthView"
                Visibility="Visible"
                Grid.ColumnSpan="3"
                Grid.Row="1"
                Margin="6,-1,6,6"
                HorizontalAlignment="Center">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto" />
                    <ColumnDefinition Width="Auto" />
                    <ColumnDefinition Width="Auto" />
                    <ColumnDefinition Width="Auto" />
                    <ColumnDefinition Width="Auto" />

```

```

        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="Auto" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>
</Grid>
<Grid x:Name="PART_YearView"
      Visibility="Hidden"
      Grid.ColumnSpan="3"
      Grid.Row="1"
      HorizontalAlignment="Center"
      Margin="6,-3,7,6">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="Auto" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>
</Grid>
</Grid>
</Border>
</Border>
<Rectangle x:Name="PART_DisabledVisual"
           Fill="{StaticResource DisabledColor}"
           Opacity="0"
           RadiusY="2"
           RadiusX="2"
           Stretch="Fill"
           Stroke="{StaticResource DisabledColor}"
           StrokeThickness="1"
           Visibility="Collapsed" />
</Grid>
<ControlTemplate.Triggers>
    <Trigger Property="IsEnabled"
             Value="False">
        <Setter Property="Visibility"
                  TargetName="PART_DisabledVisual"
                  Value="Visible" />
    </Trigger>
    <DataTrigger Binding="{Binding DisplayMode,
                    RelativeSource={RelativeSource FindAncestor,
                    AncestorType={x:Type Calendar}}}"
                 Value="Year">
        <Setter Property="Visibility"
                  TargetName="PART_MonthView"
                  Value="Hidden" />
        <Setter Property="Visibility"
                  TargetName="PART_YearView"
                  Value="Visible" />
    </DataTrigger>
    <DataTrigger Binding="{Binding DisplayMode,
                    RelativeSource={RelativeSource FindAncestor,
                    AncestorType={x:Type Calendar}}}"
                 Value="Decade">
        <Setter Property="Visibility"
                  TargetName="PART_MonthView"
                  Value="Hidden" />
    </DataTrigger>

```

```

        <Setter Property="Visibility"
            TargetName="PART_YearView"
            Value="Visible" />
    </DataTrigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style TargetType="{x:Type Calendar}">
    <Setter Property="CalendarButtonStyle"
        Value="{StaticResource CalendarButtonStyle}" />
    <Setter Property="CalendarDayButtonStyle"
        Value="{StaticResource CalendarDayButtonStyle}" />
    <Setter Property="CalendarItemStyle"
        Value="{StaticResource CalendarItemStyle}" />
    <Setter Property="Foreground"
        Value="#FF333333" />
    <Setter Property="Background">
        <Setter.Value>
            <LinearGradientBrush EndPoint="0.5,1"
                StartPoint="0.5,0">
                <!--The first two gradient stops specifies the background for
                    the calendar's heading and navigation buttons.-->
                <GradientStop Color="{DynamicResource HeaderTopColor}"
                    Offset="0" />
                <GradientStop Color="{DynamicResource ControlMediumColor}"
                    Offset="0.16" />
                <!--The next gradient stop specifies the background for
                    the calendar area.-->
                <GradientStop Color="{DynamicResource ControlLightColor}"
                    Offset="0.16" />
            </LinearGradientBrush>
        </Setter.Value>
    </Setter>
    <Setter Property="BorderBrush">
        <Setter.Value>
            <LinearGradientBrush EndPoint="0,1"
                StartPoint="0,0">
                <GradientStop Color="{DynamicResource BorderLightColor}"
                    Offset="0" />
                <GradientStop Color="{DynamicResource BorderDarkColor}"
                    Offset="1" />
            </LinearGradientBrush>
        </Setter.Value>
    </Setter>
    <Setter Property="BorderThickness"
        Value="1" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Calendar}">
                <StackPanel x:Name="PART_Root"
                    HorizontalAlignment="Center">
                    <CalendarItem x:Name="PART_CalendarItem"
                        BorderBrush="{TemplateBinding BorderBrush}"
                        BorderThickness="{TemplateBinding BorderThickness}"
                        Background="{TemplateBinding Background}"
                        Style="{TemplateBinding CalendarItemStyle}" />
                </StackPanel>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

The preceding example uses one or more of the following resources.

```
<!--Control colors-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FCFFFFFF</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

```
        Offset="1" />
    </GradientStopCollection>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# CheckBox Styles and Templates

2 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [CheckBox](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## CheckBox Parts

The [CheckBox](#) control does not have any named parts.

## CheckBox States

The following table lists the visual states for the [CheckBox](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
MouseOver	CommonStates	The mouse pointer is positioned over the control.
Pressed	CommonStates	The control is pressed.
Disabled	CommonStates	The control is disabled.
Focused	FocusStates	The control has focus.
Unfocused	FocusStates	The control does not have focus.
Checked	CheckStates	<code>IsChecked</code> is <code>true</code> .
Unchecked	CheckStates	<code>IsChecked</code> is <code>false</code> .
Indeterminate	CheckStates	<code>IsThreeState</code> is <code>true</code> , and <code>IsChecked</code> is <code>null</code> .
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## CheckBox ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [CheckBox](#) control.

```

<Style x:Key="{x:Type CheckBox}"
       TargetType="{x:Type CheckBox}">
    <Setter Property="SnapsToDevicePixels"
           Value="true" />
    <Setter Property="OverridesDefaultStyle"
           Value="true" />
    <Setter Property="FocusVisualStyle"
           Value="{DynamicResource CheckBoxFocusVisual}" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type CheckBox}">
                <BulletDecorator Background="Transparent">
                    <BulletDecorator.Bullet>
                        <Border x:Name="Border"
                                Width="13"
                                Height="13"
                                CornerRadius="0"
                                BorderThickness="1">
                            <Border.BorderBrush>
                                <LinearGradientBrush StartPoint="0,0"
                                                       EndPoint="0,1">
                                    <LinearGradientBrush.GradientStops>
                                        <GradientStopCollection>
                                            <GradientStop Color="{DynamicResource BorderLightColor}"
                                                          Offset="0.0" />
                                            <GradientStop Color="{DynamicResource BorderDarkColor}"
                                                          Offset="1.0" />
                                        </GradientStopCollection>
                                    </LinearGradientBrush.GradientStops>
                                </LinearGradientBrush>
                            </Border.BorderBrush>
                            <Border.Background>
                                <LinearGradientBrush StartPoint="0,0"
                                                       EndPoint="0,1">
                                    <LinearGradientBrush.GradientStops>
                                        <GradientStopCollection>
                                            <GradientStop Color="{DynamicResource ControlLightColor}" />
                                            <GradientStop Color="{DynamicResource ControlMediumColor}"
                                                          Offset="1.0" />
                                        </GradientStopCollection>
                                    </LinearGradientBrush.GradientStops>
                                </LinearGradientBrush>
                            </Border.Background>
                        </Border>
                    </BulletDecorator.Bullet>
                </BulletDecorator>
                <Grid>
                    <Path Visibility="Collapsed"
                          Width="7"
                          Height="7"
                          x:Name="CheckMark"
                          SnapsToDevicePixels="False"
                          StrokeThickness="2"
                          Data="M 0 0 L 7 7 M 0 7 L 7 0">
                        <Path.Stroke>
                            <SolidColorBrush Color="{DynamicResource GlyphColor}" />
                        </Path.Stroke>
                    </Path>
                    <Path Visibility="Collapsed"
                          Width="7"
                          Height="7"
                          x:Name="IndeterminateMark"
                          SnapsToDevicePixels="False"
                          StrokeThickness="2"
                          Data="M 0 7 L 7 0">
                        <Path.Stroke>
                            <SolidColorBrush Color="{DynamicResource GlyphColor}" />
                        </Path.Stroke>
                    </Path>
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

```

        </Border>
    </BulletDecorator.Bullet>
<VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="CommonStates">
        <VisualState x:Name="Normal" />
        <VisualState x:Name="MouseOver">
            <Storyboard>
                <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                               Storyboard.TargetProperty="(Panel.Background).(
                    GradientBrush.GradientStops)[1].(GradientStop.Color)">
                    <EasingColorKeyFrame KeyTime="0"
                                         Value="{StaticResource ControlMouseOverColor}" />
                </ColorAnimationUsingKeyFrames>
            </Storyboard>
        </VisualState>
        <VisualState x:Name="Pressed">
            <Storyboard>
                <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                               Storyboard.TargetProperty="(Panel.Background).(
                    GradientBrush.GradientStops)[1].(GradientStop.Color)">
                    <EasingColorKeyFrame KeyTime="0"
                                         Value="{StaticResource ControlPressedColor}" />
                </ColorAnimationUsingKeyFrames>
                <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                               Storyboard.TargetProperty="(Border.BorderBrush).(
                    GradientBrush.GradientStops)[0].(GradientStop.Color)">
                    <EasingColorKeyFrame KeyTime="0"
                                         Value="{StaticResource PressedBorderDarkColor}" />
                </ColorAnimationUsingKeyFrames>
                <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                               Storyboard.TargetProperty="(Border.BorderBrush).(
                    GradientBrush.GradientStops)[1].(GradientStop.Color)">
                    <EasingColorKeyFrame KeyTime="0"
                                         Value="{StaticResource PressedBorderLightColor}" />
                </ColorAnimationUsingKeyFrames>
            </Storyboard>
        </VisualState>
        <VisualState x:Name="Disabled" />
    </VisualStateGroup>
    <VisualStateGroup x:Name="CheckStates">
        <VisualState x:Name="Checked">
            <Storyboard>
                <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)"
                                              Storyboard.TargetName="CheckMark">
                    <DiscreteObjectKeyFrame KeyTime="0"
                                           Value="{x:Static Visibility.Visible}" />
                </ObjectAnimationUsingKeyFrames>
            </Storyboard>
        </VisualState>
        <VisualState x:Name="Unchecked" />
        <VisualState x:Name="Indeterminate">
            <Storyboard>
                <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)"
                                              Storyboard.TargetName="IneterminateMark">
                    <DiscreteObjectKeyFrame KeyTime="0"
                                           Value="{x:Static Visibility.Visible}" />
                </ObjectAnimationUsingKeyFrames>
            </Storyboard>
        </VisualState>
    </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<ContentPresenter Margin="4,0,0,0"
                  VerticalAlignment="Center"
                  HorizontalAlignment="Left"
                  RecognizesAccessKey="True" />

</BulletDecorator>
</ControlTemplate>
</Setter.Value>
</Setter>

```

```
    ,/.....>
```

```
</Style>
```

The preceding example uses one or more of the following resources.

```
<!--Control colors-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

```
        <GradientStopCollection>
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# ComboBox Styles and Templates

4 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [ComboBox](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## ComboBox Parts

The following table lists the named parts for the [ComboBox](#) control.

PART	TYPE	DESCRIPTION
PART_EditableTextBox	TextBox	Contains the text of the <a href="#">ComboBox</a> .
PART_Popup	Popup	The drop-down that contains the items in the combo box.

When you create a [ControlTemplate](#) for a [ComboBox](#), your template might contain an [ItemsPresenter](#) within a [ScrollViewer](#). (The [ItemsPresenter](#) displays each item in the [ComboBox](#); the [ScrollViewer](#) enables scrolling within the control). If the [ItemsPresenter](#) is not the direct child of the [ScrollViewer](#), you must give the [ItemsPresenter](#) the name, `ItemsPresenter`.

## ComboBox States

The following table lists the states for the [ComboBox](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
Disabled	CommonStates	The control is disabled.
MouseOver	CommonStates	The mouse pointer is over the <a href="#">ComboBox</a> control.
Focused	FocusStates	The control has focus.
Unfocused	FocusStates	The control does not have focus.
FocusedDropDown	FocusStates	The drop-down for the <a href="#">ComboBox</a> has focus.
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.
Editable	EditStates	The <a href="#">IsEditable</a> property is <code>true</code> .
Uneditable	EditStates	The <a href="#">IsEditable</a> property is <code>false</code> .

## ComboBoxItem Parts

The [ComboBoxItem](#) control does not have any named parts.

## ComboBoxItem States

The following table lists the states for the [ComboBoxItem](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
Disabled	CommonStates	The control is disabled.
MouseOver	CommonStates	The mouse pointer is over the <a href="#">ComboBoxItem</a> control.
Focused	FocusStates	The control has focus.
Unfocused	FocusStates	The control does not have focus.
Selected	SelectionStates	The item is currently selected.
Unselected	SelectionStates	The item is not selected.
SelectedUnfocused	SelectionStates	The item is selected, but does not have focus.
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## ComboBox ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [ComboBox](#) control and associated types.

```
<ControlTemplate x:Key="ComboBoxToggleButton">
```

```

<ControlTemplate x:Key="CommandBoxToggleButton"
    TargetType="{x:Type ToggleButton}">
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition Width="20" />
    </Grid.ColumnDefinitions>
    <VisualStateManager.VisualStateGroups>
        <VisualStateGroup x:Name="CommonStates">
            <VisualState x:Name="Normal" />
            <VisualState x:Name="MouseOver">
                <Storyboard>
                    <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Panel.Background).(
                        GradientBrush.GradientStops)[1].(GradientStop.Color)"
                        Storyboard.TargetName="Border">
                        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource ControlMouseOverColor}" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>
            <VisualState x:Name="Pressed" />
            <VisualState x:Name="Disabled">
                <Storyboard>
                    <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Panel.Background).(
                        GradientBrush.GradientStops)[1].(GradientStop.Color)"
                        Storyboard.TargetName="Border">
                        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource DisabledControlDarkColor}" />
                    </ColorAnimationUsingKeyFrames>
                    <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Shape.Fill).(
                        SolidColorBrush.Color)"
                        Storyboard.TargetName="Arrow">
                        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource DisabledForegroundColor}" />
                    </ColorAnimationUsingKeyFrames>
                    <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Border.BorderBrush).(
                        GradientBrush.GradientStops)[1].(GradientStop.Color)"
                        Storyboard.TargetName="Border">
                        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource DisabledBorderDarkColor}" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>
        </VisualStateGroup>
        <VisualStateGroup x:Name="CheckStates">
            <VisualState x:Name="Checked">
                <Storyboard>
                    <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Panel.Background).(
                        GradientBrush.GradientStops)[1].(GradientStop.Color)"
                        Storyboard.TargetName="Border">
                        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource ControlPressedColor}" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>
            <VisualState x:Name="Unchecked" />
            <VisualState x:Name="Indeterminate" />
        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>
    <Border x:Name="Border"
        Grid.ColumnSpan="2"
        CornerRadius="2"
        BorderThickness="1">
        <Border.BorderBrush>
            <LinearGradientBrush EndPoint="0,1"
                StartPoint="0,0">
                <GradientStop Color="{DynamicResource BorderLightColor}"
                    Offset="0" />
                <GradientStop Color="{DynamicResource BorderDarkColor}">
                    <!-- -->
                </GradientStop>
            </LinearGradientBrush>
        </Border.BorderBrush>
    </Border>

```

```

        Offset="1" />
    </LinearGradientBrush>
</Border.BorderBrush>
<Border.Background>

    <LinearGradientBrush StartPoint="0,0"
                        EndPoint="0,1">
        <LinearGradientBrush.GradientStops>
            <GradientStopCollection>
                <GradientStop Color="{DynamicResource ControlLightColor}" />
                <GradientStop Color="{DynamicResource ControlMediumColor}"
                               Offset="1.0" />
            </GradientStopCollection>
        </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>

</Border.Background>
</Border>
<Border Grid.Column="0"
        CornerRadius="2,0,0,2"
        Margin="1" >
    <Border.Background>
        <SolidColorBrush Color="{DynamicResource ControlLightColor}"/>
    </Border.Background>
</Border>
<Path x:Name="Arrow"
      Grid.Column="1"
      HorizontalAlignment="Center"
      VerticalAlignment="Center"
      Data="M 0 0 L 4 4 L 8 0 Z" >
    <Path.Fill>
        <SolidColorBrush Color="{DynamicResource GlyphColor}"/>
    </Path.Fill>
</Path>
</Grid>
</ControlTemplate>

<ControlTemplate x:Key="ComboBoxTextBox"
                 TargetType="{x:Type TextBox}">
    <Border x:Name="PART_ContentHost"
           Focusable="False"
           Background="{TemplateBinding Background}" />
</ControlTemplate>

<Style x:Key="{x:Type ComboBox}"
       TargetType="{x:Type ComboBox}">
    <Setter Property="SnapsToDevicePixels"
           Value="true" />
    <Setter Property="OverridesDefaultStyle"
           Value="true" />
    <Setter Property="ScrollViewer.HorizontalScrollBarVisibility"
           Value="Auto" />
    <Setter Property="ScrollViewer.VerticalScrollBarVisibility"
           Value="Auto" />
    <Setter Property="ScrollViewer.CanContentScroll"
           Value="true" />
    <Setter Property="MinWidth"
           Value="120" />
    <Setter Property="MinHeight"
           Value="20" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type ComboBox}">
                <Grid>
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="Normal" />
                            <VisualState x:Name="MouseOver" />
                            <VisualState x:Name="Disabled" />
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>

```

```

<Storyboard>
    <ColorAnimationUsingKeyFrames Storyboard.TargetName="PART_EditableTextBox"
        Storyboard.TargetProperty="(TextElement.Foreground).Color">
        <EasingColorKeyFrame KeyTime="0"
            Value="{StaticResource DisabledForegroundColor}" />
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup x:Name="EditStates">
    <VisualState x:Name="Editable">
        <Storyboard>
            <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)"
                Storyboard.TargetName="PART_EditableTextBox">
                <DiscreteObjectKeyFrame KeyTime="0"
                    Value="{x:Static Visibility.Visible}" />
            </ObjectAnimationUsingKeyFrames>
            <ObjectAnimationUsingKeyFrames
                Storyboard.TargetProperty="(UIElement.Visibility)">
                Storyboard.TargetName="ContentSite">
                <DiscreteObjectKeyFrame KeyTime="0"
                    Value="{x:Static Visibility.Hidden}" />
            </ObjectAnimationUsingKeyFrames>
        </Storyboard>
    </VisualState>
    <VisualState x:Name="Uneditable" />
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<ToggleButton x:Name="ToggleButton"
    Template="{StaticResource ComboBoxToggleButton}"
    Grid.Column="2"
    Focusable="false"
    ClickMode="Press"
    IsChecked="{Binding IsDropDownOpen, Mode=TwoWay,
    RelativeSource={RelativeSource TemplatedParent}}"/>
<ContentPresenter x:Name="ContentSite"
    IsHitTestVisible="False"
    Content="{TemplateBinding SelectionBoxItem}"
    ContentTemplate="{TemplateBinding SelectionBoxItemTemplate}"
    ContentTemplateSelector="{TemplateBinding ItemTemplateSelector}"
    Margin="3,3,23,3"
    VerticalAlignment="Stretch"
    HorizontalAlignment="Left">
</ContentPresenter>
<TextBox x:Name="PART_EditableTextBox"
    Style="{x:Null}"
    Template="{StaticResource ComboBoxTextBox}"
    HorizontalAlignment="Left"
    VerticalAlignment="Bottom"
    Margin="3,3,23,3"
    Focusable="True"
    Background="Transparent"
    Visibility="Hidden"
    IsReadOnly="{TemplateBinding IsReadOnly}" />
<Popup x:Name="Popup"
    Placement="Bottom"
    IsOpen="{TemplateBinding IsDropDownOpen}"
    AllowsTransparency="True"
    Focusable="False"
    PopupAnimation="Slide">
    <Grid x:Name="DropDown"
        SnapsToDevicePixels="True"
        MinWidth="{TemplateBinding ActualWidth}"
        MaxHeight="{TemplateBinding MaxDropDownHeight}">
        <Border x:Name="DropDownBorder"
            BorderThickness="1">
            <Border.BorderBrush>
                <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
            </Border.BorderBrush>
        </Border>
    </Grid>
</Popup>

```

```

        </Border.BorderBrush>
        <Border.Background>
            <SolidColorBrush Color="{DynamicResource ControlLightColor}" />
        </Border.Background>
    </Border>
    <ScrollViewer Margin="4,6,4,6"
                  SnapsToDevicePixels="True">
        <StackPanel IsItemsHost="True"
                    KeyboardNavigation.DirectionalNavigation="Contained" />
    </ScrollViewer>
</Grid>
</Popup>
</Grid>
<ControlTemplate.Triggers>
    <Trigger Property="HasItems"
             Value="false">
        <Setter TargetName="DropDownBorder"
               Property="MinHeight"
               Value="95" />
    </Trigger>
    <Trigger Property="IsGrouping"
             Value="true">
        <Setter Property="ScrollViewer.CanContentScroll"
               Value="false" />
    </Trigger>
    <Trigger SourceName="Popup"
             Property="AllowsTransparency"
             Value="true">
        <Setter TargetName="DropDownBorder"
               Property="CornerRadius"
               Value="4" />
        <Setter TargetName="DropDownBorder"
               Property="Margin"
               Value="0,2,0,0" />
    </Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="{x:Type ComboBoxItem}"
       TargetType="{x:Type ComboBoxItem}">
    <Setter Property="SnapsToDevicePixels"
           Value="true" />
    <Setter Property="OverridesDefaultStyle"
           Value="true" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type ComboBoxItem}">
                <Border x:Name="Border"
                       Padding="2"
                       SnapsToDevicePixels="true"
                       Background="Transparent">
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="SelectionStates">
                            <VisualState x:Name="Unselected" />
                            <VisualState x:Name="Selected">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                                               Storyboard.TargetProperty="(Panel.Background).{SolidColorBrush.Color}">
                                        <EasingColorKeyFrame KeyTime="0"
                                                               Value="{StaticResource SelectedBackgroundColor}" />
                                    </ColorAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="SelectedUnfocused">
                                <Storyboard>

```

```

<ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
    Storyboard.TargetProperty="(Panel.Background).  

    (SolidColorBrush.Color)">
    <EasingColorKeyFrame KeyTime="0"
        Value="{StaticResource SelectedUnfocusedColor}" />
</ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<ContentPresenter />
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

The preceding example uses one or more of the following resources.

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />

```

```
<GradientStop Color="{DynamicResource ControlMediumColor}"
    Offset="0.5" />
<GradientStop Color="{DynamicResource ControlLightColor}"
    Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
<LinearGradientBrush.GradientStops>
    <GradientStopCollection>
        <GradientStop Color="#000000FF"
            Offset="0" />
        <GradientStop Color="#600000FF"
            Offset="0.4" />
        <GradientStop Color="#600000FF"
            Offset="0.6" />
        <GradientStop Color="#000000FF"
            Offset="1" />
    </GradientStopCollection>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# ContextMenu Styles and Templates

2 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [ContextMenu](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## ContextMenu Parts

The [ContextMenu](#) control does not have any named parts.

When you create a [ControlTemplate](#) for a [ContextMenu](#), your template might contain an [ItemsPresenter](#) within a [ScrollViewer](#). (The [ItemsPresenter](#) displays each item in the [ContextMenu](#); the [ScrollViewer](#) enables scrolling within the control). If the [ItemsPresenter](#) is not the direct child of the [ScrollViewer](#), you must give the [ItemsPresenter](#) the name, `ItemsPresenter`.

## ContextMenu States

The following table lists the visual states for the [ContextMenu](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## ContextMenu ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [ContextMenu](#) control.

```

<Style TargetType="{x:Type ContextMenu}">
    <Setter Property="SnapsToDevicePixels"
        Value="True" />
    <Setter Property="OverridesDefaultStyle"
        Value="True" />
    <Setter Property="Grid.IsSharedSizeScope"
        Value="true" />
    <Setter Property="HasDropShadow"
        Value="True" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type ContextMenu}">
                <Border x:Name="Border"
                    Background="{StaticResource MenuPopupBrush}"
                    BorderThickness="1">
                    <Border.BorderBrush>
                        <SolidColorBrush Color="{StaticResource BorderMediumColor}" />
                    </Border.BorderBrush>
                    <StackPanel IsItemsHost="True"
                        KeyboardNavigation.Directionality="Cycle" />
                </Border>
                <ControlTemplate.Triggers>
                    <Trigger Property="HasDropShadow"
                        Value="true">
                        <Setter TargetName="Border"
                            Property="Padding"
                            Value="0,3,0,3" />
                        <Setter TargetName="Border"
                            Property="CornerRadius"
                            Value="4" />
                    </Trigger>
                </ControlTemplate.Triggers>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

The [ControlTemplate](#) uses the following resources.

```

<!--Control colors-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

```

```

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# DataGrid Styles and Templates

12 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [DataGrid](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## DataGrid Parts

The following table lists the named parts for the [DataGrid](#) control.

PART	TYPE	DESCRIPTION
PART_ColumnHeadersPresenter	<a href="#">DataGridColumnHeadersPresenter</a>	The row that contains the column headers.

When you create a [ControlTemplate](#) for a [DataGrid](#), your template might contain an [ItemsPresenter](#) within a [ScrollViewer](#). (The [ItemsPresenter](#) displays each item in the [DataGrid](#); the [ScrollViewer](#) enables scrolling within the control). If the [ItemsPresenter](#) is not the direct child of the [ScrollViewer](#), you must give the [ItemsPresenter](#) the name, `ItemsPresenter`.

The default template for the [DataGrid](#) contains a [ScrollViewer](#) control. For more information about the parts defined by the [ScrollViewer](#), see [ScrollViewer Styles and Templates](#).

## DataGrid States

The following table lists the visual states for the [DataGrid](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
Disabled	CommonStates	The control is disabled.
InvalidFocused	ValidationStates	The control is not valid and has focus.
InvalidUnfocused	ValidationStates	The control is not valid and does not have focus.
Valid	ValidationStates	The control is valid.

## DataGridCell Parts

The [DataGridCell](#) element does not have any named parts.

## DataGridCell States

The following table lists the visual states for the [DataGridCell](#) element.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
MouseOver	CommonStates	The mouse pointer is positioned over the cell.
Focused	FocusStates	The cell has focus.
Unfocused	FocusStates	The cell does not have focus
Current	CurrentStates	The cell is the current cell.
Regular	CurrentStates	The cell is not the current cell.
Display	InteractionStates	The cell is in display mode.
Editing	InteractionStates	The cell is in edit mode.
Selected	SelectionStates	The cell is selected.
Unselected	SelectionStates	The cell is not selected.
InvalidFocused	ValidationStates	The cell is not valid and has focus.
InvalidUnfocused	ValidationStates	The cell is not valid and does not have focus.
Valid	ValidationStates	The cell is valid.

## DataGridRow Parts

The [DataGridRow](#) element does not have any named parts.

## DataGridRow States

The following table lists the visual states for the [DataGridRow](#) element.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
MouseOver	CommonStates	The mouse pointer is positioned over the row.
MouseOver_Editing	CommonStates	The mouse pointer is positioned over the row and the row is in edit mode.
MouseOver_Selected	CommonStates	The mouse pointer is positioned over the row and the row is selected.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
MouseOver_Unfocused_Editing	CommonStates	The mouse pointer is positioned over the row, the row is in edit mode, and does not have focus.
MouseOver_Unfocused_Selected	CommonStates	The mouse pointer is positioned over the row, the row is selected, and does not have focus.
Normal_AlternatingRow	CommonStates	The row is an alternating row.
Normal_Editing	CommonStates	The row is in edit mode.
Normal_Selected	CommonStates	The row is selected.
Unfocused_Editing	CommonStates	The row is in edit mode and does not have focus.
Unfocused_Selected	CommonStates	The row is selected and does not have focus.
InvalidFocused	ValidationStates	The control is not valid and has focus.
InvalidUnfocused	ValidationStates	The control is not valid and does not have focus.
Valid	ValidationStates	The control is valid.

## DataGridRowHeader Parts

The following table lists the named parts for the [DataGridRowHeader](#) element.

PART	TYPE	DESCRIPTION
PART_TopHeaderGripper	Thumb	The element that is used to resize the row header from the top.
PART_BottomHeaderGripper	Thumb	The element that is used to resize the row header from the bottom.

## DataGridRowHeader States

The following table lists the visual states for the [DataGridRowHeader](#) element.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
MouseOver	CommonStates	The mouse pointer is positioned over the row.
MouseOver_CurrentRow	CommonStates	The mouse pointer is positioned over the row and the row is the current row.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
MouseOver_CurrentRow_Selected	CommonStates	The mouse pointer is positioned over the row, and the row is current and selected.
MouseOver_EditingRow	CommonStates	The mouse pointer is positioned over the row and the row is in edit mode.
MouseOver_Selected	CommonStates	The mouse pointer is positioned over the row and the row is selected.
MouseOver_Unfocused_CurrentRow_Selected	CommonStates	The mouse pointer is positioned over the row, the row is current and selected, and does not have focus.
MouseOver_Unfocused_EditingRow	CommonStates	The mouse pointer is positioned over the row, the row is in edit mode, and does not have focus.
MouseOver_Unfocused_Selected	CommonStates	The mouse pointer is positioned over the row, the row is selected, and does not have focus.
Normal_CurrentRow	CommonStates	The row is the current row.
Normal_CurrentRow_Selected	CommonStates	The row is the current row and is selected.
Normal_EditingRow	CommonStates	The row is in edit mode.
Normal_Selected	CommonStates	The row is selected.
Unfocused_CurrentRow_Selected	CommonStates	The row is the current row, is selected, and does not have focus.
Unfocused_EditingRow	CommonStates	The row is in edit mode and does not have focus.
Unfocused_Selected	CommonStates	The row is selected and does not have focus.
InvalidFocused	ValidationStates	The control is not valid and has focus.
InvalidUnfocused	ValidationStates	The control is not valid and does not have focus.
Valid	ValidationStates	The control is valid.

## DataGridColumnHeadersPresenter Parts

The following table lists the named parts for the [DataGridColumnHeadersPresenter](#) element.

PART	TYPE	DESCRIPTION
PART_FillerColumnHeader	DataGridColumnHeader	The placeholder for column headers.

## DataGridColumnHeadersPresenter States

The following table lists the visual states for the [DataGridColumnHeadersPresenter](#) element.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
InvalidFocused	ValidationStates	The cell is not valid and has focus.
InvalidUnfocused	ValidationStates	The cell is not valid and does not have focus.
Valid	ValidationStates	The cell is valid.

## DataGridColumnHeader Parts

The following table lists the named parts for the [DataGridColumnHeader](#) element.

PART	TYPE	DESCRIPTION
PART_LeftHeaderGripper	Thumb	The element that is used to resize the column header from the left.
PART_RightHeaderGripper	Thumb	The element that is used to resize the column header from the right.

## DataGridColumnHeader States

The following table lists the visual states for the [DataGridColumnHeader](#) element.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
MouseOver	CommonStates	The mouse pointer is positioned over the control.
Pressed	CommonStates	The control is pressed.
SortAscending	SortStates	The column is sorted in ascending order.
SortDescending	SortStates	The column is sorted in descending order.
Unsorted	SortStates	The column is not sorted.
InvalidFocused	ValidationStates	The control is not valid and has focus.

VisualState Name	VisualStateGroup Name	Description
InvalidUnfocused	ValidationStates	The control is not valid and does not have focus.
Valid	ValidationStates	The control is valid.

## DataGrid ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [DataGrid](#) control and its associated types.

```

<BooleanToVisibilityConverter x:Key="bool2VisibilityConverter" />

<!--Style and template for the button in the upper left corner of the DataGrid.-->
<Style TargetType="{x:Type Button}"
      x:Key="{ComponentResourceKey ResourceId=DataGridSelectAllButtonStyle,
      TypeInTargetAssembly={x:Type DataGrid}}">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="{x:Type Button}">
        <Grid>
          <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="CommonStates">
              <VisualState x:Name="Normal" />
              <VisualState x:Name="MouseOver">
                <Storyboard>
                  <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                                 Storyboard.TargetProperty="(Shape.Fill)".
                                                 (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                    <EasingColorKeyFrame KeyTime="0"
                                         Value="{StaticResource ControlMouseOverColor}" />
                  </ColorAnimationUsingKeyFrames>
                </Storyboard>
              </VisualState>
              <VisualState x:Name="Pressed">
                <Storyboard>
                  <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                                 Storyboard.TargetProperty="(Shape.Fill)".
                                                 (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                    <EasingColorKeyFrame KeyTime="0"
                                         Value="{StaticResource ControlPressedColor}" />
                  </ColorAnimationUsingKeyFrames>
                </Storyboard>
              </VisualState>
              <VisualState x:Name="Disabled">
                <Storyboard>
                  <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)".
                                                Storyboard.TargetName="Arrow">
                    <DiscreteObjectKeyFrame KeyTime="0"
                                           Value="{x:Static Visibility.Collapsed}" />
                  </ObjectAnimationUsingKeyFrames>
                </Storyboard>
              </VisualState>
            </VisualStateGroup>
          </VisualStateManager.VisualStateGroups>
          <Rectangle x:Name="Border"
                     SnapsToDevicePixels="True">
            <Rectangle.Stroke>
              <LinearGradientBrush EndPoint="0.5,1"
                                 StartPoint="0.5,0">
                <GradientStop Color="{DynamicResource BorderLightColor}"
                              Offset="0" />
                <GradientStop Color="{DynamicResource BorderMediumColor}"
                              Offset="1" />
              </LinearGradientBrush>
            </Rectangle.Stroke>
          </Rectangle>
        </Grid>
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>

```

```

    </LinearGradientBrush>
</Rectangle.Stroke>
<Rectangle.Fill>
    <LinearGradientBrush EndPoint="0.5,1"
        StartPoint="0.5,0">
        <GradientStop Color="{DynamicResource ControlLightColor}"
            Offset="0" />
        <GradientStop Color="{DynamicResource ControlMediumColor}"
            Offset="1" />
    </LinearGradientBrush>
</Rectangle.Fill>
</Rectangle>
<Polygon x:Name="Arrow"
    HorizontalAlignment="Right"
    Margin="8,8,3,3"
    Opacity="0.15"
    Points="0,10 10,10 10,0"
    Stretch="Uniform"
    VerticalAlignment="Bottom">
    <Polygon.Fill>
        <SolidColorBrush Color="{DynamicResource GlyphColor}" />
    </Polygon.Fill>
</Polygon>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<!--Style and template for the DataGrid.--&gt;
&lt;Style TargetType="{x:Type DataGrid}"&gt;
    &lt;Setter Property="Foreground"
        Value="{DynamicResource {x:Static SystemColors.ControlTextBrushKey}}" /&gt;
    &lt;Setter Property="BorderBrush"&gt;
        &lt;Setter.Value&gt;
            &lt;LinearGradientBrush EndPoint="0.5,1"
                StartPoint="0.5,0"&gt;
                &lt;GradientStop Color="{DynamicResource BorderLightColor}"
                    Offset="0" /&gt;
                &lt;GradientStop Color="{DynamicResource BorderDarkColor}"
                    Offset="1" /&gt;
            &lt;/LinearGradientBrush&gt;
        &lt;/Setter.Value&gt;
    &lt;/Setter&gt;
    &lt;Setter Property="BorderThickness"
        Value="1" /&gt;
    &lt;Setter Property="RowDetailsVisibilityMode"
        Value="VisibleWhenSelected" /&gt;
    &lt;Setter Property="ScrollViewer.CanContentScroll"
        Value="true" /&gt;
    &lt;Setter Property="ScrollViewer.PanningMode"
        Value="Both" /&gt;
    &lt;Setter Property="Stylus.IsFlicksEnabled"
        Value="False" /&gt;
    &lt;Setter Property="Template"&gt;
        &lt;Setter.Value&gt;
            &lt;ControlTemplate TargetType="{x:Type DataGrid}"&gt;
                &lt;Border x:Name="border"
                    SnapsToDevicePixels="True"
                    BorderBrush="{TemplateBinding BorderBrush}"
                    BorderThickness="{TemplateBinding BorderThickness}"
                    Padding="{TemplateBinding Padding}"&gt;
                    &lt;Border.Background&gt;
                        &lt;SolidColorBrush Color="{DynamicResource ControlLightColor}" /&gt;
                    &lt;/Border.Background&gt;
                    &lt;VisualStateManager.VisualStateGroups&gt;
                        &lt;VisualStateGroup x:Name="CommonStates"&gt;
                            &lt;VisualState x:Name="Disabled"&gt;
                                &lt;Storyboard&gt;
</pre>

```

```

<ColorAnimationUsingKeyFrames Storyboard.TargetName="border"
    Storyboard.TargetProperty="(Panel.Background).SolidColorBrush.Color)">
    <EasingColorKeyFrame KeyTime="0"
        Value="{DynamicResource ControlLightColor}" />
</ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Normal" />
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<ScrollViewer x:Name="DG_ScrollViewer"
    Focusable="false"
    Background="Black">
<ScrollViewer.Template>
    <ControlTemplate TargetType="{x:Type ScrollViewer}">
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="Auto" />
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto" />
                <RowDefinition Height="*" />
                <RowDefinition Height="Auto" />
            </Grid.RowDefinitions>

            <Button Focusable="false"
                Command="{x:Static DataGrid.SelectAllCommand}"
                Style="{DynamicResource {ComponentResourceKey
                    ResourceId=DataGridSelectAllButtonStyle,
                    TypeInTargetAssembly={x:Type DataGrid}}}"
                Visibility="{Binding HeadersVisibility,
                    ConverterParameter={x:Static DataGridHeadersVisibility.All},
                    Converter={x:Static DataGrid.HeadersVisibilityConverter},
                    RelativeSource={RelativeSource AncestorType={x:Type DataGrid}}}"
                Width="{Binding CellsPanelHorizontalOffset,
                    RelativeSource={RelativeSource AncestorType={x:Type DataGrid}}}" />

            <DataGridColumnHeadersPresenter x:Name="PART_ColumnHeadersPresenter"
                Grid.Column="1"
                Visibility="{Binding HeadersVisibility,
                    ConverterParameter={x:Static DataGridHeadersVisibility.Column},
                    Converter={x:Static DataGrid.HeadersVisibilityConverter},
                    RelativeSource={RelativeSource AncestorType={x:Type DataGrid}}}" />

            <ScrollContentPresenter x:Name="PART_ScrollContentPresenter"
                Grid.ColumnSpan="2"
                Grid.Row="1"
                CanContentScroll="{TemplateBinding CanContentScroll}" />

            <ScrollBar x:Name="PART_VirtualScrollBar"
                Grid.Column="2"
                Grid.Row="1"
                Orientation="Vertical"
                ViewportSize="{TemplateBinding ViewportHeight}"
                Maximum="{TemplateBinding ScrollableHeight}"
                Visibility="{TemplateBinding ComputedVerticalScrollBarVisibility}"
                Value="{Binding VerticalOffset, Mode=OneWay,
                    RelativeSource={RelativeSource TemplatedParent}}"/>

            <Grid Grid.Column="1"
                Grid.Row="2">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="{Binding NonFrozenColumnsViewportHorizontalOffset,
                        RelativeSource={RelativeSource AncestorType={x:Type DataGrid}}}" />
                    <ColumnDefinition Width="*" />
                </Grid.ColumnDefinitions>
            </Grid>
        </Grid>
    </ControlTemplate>
</ScrollViewer.Template>
</ScrollViewer>

```

```

        <ScrollBar x:Name="PART_HorizontalScrollBar"
            Grid.Column="1"
            Orientation="Horizontal"
            ViewportSize="{TemplateBinding ViewportWidth}"
            Maximum="{TemplateBinding ScrollableWidth}"
            Visibility="{TemplateBinding ComputedHorizontalScrollBarVisibility}"
            Value="{Binding HorizontalOffset, Mode=OneWay,
                RelativeSource={RelativeSource TemplatedParent}}"/>
    </Grid>
</Grid>
</ControlTemplate>
</ScrollViewer.Template>
<ItemsPresenter SnapsToDevicePixels="{TemplateBinding SnapsToDevicePixels}" />
</ScrollViewer>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
<Style.Triggers>
<Trigger Property="IsGrouping"
    Value="true">
    <Setter Property="ScrollViewer.CanContentScroll"
        Value="false" />
</Trigger>
</Style.Triggers>
</Style>

<!--Style and template for the DataGridCell.-->
<Style TargetType="{x:Type DataGridCell}">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type DataGridCell}">
                <Border x:Name="border"
                    BorderBrush="Transparent"
                    BorderThickness="1"
                    Background="Transparent"
                    SnapsToDevicePixels="True">
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="FocusStates">
                            <VisualState x:Name="Unfocused" />
                            <VisualState x:Name="Focused" />
                        </VisualStateGroup>
                        <VisualStateGroup x:Name="CurrentStates">
                            <VisualState x:Name="Regular" />
                            <VisualState x:Name="Current">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames Storyboard.TargetName="border"
                                        Storyboard.TargetProperty="(Border.BorderBrush)".
                                        (SolidColorBrush.Color)">
                                        <EasingColorKeyFrame KeyTime="0"
                                            Value="{StaticResource DatagridCurrentCellBorderColor}" />
                                    </ColorAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                    <ContentPresenter SnapsToDevicePixels="{TemplateBinding SnapsToDevicePixels}" />
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

<!--Style and template for the DataGridRow.-->
<Style TargetType="{x:Type DataGridRow}">
    <Setter Property="Background">
        <Setter.Value>

```

```

        <SolidColorBrush Color="{DynamicResource ControlLightColor}" />
    </Setter.Value>
</Setter>
<Setter Property="SnapsToDevicePixels"
        Value="true" />
<Setter Property="Validation.ErrorTemplate"
        Value="{x:Null}" />
<Setter Property="ValidationErrorTemplate">
    <Setter.Value>
        <ControlTemplate>
            <TextBlock Foreground="Red"
                       Margin="2,0,0,0"
                       Text="!"
                       VerticalAlignment="Center" />
        </ControlTemplate>
    </Setter.Value>
</Setter>
<Setter Property="Template">
    <Setter.Value>
        <ControlTemplate TargetType="{x:Type DataGridRow}">
            <Border x:Name="DGR_Border">
                BorderBrush="{TemplateBinding BorderBrush}"
                BorderThickness="{TemplateBinding BorderThickness}"
                SnapsToDevicePixels="True">
                <Border.Background>
                    <LinearGradientBrush EndPoint="0.5,1"
                                         StartPoint="0.5,0">
                        <GradientStop Color="Transparent"
                                      Offset="0" />
                        <GradientStop Color="{DynamicResource ControlLightColor}"
                                      Offset="1" />
                    </LinearGradientBrush>
                </Border.Background>
                <VisualStateManager.VisualStateGroups>
                    <VisualStateGroup x:Name="CommonStates">
                        <VisualState x:Name="Normal" />

                        <!--Provide a different appearance for every other row.-->
                        <VisualState x:Name="Normal_AlternatingRow">
                            <Storyboard>
                                <ColorAnimationUsingKeyFrames Storyboard.TargetName="DGR_Border"
                                                               Storyboard.TargetProperty="(Panel.Background).>
(GradientBrush.GradientStops)[0].(GradientStop.Color)">
                                    <EasingColorKeyFrame KeyTime="0"
                                         Value="{StaticResource ContentAreaColorLight}" />
                                </ColorAnimationUsingKeyFrames>
                                <ColorAnimationUsingKeyFrames Storyboard.TargetName="DGR_Border"
                                                               Storyboard.TargetProperty="(Panel.Background).>
(GradientBrush.GradientStops)[1].(GradientStop.Color)">
                                    <EasingColorKeyFrame KeyTime="0"
                                         Value="{StaticResource ContentAreaColorDark}" />
                                </ColorAnimationUsingKeyFrames>
                            </Storyboard>
                        </VisualState>
                    </VisualStateGroup>
                </VisualStateManager.VisualStateGroups>
            </Border>
        </ControlTemplate>
    </Setter.Value>
</Setter>
<!--In this example, a row in Editing or selected mode has an
identical appearances. In other words, the states
Normal_Selected, Unfocused_Selected, Normal_Editing,
MouseOver_Editing, MouseOver_Unfocused_Editing,
and Unfocused_Editing are identical.-->
<VisualState x:Name="Normal_Selected">
    <Storyboard>
        <ColorAnimationUsingKeyFrames Storyboard.TargetName="DGR_Border"
                                       Storyboard.TargetProperty="(Panel.Background).>
(GradientBrush.GradientStops)[0].(GradientStop.Color)">
            <EasingColorKeyFrame KeyTime="0"
                               Value="{StaticResource ControlMediumColor}" />
        </ColorAnimationUsingKeyFrames>
    </Storyboard>

```

```

<ColorAnimationUsingKeyFrames Storyboard.TargetName="DGR_Border"
                               Storyboard.TargetProperty="(Panel.Background).
                               (GradientBrush.GradientStops)[1].(GradientStop.Color)">
    <EasingColorKeyFrame KeyTime="0"
                         Value="{StaticResource ControlDarkColor}" />
</ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>

<VisualState x:Name="Unfocused_Selected">
<Storyboard>
    <ColorAnimationUsingKeyFrames Storyboard.TargetName="DGR_Border"
                                   Storyboard.TargetProperty="(Panel.Background).
                                   (GradientBrush.GradientStops)[0].(GradientStop.Color)">
        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource ControlMediumColor}" />
    </ColorAnimationUsingKeyFrames>
    <ColorAnimationUsingKeyFrames Storyboard.TargetName="DGR_Border"
                                   Storyboard.TargetProperty="(Panel.Background).
                                   (GradientBrush.GradientStops)[1].(GradientStop.Color)">
        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource ControlDarkColor}" />
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>

<VisualState x:Name="Normal_Editing">
<Storyboard>
    <ColorAnimationUsingKeyFrames Storyboard.TargetName="DGR_Border"
                                   Storyboard.TargetProperty="(Panel.Background).
                                   (GradientBrush.GradientStops)[0].(GradientStop.Color)">
        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource ControlMediumColor}" />
    </ColorAnimationUsingKeyFrames>
    <ColorAnimationUsingKeyFrames Storyboard.TargetName="DGR_Border"
                                   Storyboard.TargetProperty="(Panel.Background).
                                   (GradientBrush.GradientStops)[1].(GradientStop.Color)">
        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource ControlDarkColor}" />
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>

<VisualState x:Name="MouseOver_Editing">
<Storyboard>
    <ColorAnimationUsingKeyFrames Storyboard.TargetName="DGR_Border"
                                   Storyboard.TargetProperty="(Panel.Background).
                                   (GradientBrush.GradientStops)[0].(GradientStop.Color)">
        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource ControlMediumColor}" />
    </ColorAnimationUsingKeyFrames>
    <ColorAnimationUsingKeyFrames Storyboard.TargetName="DGR_Border"
                                   Storyboard.TargetProperty="(Panel.Background).
                                   (GradientBrush.GradientStops)[1].(GradientStop.Color)">
        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource ControlDarkColor}" />
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>

<VisualState x:Name="MouseOver_Unfocused_Editing">
<Storyboard>
    <ColorAnimationUsingKeyFrames Storyboard.TargetName="DGR_Border"
                                   Storyboard.TargetProperty="(Panel.Background).
                                   (GradientBrush.GradientStops)[0].(GradientStop.Color)">
        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource ControlMediumColor}" />
    </ColorAnimationUsingKeyFrames>
    <ColorAnimationUsingKeyFrames Storyboard.TargetName="DGR_Border"
                                   Storyboard.TargetProperty="(Panel.Background).
                                   (GradientBrush.GradientStops)[1].(GradientStop.Color)">

```

```

        Storyboard.TargetProperty="(Panel.Background).
    (GradientBrush.GradientStops)[1].(GradientStop.Color)">
    <EasingColorKeyFrame KeyTime="0"
        Value="{StaticResource ControlDarkColor}" />
</ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>

<VisualState x:Name="Unfocused_Editing">
<Storyboard>
<ColorAnimationUsingKeyFrames Storyboard.TargetName="DGR_Border"
    Storyboard.TargetProperty="(Panel.Background).
    (GradientBrush.GradientStops)[0].(GradientStop.Color)">
    <EasingColorKeyFrame KeyTime="0"
        Value="{StaticResource ControlMediumColor}" />
</ColorAnimationUsingKeyFrames>
<ColorAnimationUsingKeyFrames Storyboard.TargetName="DGR_Border"
    Storyboard.TargetProperty="(Panel.Background).
    (GradientBrush.GradientStops)[1].(GradientStop.Color)">
    <EasingColorKeyFrame KeyTime="0"
        Value="{StaticResource ControlDarkColor}" />
</ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>

<VisualState x:Name="MouseOver">
<Storyboard>
<ColorAnimationUsingKeyFrames Storyboard.TargetName="DGR_Border"
    Storyboard.TargetProperty="(Panel.Background).
    (GradientBrush.GradientStops)[0].(GradientStop.Color)">
    <EasingColorKeyFrame KeyTime="0"
        Value="{StaticResource ControlMediumColor}" />
</ColorAnimationUsingKeyFrames>
<ColorAnimationUsingKeyFrames Storyboard.TargetName="DGR_Border"
    Storyboard.TargetProperty="(Panel.Background).
    (GradientBrush.GradientStops)[1].(GradientStop.Color)">
    <EasingColorKeyFrame KeyTime="0"
        Value="{StaticResource ControlMouseOverColor}" />
</ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>

<!--In this example, the appearance of a selected row
that has the mouse over it is the same regardless of
whether the row is selected. In other words, the states
MouseOver_Editing and MouseOver_Unfocused_Editing are identical.-->
<VisualState x:Name="MouseOver_Selected">
<Storyboard>
<ColorAnimationUsingKeyFrames Storyboard.TargetName="DGR_Border"
    Storyboard.TargetProperty="(Panel.Background).
    (GradientBrush.GradientStops)[0].(GradientStop.Color)">
    <EasingColorKeyFrame KeyTime="0"
        Value="{StaticResource ControlMouseOverColor}" />
</ColorAnimationUsingKeyFrames>
<ColorAnimationUsingKeyFrames Storyboard.TargetName="DGR_Border"
    Storyboard.TargetProperty="(Panel.Background).
    (GradientBrush.GradientStops)[1].(GradientStop.Color)">
    <EasingColorKeyFrame KeyTime="0"
        Value="{StaticResource ControlMouseOverColor}" />
</ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>

<VisualState x:Name="MouseOver_Unfocused_Selected">
<Storyboard>
<ColorAnimationUsingKeyFrames Storyboard.TargetName="DGR_Border"
    Storyboard.TargetProperty="(Panel.Background).
    (GradientBrush.GradientStops)[0].(GradientStop.Color)">
    <EasingColorKeyFrame KeyTime="0"

```

```

        <EasingColorKeyFrame KeyTime="0"
            Value="{StaticResource ControlMouseOverColor}" />
        </ColorAnimationUsingKeyFrames>
        <ColorAnimationUsingKeyFrames Storyboard.TargetName="DGR_Border"
            Storyboard.TargetProperty="(Panel.Background).(
                GradientBrush.GradientStops)[1].(GradientStop.Color)">
            <EasingColorKeyFrame KeyTime="0"
                Value="{StaticResource ControlMouseOverColor}" />
        </ColorAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>

<SelectiveScrollingGrid>
    <SelectiveScrollingGrid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="*" />
    </SelectiveScrollingGrid.ColumnDefinitions>
    <SelectiveScrollingGrid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="Auto" />
    </SelectiveScrollingGrid.RowDefinitions>
    <DataGridCellsPresenter Grid.Column="1"
        ItemsPanel="{TemplateBinding ItemsPanel}"
        SnapsToDevicePixels="{TemplateBinding SnapsToDevicePixels}" />
    <DataGridDetailsPresenter Grid.Column="1"
        Grid.Row="1"
        Visibility="{TemplateBinding DetailsVisibility}"
        SelectiveScrollingGrid.SelectiveScrollingOrientation=
        "{Binding AreRowDetailsFrozen,
        ConverterParameter={x:Static SelectiveScrollingOrientation.Vertical},
        Converter={x:Static DataGrid.RowDetailsScrollingConverter},
        RelativeSource={RelativeSource AncestorType={x:Type DataGrid}}}" />
    <DataGridRowHeader Grid.RowSpan="2"
        SelectiveScrollingGrid.SelectiveScrollingOrientation="Vertical"
        Visibility="{Binding HeadersVisibility,
        ConverterParameter={x:Static DataGridHeadersVisibility.Row},
        Converter={x:Static DataGrid.HeadersVisibilityConverter},
        RelativeSource={RelativeSource AncestorType={x:Type DataGrid}}}" />
    </SelectiveScrollingGrid>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<!--Style and template for the resize control on the DataGridRowHeader.-->
<Style x:Key="RowHeaderGripperStyle"
    TargetType="{x:Type Thumb}">
    <Setter Property="Height"
        Value="8" />
    <Setter Property="Background"
        Value="Transparent" />
    <Setter Property="Cursor"
        Value="SizeNS" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Thumb}">
                <Border Background="{TemplateBinding Background}"
                    Padding="{TemplateBinding Padding}" />
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

<!--Style and template for the DataGridRowHeader.-->
<Style TargetType="{x:Type DataGridRowHeader}">
    <Setter Property="Template">

```

```

<Setter.value>
<ControlTemplate TargetType="{x:Type DataGridRowHeader}">
<Grid>
    <VisualStateManager.VisualStateGroups>
        <!--This example does not specify an appearance for every
            state. You can add storyboard to the states that are listed
            to change the appearance of the DataGridRowHeader when it is
            in a specific state.-->
        <VisualStateGroup x:Name="CommonStates">
            <VisualState x:Name="Normal" />
            <VisualState x:Name="Normal_CurrentRow" />
            <VisualState x:Name="Unfocused_EditingRow" />
            <VisualState x:Name="Normal_EditingRow" />
            <VisualState x:Name="MouseOver">
                <Storyboard>
                    <ColorAnimationUsingKeyFrames Storyboard.TargetName="rowHeaderBorder"
                        Storyboard.TargetProperty="(Panel.Background).<br>
                            (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource ControlMouseOverColor}" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>
            <VisualState x:Name="MouseOver_CurrentRow" />
            <VisualState x:Name="MouseOver_Unfocused_EditingRow" />
            <VisualState x:Name="MouseOver_EditingRow" />
            <VisualState x:Name="MouseOver_Unfocused_Selected" />
            <VisualState x:Name="MouseOver_Selected" />
            <VisualState x:Name="MouseOver_Unfocused_CurrentRow_Selected" />
            <VisualState x:Name="MouseOver_CurrentRow_Selected" />
            <VisualState x:Name="Unfocused_Selected" />
            <VisualState x:Name="Unfocused_CurrentRow_Selected" />
            <VisualState x:Name="Normal_CurrentRow_Selected" />
            <VisualState x:Name="Normal_Selected" />
        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>
    <Border x:Name="rowHeaderBorder">
        <Width>10</Width>
        <BorderThickness>1</BorderThickness>
        <Border.BorderBrush>
            <LinearGradientBrush EndPoint="0.5,1"
                StartPoint="0.5,0">
                <GradientStop Color="{DynamicResource BorderLightColor}"
                    Offset="0" />
                <GradientStop Color="{DynamicResource BorderDarkColor}"
                    Offset="1" />
            </LinearGradientBrush>
        </Border.BorderBrush>
        <Border.Background>
            <LinearGradientBrush EndPoint="0.5,1"
                StartPoint="0.5,0">
                <GradientStop Color="{DynamicResource ControlLightColor}"
                    Offset="0" />
                <GradientStop Color="{DynamicResource ControlMediumColor}"
                    Offset="1" />
            </LinearGradientBrush>
        </Border.Background>
        <StackPanel Orientation="Horizontal">
            <ContentPresenter VerticalAlignment="Center"
                SnapsToDevicePixels="{TemplateBinding SnapsToDevicePixels}" />
            <Control SnapsToDevicePixels="false"
                Template="{Binding ValidationTemplate,
                    RelativeSource={RelativeSource AncestorType={x:Type DataGridRow}}}"
                    Visibility="{Binding (Validation.HasError),
                    Converter={StaticResource bool2VisibilityConverter},
                    RelativeSource={RelativeSource AncestorType={x:Type DataGridRow}}}" />
        </StackPanel>
    </Border>

```

```

        <Thumb x:Name="PART_TopHeaderGripper"
            Style="{StaticResource RowHeaderGripperStyle}"
            VerticalAlignment="Top" />
        <Thumb x:Name="PART_BottomHeaderGripper"
            Style="{StaticResource RowHeaderGripperStyle}"
            VerticalAlignment="Bottom" />
    </Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<!--Style and template for the resize control on the DataGridColumnHeader.-->
<Style x:Key="ColumnHeaderGripperStyle"
    TargetType="{x:Type Thumb}">
    <Setter Property="Width"
        Value="8" />
    <Setter Property="Background"
        Value="Transparent" />
    <Setter Property="Cursor"
        Value="SizeWE" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Thumb}">
                <Border Background="{TemplateBinding Background}"
                    Padding="{TemplateBinding Padding}" />
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

<!--Style and template for the DataGridColumnHeader.-->
<Style TargetType="{x:Type DataGridColumnHeader}">
    <Setter Property="VerticalContentAlignment"
        Value="Center" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type DataGridColumnHeader}">
                <Grid>
                    <Border x:Name="columnHeaderBorder"
                        BorderThickness="1"
                        Padding="3,0,3,0">
                        <Border.BorderBrush>
                            <LinearGradientBrush EndPoint="0.5,1"
                                StartPoint="0.5,0">
                                <GradientStop Color="{DynamicResource BorderLightColor}"
                                    Offset="0" />
                                <GradientStop Color="{DynamicResource BorderDarkColor}"
                                    Offset="1" />
                            </LinearGradientBrush>
                        </Border.BorderBrush>
                        <Border.Background>
                            <LinearGradientBrush EndPoint="0.5,1"
                                StartPoint="0.5,0">
                                <GradientStop Color="{DynamicResource ControlLightColor}"
                                    Offset="0" />
                                <GradientStop Color="{DynamicResource ControlMediumColor}"
                                    Offset="1" />
                            </LinearGradientBrush>
                        </Border.Background>
                        <ContentPresenter HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"
                            SnapsToDevicePixels="{TemplateBinding SnapsToDevicePixels}"
                            VerticalAlignment="{TemplateBinding VerticalContentAlignment}" />
                    </Border>
                    <Thumb x:Name="PART_LeftHeaderGripper"
                        HorizontalAlignment="Left"
                        Style="{StaticResource ColumnHeaderGripperStyle}" />
                    <Thumb x:Name="PART_RightHeaderGripper"

```

```

        HorizontalAlignment="Right"
        Style="{StaticResource ColumnHeaderGripperStyle}" />
    </Grid>
</ControlTemplate>
<Setter.Value>
</Setter>
<Setter Property="Background">
<Setter.Value>
    <LinearGradientBrush EndPoint="0.5,1"
        StartPoint="0.5,0">
        <GradientStop Color="{DynamicResource ControlLightColor}"
            Offset="0" />
        <GradientStop Color="{DynamicResource ControlMediumColor}"
            Offset="1" />
    </LinearGradientBrush>
</Setter.Value>
</Setter>
</Style>

<!--Style and template for the DataGridColumnHeadersPresenter.-->
<Style TargetType="{x:Type DataGridColumnHeadersPresenter}">
<Setter Property="Template">
<Setter.Value>
    <ControlTemplate TargetType="{x:Type DataGridColumnHeadersPresenter}">
        <Grid>
            <DataGridColumnHeader x:Name="PART_FillerColumnHeader"
                IsHitTestVisible="False" />
            <ItemsPresenter />
        </Grid>
    </ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

The preceding example uses one or more of the following resources.

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

```

```

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# DatePicker Styles and Templates

5 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [DatePicker](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## DatePicker Parts

The following table lists the named parts for the [DatePicker](#) control.

PART	TYPE	DESCRIPTION
PART_Root	Grid	The root of the control.
PART_Button	Button	The button that opens and closes the <a href="#">Calendar</a> .
PART_TextBox	DatePickerTextBox	The text box that allows you to input a date.
PART_Popup	Popup	The popup for the <a href="#">DatePicker</a> control.

## DatePicker States

The following table lists the visual states for the [DatePicker](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
Disabled	CommonStates	The <a href="#">DatePicker</a> is disabled.
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## DatePickerTextBox Parts

The following table lists the named parts for the [DatePickerTextBox](#) control.

PART	TYPE	DESCRIPTION
PART_Watermark	ContentControl	The element that contains the initial text in the <a href="#">DatePicker</a> .
PART_ContentElement	FrameworkElement	A visual element that can contain a <a href="#">FrameworkElement</a> . The text of the <a href="#">TextBox</a> is displayed in this element.

## DatePickerTextBox States

The following table lists the visual states for the [DatePickerTextBox](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
Disabled	CommonStates	The <a href="#">DatePickerTextBox</a> is disabled.
MouseOver	CommonStates	The mouse pointer is positioned over the <a href="#">DatePickerTextBox</a> .
ReadOnly	CommonStates	The user cannot change the text in the <a href="#">DatePickerTextBox</a> .
Focused	FocusStates	The control has focus.
Unfocused	FocusStates	The control does not have focus.
Watermarked	WatermarkStates	The control displays its initial text. The <a href="#">DatePickerTextBox</a> is in the state when the user has not entered text or selected a date.
Unwatermarked	WatermarkStates	The user has entered text into the <a href="#">DatePickerTextBox</a> or selected a date in the <a href="#">DatePicker</a> .
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> and the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> and the control does not have focus.

## DatePicker ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [DatePicker](#) control.

```
<!--In this example, an implicit style for Calendar is defined elsewhere  
in the application. DatePickerCalendarStyle is based on the implicit
```

```

style so that the DatePicker will use the application's calendar style.-->
<Style x:Key="DatePickerCalendarStyle"
    TargetType="{x:Type Calendar}"
    BasedOn="{StaticResource {x:Type Calendar}}"/>

<!--The template for the button that displays the calendar.-->
<Style x:Key="DropDownButtonStyle"
    TargetType="Button">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Button}">
                <Grid>
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualStateGroup.Transitions>
                                <VisualTransition GeneratedDuration="0" />
                                <VisualTransition GeneratedDuration="0:0:0.1"
                                    To="MouseOver" />
                                <VisualTransition GeneratedDuration="0:0:0.1"
                                    To="Pressed" />
                            </VisualStateGroup.Transitions>
                            <VisualState x:Name="Normal" />
                            <VisualState x:Name="MouseOver">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames BeginTime="0"
                                        Duration="00:00:00.001"
                                        Storyboard.TargetName="BackgroundGradient"
                                        Storyboard.TargetProperty="(Border.Background).(
                                            GradientBrush.GradientStops)[1].(GradientStop.Color)">
                                        <SplineColorKeyFrame KeyTime="0"
                                            Value="#F2FFFFFF" />
                                    </ColorAnimationUsingKeyFrames>
                                    <ColorAnimationUsingKeyFrames BeginTime="0"
                                        Duration="00:00:00.001"
                                        Storyboard.TargetName="BackgroundGradient"
                                        Storyboard.TargetProperty="(Border.Background).(
                                            GradientBrush.GradientStops)[2].(GradientStop.Color)">
                                        <SplineColorKeyFrame KeyTime="0"
                                            Value="#CCFFFFFF" />
                                    </ColorAnimationUsingKeyFrames>
                                    <ColorAnimation Duration="0"
                                        To="#FF448DCA"
                                        Storyboard.TargetProperty="(Border.Background).(
                                            SolidColorBrush.Color)">
                                        Storyboard.TargetName="Background" />
                                    <ColorAnimationUsingKeyFrames BeginTime="0"
                                        Duration="00:00:00.001"
                                        Storyboard.TargetName="BackgroundGradient"
                                        Storyboard.TargetProperty="(Border.Background).(
                                            GradientBrush.GradientStops)[3].(GradientStop.Color)">
                                        <SplineColorKeyFrame KeyTime="0"
                                            Value="#7FFFFFFF" />
                                    </ColorAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="Pressed">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames BeginTime="0"
                                        Duration="00:00:00.001"
                                        Storyboard.TargetName="Background"
                                        Storyboard.TargetProperty="(Border.Background).(
                                            SolidColorBrush.Color)">
                                        <SplineColorKeyFrame KeyTime="0"
                                            Value="#FF448DCA" />
                                    </ColorAnimationUsingKeyFrames>
                                    <DoubleAnimationUsingKeyFrames BeginTime="0"
                                        Duration="00:00:00.001"
                                        Storyboard.TargetProperty="(UIElement.Opacity)"
                                        Storyboard.TargetName="Highlight">

```

```

        <SplineDoubleKeyFrame KeyTime="0"
            Value="1" />
    </DoubleAnimationUsingKeyFrames>
    <ColorAnimationUsingKeyFrames BeginTime="0"
        Duration="00:00:00.001"
        Storyboard.TargetName="BackgroundGradient"
        Storyboard.TargetProperty="(Border.Background).(
            GradientBrush.GradientStops)[0].(GradientStop.Color)">
        <SplineColorKeyFrame KeyTime="0"
            Value="#F4FFFFFF" />
    </ColorAnimationUsingKeyFrames>
    <ColorAnimationUsingKeyFrames BeginTime="0"
        Duration="00:00:00.001"
        Storyboard.TargetName="BackgroundGradient"
        Storyboard.TargetProperty="(Border.Background).(
            GradientBrush.GradientStops)[1].(GradientStop.Color)">
        <SplineColorKeyFrame KeyTime="0"
            Value="#EAFFFFFF" />
    </ColorAnimationUsingKeyFrames>
    <ColorAnimationUsingKeyFrames BeginTime="0"
        Duration="00:00:00.001"
        Storyboard.TargetName="BackgroundGradient"
        Storyboard.TargetProperty="(Border.Background).(
            GradientBrush.GradientStops)[2].(GradientStop.Color)">
        <SplineColorKeyFrame KeyTime="0"
            Value="#C6FFFFFF" />
    </ColorAnimationUsingKeyFrames>
    <ColorAnimationUsingKeyFrames BeginTime="0"
        Duration="00:00:00.001"
        Storyboard.TargetName="BackgroundGradient"
        Storyboard.TargetProperty="(Border.Background).(
            GradientBrush.GradientStops)[3].(GradientStop.Color)">
        <SplineColorKeyFrame KeyTime="0"
            Value="#6BFFFFFF" />
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Disabled" />
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Grid Background="#11FFFFFF"
    FlowDirection="LeftToRight"
    HorizontalAlignment="Center"
    Height="18"
    Margin="0"
    VerticalAlignment="Center"
    Width="19">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="20*" />
        <ColumnDefinition Width="20*" />
        <ColumnDefinition Width="20*" />
        <ColumnDefinition Width="20*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="23*" />
        <RowDefinition Height="19*" />
        <RowDefinition Height="19*" />
        <RowDefinition Height="19*" />
    </Grid.RowDefinitions>
    <Border x:Name="Highlight"
        BorderThickness="1"
        Grid.ColumnSpan="4"
        CornerRadius="0,0,1,1"
        Margin="-1"
        Opacity="1"
        Grid.Row="0"
        Grid.RowSpan="4">
        <Border.BorderBrush>
            <SolidColorBrush Color="{DynamicResource ControlPressedColor}" />

```

```

        </Border.BorderBrush>
    </Border>
    <Border x:Name="Background"
        BorderBrush="#FFFFFF"
        BorderThickness="1"
        Grid.ColumnSpan="4"
        CornerRadius=".5"
        Margin="0,-1,0,0"
        Opacity="1"
        Grid.Row="1"
        Grid.RowSpan="3">
        <Border.Background>
            <SolidColorBrush Color="{DynamicResource ControlDarkColor}" />
        </Border.Background>
    </Border>
    <Border x:Name="BackgroundGradient"
        BorderBrush="#BF000000"
        BorderThickness="1"
        Grid.ColumnSpan="4"
        CornerRadius=".5"
        Margin="0,-1,0,0"
        Opacity="1"
        Grid.Row="1"
        Grid.RowSpan="3">
        <Border.Background>
            <LinearGradientBrush EndPoint=".7,1"
                StartPoint=".7,0">
                <GradientStop Color="#FFFFFF"
                    Offset="0" />
                <GradientStop Color="#F9FFFFFF"
                    Offset="0.375" />
                <GradientStop Color="#E5FFFFFF"
                    Offset="0.625" />
                <GradientStop Color="#C6FFFFFF"
                    Offset="1" />
            </LinearGradientBrush>
        </Border.Background>
    </Border>
    <Rectangle Grid.ColumnSpan="4"
        Grid.RowSpan="1"
        StrokeThickness="1">
        <Rectangle.Fill>
            <LinearGradientBrush EndPoint="0,1"
                StartPoint="0,0">
                <GradientStop Color="{DynamicResource HeaderTopColor}" />
                <GradientStop Color="{DynamicResource ControlMediumColor}"
                    Offset="1" />
            </LinearGradientBrush>
        </Rectangle.Fill>
        <Rectangle.Stroke>
            <LinearGradientBrush EndPoint="0.48,-1"
                StartPoint="0.48,1.25">
                <GradientStop Color="#FF494949" />
                <GradientStop Color="#FF9F9F9F"
                    Offset="1" />
            </LinearGradientBrush>
        </Rectangle.Stroke>
    </Rectangle>
    <Path Fill="#FF2F2F2F"
        Grid.Row="1"
        Grid.Column="0"
        Grid.RowSpan="3"
        Grid.ColumnSpan="4"
        HorizontalAlignment="Center"
        VerticalAlignment="Center"
        RenderTransformOrigin="0.5,0.5"
        Margin="4,3,4,3"
        Stretch="Fill"
        Data="M11 426758 & 4305077 L 11 710023 & 4305077
    
```

```

        Data="L11.7420730,0.45000// L11.7420730,0.45000//"
        L11.749023,16.331387 L10.674805,16.331387
        L10.674805,10.299648 L9.0742188,11.298672
        L9.0742188,10.294277 C9.4788408,10.090176
        9.9094238,9.8090878 10.365967,9.4510155
        C10.82251,9.0929432 11.176106,8.7527733
        11.426758,8.4305077 z M14.65086,8.4305077
        L18.566387,8.4305077 L18.566387,9.3435936
        L15.671368,9.3435936 L15.671368,11.255703
        C15.936341,11.058764 16.27293,10.960293
        16.681133,10.960293 C17.411602,10.960293
        17.969301,11.178717 18.354229,11.615566
        C18.739157,12.052416 18.931622,12.673672
        18.931622,13.479336 C18.931622,15.452317
        18.052553,16.438808 16.294415,16.438808
        C15.560365,16.438808 14.951641,16.234707
        14.468243,15.826504 L14.881817,14.929531
        C15.368796,15.326992 15.837872,15.525723
        16.289043,15.525723 C17.298809,15.525723
        17.803692,14.895514 17.803692,13.635098
        C17.803692,12.460618 17.305971,11.873379
        16.310528,11.873379 C15.83071,11.873379
        15.399232,12.079271 15.016094,12.491055
        L14.65086,12.238613 z" />
    <Ellipse Grid.ColumnSpan="4"
        Fill="#FFFFFF"
        HorizontalAlignment="Center"
        Height="3"
        StrokeThickness="0"
        VerticalAlignment="Center"
        Width="3" />
    <Border x:Name="DisabledVisual"
        BorderBrush="#B2FFFFFF"
        BorderThickness="1"
        Grid.ColumnSpan="4"
        CornerRadius="0,0,.5,.5"
        Opacity="0"
        Grid.Row="0"
        Grid.RowSpan="4" />
    </Grid>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style TargetType="{x:Type DatePicker}">
    <Setter Property="Foreground"
        Value="#FF333333" />
    <Setter Property="IsTodayHighlighted"
        Value="True" />
    <Setter Property="SelectedDateFormat"
        Value="Short" />
    <Setter Property="Padding"
        Value="2" />
    <Setter Property="BorderThickness"
        Value="1" />
    <Setter Property="HorizontalContentAlignment"
        Value="Stretch" />
    <!--Set CalendarStyle to DatePickerCalendarStyle.-->
    <Setter Property="CalendarStyle"
        Value="{DynamicResource DatePickerCalendarStyle}" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type DatePicker}">
                <Border BorderThickness="{TemplateBinding BorderThickness}"
                    Padding="{TemplateBinding Padding}">
                    <Border.BorderBrush>
                        <LinearGradientBrush EndPoint="0.5,1"
                            StartPoint="0,0">

```

```

        StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource BorderLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource BorderDarkColor}"
        Offset="1" />
</LinearGradientBrush>
</Border.BorderBrush>
<Border.Background>
    <LinearGradientBrush EndPoint="0.5,1"
        StartPoint="0.5,0">
        <GradientStop Color="{DynamicResource HeaderTopColor}"
            Offset="0" />
        <GradientStop Color="{DynamicResource ControlMediumColor}"
            Offset="1" />
    </LinearGradientBrush>
</Border.Background>
<VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="CommonStates">
        <VisualState x:Name="Normal" />
        <VisualState x:Name="Disabled">
            <Storyboard>
                <DoubleAnimation Duration="0"
                    To="1"
                    Storyboard.TargetProperty="Opacity"
                    Storyboard.TargetName="PART_DisabledVisual" />
            </Storyboard>
        </VisualState>
    </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Grid x:Name="PART_Root"
    HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"
    VerticalAlignment="{TemplateBinding VerticalContentAlignment}">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="Auto" />
    </Grid.ColumnDefinitions>
    <Button x:Name="PART_Button"
        Grid.Column="1"
        Foreground="{TemplateBinding Foreground}"
        Focusable="False"
        HorizontalAlignment="Left"
        Margin="3,0,3,0"
        Grid.Row="0"
        Style="{StaticResource DropDownButtonStyle}"
        VerticalAlignment="Top" />
    <DatePickerTextBox x:Name="PART_TextBox"
        Grid.Column="0"
        Foreground="{TemplateBinding Foreground}"
        Focusable="{TemplateBinding Focusable}"
        HorizontalContentAlignment="Stretch"
        Grid.Row="0"
        VerticalContentAlignment="Stretch" />
    <Grid x:Name="PART_DisabledVisual"
        Grid.ColumnSpan="2"
        Grid.Column="0"
        IsHitTestVisible="False"
        Opacity="0"
        Grid.Row="0">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="Auto" />
        </Grid.ColumnDefinitions>
        <Rectangle Grid.Column="0"
            Fill="#A5FFFFFF"
            RadiusY="1"
            Grid.Row="0"
            RadiusX="1" />
        <Rectangle Grid.Column="1"
            Fill="#A5FFFFFF"

```

```

        Height="18"
        Margin="3,0,3,0"
        RadiusY="1"
        Grid.Row="0"
        RadiusX="1"
        Width="19" />
    <Popup x:Name="PART_Popup"
        AllowsTransparency="True"
        Placement="Bottom"
        PlacementTarget="{Binding ElementName=PART_TextBox}"
        StaysOpen="False" />
</Grid>
</Grid>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

The preceding example uses one or more of the following resources.

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">

```

```
<GradientStop Color="{DynamicResource ControlLightColor}"
    Offset="0" />
<GradientStop Color="{DynamicResource ControlMediumColor}"
    Offset="0.5" />
<GradientStop Color="{DynamicResource ControlLightColor}"
    Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# DocumentViewer Styles and Templates

2 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [DocumentViewer](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## DocumentViewer Parts

The following table lists the named parts for the [DocumentViewer](#) control.

PART	TYPE	DESCRIPTION
PART_ContentHost	ScrollViewer	The content and scrolling area.
PART_FindToolBarHost	ContentControl	The search box, at the bottom by default.

## DocumentViewer States

The following table lists the visual states for the [DocumentViewer](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## DocumentViewer ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [DocumentViewer](#) control.

```
<Style x:Key="{x:Type DocumentViewer}">
    TargetType="{x:Type DocumentViewer}"
    <Setter Property="Foreground"
        Value="{DynamicResource {x:Static SystemColors.WindowTextBrushKey}}"/>
    <Setter Property="Background"
        Value="{DynamicResource {x:Static SystemColors.ControlBrushKey}}"/>
    <Setter Property="FocusVisualStyle"
        Value="{x:Null}"/>
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type DocumentViewer}">
                <Border BorderThickness="{TemplateBinding BorderThickness}"
                    BorderBrush="{TemplateBinding BorderBrush}"
                    Focusable="False">
```

```

<Grid KeyboardNavigation.TabNavigation="Local">
    <Grid.Background>
        <SolidColorBrush Color="{DynamicResource ControlLightColor}" />
    </Grid.Background>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="*" />
        <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>
    <ToolBar ToolBarTray.IsLocked="True"
        KeyboardNavigation.TabNavigation="Continue">
        <Button Command="ApplicationCommands.Print"
            CommandTarget="{Binding RelativeSource={RelativeSource TemplatedParent}}"
            Content="Print" />
        <Button Command="ApplicationCommands.Copy"
            CommandTarget="{Binding RelativeSource={RelativeSource TemplatedParent}}"
            Content="Copy" />
        <Separator />
        <Button Command="NavigationCommands.IncreaseZoom"
            CommandTarget="{Binding RelativeSource={RelativeSource TemplatedParent}}"
            Content="Zoom In" />
        <Button Command="NavigationCommands.DecreaseZoom"
            CommandTarget="{Binding RelativeSource={RelativeSource TemplatedParent}}"
            Content="Zoom Out" />
        <Separator />
        <Button Command="NavigationCommands.Zoom"
            CommandTarget="{Binding RelativeSource={RelativeSource TemplatedParent}}"
            CommandParameter="100.0"
            Content="Actual Size" />
        <Button Command="DocumentViewer.FitToWidthCommand"
            CommandTarget="{Binding RelativeSource={RelativeSource TemplatedParent}}"
            Content="Fit to Width" />
        <Button Command="DocumentViewer.FitToMaxPagesAcrossCommand"
            CommandTarget="{Binding RelativeSource={RelativeSource TemplatedParent}}"
            CommandParameter="1"
            Content="Whole Page" />
        <Button Command="DocumentViewer.FitToMaxPagesAcrossCommand"
            CommandTarget="{Binding RelativeSource={RelativeSource TemplatedParent}}"
            CommandParameter="2"
            Content="Two Pages" />
    </ToolBar>
</Grid>
<ScrollViewer Grid.Row="1"
    CanContentScroll="true"
    HorizontalScrollBarVisibility="Auto"
    x:Name="PART_ContentHost"
    IsTabStop="true">
    <ScrollViewer.Background>
        <LinearGradientBrush EndPoint="0.5,1"
            StartPoint="0.5,0">
            <GradientStop Color="{DynamicResource ControlLightColor}"
                Offset="0" />
            <GradientStop Color="{DynamicResource ControlMediumColor}"
                Offset="1" />
        </LinearGradientBrush>
    </ScrollViewer.Background>
</ScrollViewer>
<ContentControl Grid.Row="2"
    x:Name="PART_FindToolBarHost"/>
</Grid>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

The preceding example uses one or more of the following resources.

```
<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FCFFFFFF</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

```
        Offset="1" />
    </GradientStopCollection>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# Expander Styles and Templates

3 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [Expander](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## Expander Parts

The [Expander](#) control does not have any named parts.

## Expander States

The following table lists the visual states for the [Expander](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
MouseOver	CommonStates	The mouse pointer is positioned over the control.
Disabled	CommonStates	The control is disabled.
Focused	FocusStates	The control has focus.
Unfocused	FocusStates	The control does not have focus.
Expanded	ExpansionStates	The control is expanded.
Collapsed	ExpansionStates	The control is not expanded.
ExpandDown	ExpandDirectionStates	The control expands down.
ExpandUp	ExpandDirectionStates	The control expands up.
ExpandLeft	ExpandDirectionStates	The control expands left.
ExpandRight	ExpandDirectionStates	The control expands right.
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

# Expander ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [Expander](#) control.

```
<ControlTemplate x:Key="ExpanderToggleButton"
    TargetType="{x:Type ToggleButton}">
    <Border x:Name="Border"
        CornerRadius="2,0,0,0"
        BorderThickness="0,0,1,0">
        <Border.Background>
            <LinearGradientBrush EndPoint="0.5,1"
                StartPoint="0.5,0">
                <GradientStop Color="{DynamicResource ControlLightColor}" />
                <GradientStop Color="{DynamicResource ControlMediumColor}"
                    Offset="1" />
            </LinearGradientBrush>
        </Border.Background>
        <Border.BorderBrush>
            <LinearGradientBrush StartPoint="0,0"
                EndPoint="0,1">
                <LinearGradientBrush.GradientStops>
                    <GradientStopCollection>
                        <GradientStop Color="{DynamicResource BorderLightColor}"
                            Offset="0.0" />
                        <GradientStop Color="{DynamicResource BorderDarkColor}"
                            Offset="1.0" />
                    </GradientStopCollection>
                </LinearGradientBrush.GradientStops>
            </LinearGradientBrush>
        </Border.BorderBrush>
        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="CommonStates">
                <VisualState x:Name="Normal" />
                <VisualState x:Name="MouseOver">
                    <Storyboard>
                        <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                            Storyboard.TargetProperty="(Panel.Background)."
                            (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource ControlMouseOverColor}" />
                        </ColorAnimationUsingKeyFrames>
                    </Storyboard>
                </VisualState>
                <VisualState x:Name="Pressed">
                    <Storyboard>
                        <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                            Storyboard.TargetProperty="(Panel.Background)."
                            (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource ControlPressedColor}" />
                        </ColorAnimationUsingKeyFrames>
                    </Storyboard>
                </VisualState>
                <VisualState x:Name="Disabled">
                    <Storyboard>
                        <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                            Storyboard.TargetProperty="(Panel.Background)."
                            (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource DisabledControlDarkColor}" />
                        </ColorAnimationUsingKeyFrames>
                        <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                            Storyboard.TargetProperty="(Border.BorderBrush)."
                            (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource DisabledBorderLightColor}" />
                        </ColorAnimationUsingKeyFrames>
                    </Storyboard>
                </VisualState>
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>
    </Border>
</ControlTemplate>
```

```

        </ColorAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup x:Name="CheckStates">
    <VisualState x:Name="Checked">
        <Storyboard>
            <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)" Storyboard.TargetName="CollapsedArrow">
                <DiscreteObjectKeyFrame KeyTime="0"
                    Value="{x:Static Visibility.Hidden}" />
            </ObjectAnimationUsingKeyFrames>
            <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)" Storyboard.TargetName="ExpandedArrow">
                <DiscreteObjectKeyFrame KeyTime="0"
                    Value="{x:Static Visibility.Visible}" />
            </ObjectAnimationUsingKeyFrames>
        </Storyboard>
    </VisualState>
    <VisualState x:Name="Unchecked" />
    <VisualState x:Name="Indeterminate" />
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Grid>
    <Path x:Name="CollapsedArrow"
        HorizontalAlignment="Center"
        VerticalAlignment="Center"
        Data="M 0 0 L 4 4 L 8 0 Z">
        <Path.Fill>
            <SolidColorBrush Color="{DynamicResource GlyphColor}" />
        </Path.Fill>
    </Path>
    <Path x:Name="ExpandedArrow"
        HorizontalAlignment="Center"
        VerticalAlignment="Center"
        Visibility="Collapsed"
        Data="M 0 4 L 4 0 L 8 4 Z">
        <Path.Fill>
            <SolidColorBrush Color="{DynamicResource GlyphColor}" />
        </Path.Fill>
    </Path>
</Grid>
</Border>
</ControlTemplate>

<Style TargetType="{x:Type Expander}">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Expander}">
                <Grid>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto" />
                        <RowDefinition x:Name="ContentRow"
                            Height="0" />
                    </Grid.RowDefinitions>
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="Normal" />
                            <VisualState x:Name="MouseOver" />
                            <VisualState x:Name="Disabled">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                        Storyboard.TargetProperty="(Panel.Background)."
                                        (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                                        <EasingColorKeyFrame KeyTime="0"
                                            Value="{StaticResource DisabledControlDarkColor}" />
                                    </ColorAnimationUsingKeyFrames>
                                    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                        Storyboard.TargetProperty="(Border.BorderBrush)."

```

```

        (GradientBrush.GradientStops)[1].(GradientStop.Color)">
        <EasingColorKeyFrame KeyTime="0"
            Value="{StaticResource DisabledBorderLightColor}" />
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Border x:Name="Border"
    Grid.Row="0"
    BorderThickness="1"
    CornerRadius="2,2,0,0">
    <Border.BorderBrush>
        <LinearGradientBrush EndPoint="0,1"
            StartPoint="0,0">
            <GradientStop Color="{DynamicResource BorderLightColor}"
                Offset="0" />
            <GradientStop Color="{DynamicResource BorderDarkColor}"
                Offset="1" />
        </LinearGradientBrush>
    </Border.BorderBrush>
    <Border.Background>

        <LinearGradientBrush StartPoint="0,0"
            EndPoint="0,1">
            <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                    <GradientStop Color="{DynamicResource ControlLightColor}"
                        Offset="0.0" />
                    <GradientStop Color="{DynamicResource ControlMediumColor}"
                        Offset="1.0" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Border.Background>
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="20" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <ToggleButton OverridesDefaultStyle="True"
            Template="{StaticResource ExpanderToggleButton}"
            IsChecked="{Binding IsExpanded, Mode=TwoWay,
            RelativeSource={RelativeSource TemplatedParent}}">
            <ToggleButton.Background>
                <LinearGradientBrush EndPoint="0.5,1"
                    StartPoint="0.5,0">
                    <GradientStop Color="{DynamicResource ControlLightColor}"
                        Offset="0" />
                    <GradientStop Color="{DynamicResource ControlMediumColor}"
                        Offset="1" />
                </LinearGradientBrush>
            </ToggleButton.Background>
        </ToggleButton>
        <ContentPresenter Grid.Column="1"
            Margin="4"
            ContentSource="Header"
            RecognizesAccessKey="True" />
    </Grid>
</Border>
<Border x:Name="Content"
    Grid.Row="1"
    BorderThickness="1,0,1,1"
    CornerRadius="0,0,2,2">
    <Border.BorderBrush>
        <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
    </Border.BorderBrush>

```

```

<Border.Background>
    <SolidColorBrush Color="{DynamicResource ContentAreaColorDark}" />
</Border.Background>
<ContentPresenter Margin="4" />
</Border>
</Grid>
<ControlTemplate.Triggers>
    <Trigger Property="IsExpanded"
        Value="True">
        <Setter TargetName="ContentRow"
            Property="Height"
            Value="{Binding DesiredHeight, ElementName=Content}" />
    </Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

The preceding example uses one or more of the following resources.

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">

```

```
<GradientStop Color="{DynamicResource ControlLightColor}"
    Offset="0" />
<GradientStop Color="{DynamicResource ControlMediumColor}"
    Offset="0.5" />
<GradientStop Color="{DynamicResource ControlLightColor}"
    Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# Frame Styles and Templates

5 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [Frame](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

# Frame Parts

The following table lists the named parts for the [Frame](#) control.

PART	TYPE	DESCRIPTION
PART_FrameCP	ContentPresenter	The content area.

# Frame States

The following table lists the visual states for the [Frame](#) control.

VisualState Name	VisualStateGroup Name	Description
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## Frame ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [Frame](#) control.

```
<!-- Back/Forward Button Style -->

<Style x:Key="FrameButtonStyle"
    TargetType="{x:Type Button}">
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="Command"
        Value="NavigationCommands.BrowseBack" />
    <Setter Property="Focusable"
        Value="false" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Button}">
                <Grid>
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="Normal" />
                            <VisualState x:Name="MouseOver" />
```

```

<Storyboard>
    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Ellipse"
        Storyboard.TargetProperty="(Shape.Fill).GradientBrush.GradientStops)[1].(GradientStop.Color)">
        <EasingColorKeyFrame KeyTime="0"
            Value="{StaticResource ControlMouseOverColor}" />
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Pressed">
    <Storyboard>
        <ColorAnimationUsingKeyFrames Storyboard.TargetName="Ellipse"
            Storyboard.TargetProperty="(Shape.Fill).GradientBrush.GradientStops)[1].(GradientStop.Color)">
            <EasingColorKeyFrame KeyTime="0"
                Value="{StaticResource ControlPressedColor}" />
        </ColorAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
<VisualState x:Name="Disabled">
    <Storyboard>
        <ColorAnimationUsingKeyFrames Storyboard.TargetName="Ellipse"
            Storyboard.TargetProperty="(Shape.Fill).GradientBrush.GradientStops)[1].(GradientStop.Color)">
            <EasingColorKeyFrame KeyTime="0"
                Value="{StaticResource DisabledControlDarkColor}" />
        </ColorAnimationUsingKeyFrames>
        <ColorAnimationUsingKeyFrames Storyboard.TargetName="Arrow"
            Storyboard.TargetProperty="(Shape.Fill).SolidColorBrush.Color)">
            <EasingColorKeyFrame KeyTime="0"
                Value="{StaticResource DisabledForegroundColor}" />
        </ColorAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
</VisualStateManager.VisualStateGroups>
<Ellipse x:Name="Ellipse"
    StrokeThickness="1"
    Width="16"
    Height="16">
    <Ellipse.Stroke>
        <SolidColorBrush Color="{DynamicResource NavButtonFrameColor}" />
    </Ellipse.Stroke>
    <Ellipse.Fill>
        <LinearGradientBrush StartPoint="0,0"
            EndPoint="0,1">
            <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                    <GradientStop Color="{DynamicResource ControlLightColor}" />
                    <GradientStop Color="{DynamicResource ControlMediumColor}"
                        Offset="1.0" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Ellipse.Fill>
</Ellipse>

<Path x:Name="Arrow"
    Margin="0,0,2,0"
    HorizontalAlignment="Center"
    VerticalAlignment="Center"
    Data="M 4 0 L 0 4 L 4 8 Z" >
    <Path.Fill>
        <SolidColorBrush Color="{DynamicResource GlyphColor}"/>
    </Path.Fill>
</Path>
</Grid>
<ControlTemplate.Triggers>

```

```

<Trigger Property="Command"
         Value="{x:Static NavigationCommands.BrowseForward}">
    <Setter TargetName="Arrow"
           Property="Data"
           Value="M 0 0 L 4 4 L 0 8 z" />
    <Setter TargetName="Arrow"
           Property="Margin"
           Value="2,0,0,0" />
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<!-- Frame Menu Style --&gt;

&lt;Style x:Key="FrameMenu"
       TargetType="{x:Type Menu}"&gt;
    &lt;Setter Property="OverridesDefaultStyle"
           Value="true" /&gt;
    &lt;Setter Property="KeyboardNavigation.TabNavigation"
           Value="None" /&gt;
    &lt;Setter Property="IsMainMenu"
           Value="false" /&gt;
    &lt;Setter Property="Template"&gt;
        &lt;Setter.Value&gt;
            &lt;ControlTemplate TargetType="{x:Type Menu}"&gt;
                &lt;DockPanel IsItemsHost="true" /&gt;
            &lt;/ControlTemplate&gt;
        &lt;/Setter.Value&gt;
    &lt;/Setter&gt;
&lt;/Style&gt;

<!-- Frame Menu Header Style --&gt;

&lt;Style x:Key="FrameHeaderMenuItem"
       TargetType="{x:Type MenuItem}"&gt;
    &lt;Setter Property="OverridesDefaultStyle"
           Value="true" /&gt;
    &lt;Setter Property="Template"&gt;
        &lt;Setter.Value&gt;
            &lt;ControlTemplate TargetType="{x:Type MenuItem}"&gt;
                &lt;Grid&gt;
                    &lt;Popup x:Name="PART_Popup"
                           Placement="Bottom"
                           VerticalOffset="2"
                           IsOpen="{TemplateBinding IsSubmenuOpen}"
                           AllowsTransparency="True"
                           Focusable="False"
                           PopupAnimation="Fade"&gt;
                        &lt;Border x:Name="SubMenuBorder"
                               BorderThickness="1"
                               Background="{DynamicResource MenuPopupBrush}"&gt;
                            &lt;Border.BorderBrush&gt;
                                &lt;SolidColorBrush Color="{StaticResource BorderMediumColor}" /&gt;
                            &lt;/Border.BorderBrush&gt;
                            &lt;StackPanel IsItemsHost="true"
                                       Margin="2"
                                       KeyboardNavigation.TabNavigation="Cycle"
                                       KeyboardNavigationDirectionalNavigation="Cycle" /&gt;
                        &lt;/Border&gt;
                    &lt;/Popup&gt;
&lt;Grid x:Name="Panel"
      Width="24"
      Background="Transparent"
      HorizontalAlignment="Right"&gt;
</pre>

```

```

<Border Visibility="Hidden"
        x:Name="HighlightBorder"
        BorderThickness="1"
        CornerRadius="2">
    <Border.BorderBrush>
        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1">
            <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                    <GradientStop Color="{DynamicResource BorderLightColor}"
                                  Offset="0.0" />
                    <GradientStop Color="{DynamicResource BorderDarkColor}"
                                  Offset="1.0" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Border.BorderBrush>
    <Border.Background>
        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1">
            <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                    <GradientStop Color="{DynamicResource ControlLightColor}" />
                    <GradientStop Color="{DynamicResource ControlMouseOverColor}"
                                  Offset="1.0" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Border.Background>
</Border>
<Path x:Name="Arrow"
      SnapsToDevicePixels="false"
      HorizontalAlignment="Right"
      VerticalAlignment="Center"
      Margin="0,2,4,0"
      StrokeLineJoin="Round"
      Data="M 0 0 L 4 4 L 8 0 Z">
    <Path.Stroke>
        <SolidColorBrush Color="{DynamicResource GlyphColor}" />
    </Path.Stroke>
</Path>
</Grid>
</Grid>
<ControlTemplate.Triggers>
    <Trigger Property="IsHighlighted"
              Value="true">
        <Setter TargetName="HighlightBorder"
                  Property="Visibility"
                  Value="Visible" />
    </Trigger>
    <Trigger Property="Is_submenuOpen"
              Value="true">
        <Setter TargetName="HighlightBorder"
                  Property="BorderBrush">
            <Setter.Value>
                <LinearGradientBrush StartPoint="0,0"
                                    EndPoint="0,1">
                    <GradientBrush.GradientStops>
                        <GradientStopCollection>
                            <GradientStop Color="{DynamicResource BorderDarkColor}"
                                          Offset="0.0" />
                            <GradientStop Color="{DynamicResource BorderMediumColor}"
                                          Offset="1.0" />
                        </GradientStopCollection>
                    </GradientBrush.GradientStops>
                </LinearGradientBrush>
            </Setter.Value>
        </Setter>
    </Trigger>

```

```

        </Setter.Value>
    </Setter>
    <Setter Property="Background"
        TargetName="HighlightBorder">
        <Setter.Value>

            <LinearGradientBrush EndPoint="0,1"
                StartPoint="0,0">
                <GradientStop Color="{DynamicResource ControlLightColor}"
                    Offset="0" />
                <GradientStop Color="{DynamicResource ControlPressedColor}"
                    Offset="0.984" />
            </LinearGradientBrush>
        </Setter.Value>
    </Setter>
</Trigger>
<Trigger Property="IsEnabled"
    Value="false">
    <Setter TargetName="Arrow"
        Property="Fill">
        <Setter.Value>
            <SolidColorBrush Color="{DynamicResource DisabledForegroundColor}" />
        </Setter.Value>
    </Setter>
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<!-- Frame Menu Item Style --&gt;

&lt;Style x:Key="FrameSubmenuItem"
    TargetType="{x:Type MenuItem}"&gt;
    &lt;Setter Property="OverridesDefaultStyle"
        Value="true" /&gt;
    &lt;Setter Property="Header"
        Value="{Binding (JournalEntry.Name)}" /&gt;
    &lt;Setter Property="Command"
        Value="NavigationCommands.NavigateJournal" /&gt;
    &lt;Setter Property="CommandTarget"
        Value="{Binding TemplatedParent,
            RelativeSource={RelativeSource AncestorType={x:Type Menu}}}" /&gt;
    &lt;Setter Property="CommandParameter"
        Value="{Binding RelativeSource={RelativeSource Self}}" /&gt;
    &lt;Setter Property="JournalEntryUnifiedViewConverter.JournalEntryPosition"
        Value="{Binding (JournalEntryUnifiedViewConverter.JournalEntryPosition)}" /&gt;
    &lt;Setter Property="Template"&gt;
        &lt;Setter.Value&gt;
            &lt;ControlTemplate TargetType="{x:Type MenuItem}"&gt;
                &lt;Border BorderThickness="1"
                    Name="Border"&gt;
                    &lt;Grid x:Name="Panel"
                        Background="Transparent"
                        SnapsToDevicePixels="true"
                        Height="35"
                        Width="250"&gt;
                        &lt;Path x:Name="Glyph"
                            SnapsToDevicePixels="false"
                            Margin="7,5"
                            Width="10"
                            Height="10"
                            HorizontalAlignment="Left"
                            StrokeStartLineCap="Triangle"
                            StrokeEndLineCap="Triangle"
                            StrokeThickness="2"&gt;
                            &lt;Path.Stroke&gt;
                                &lt;SolidColorBrush Color="{DynamicResource GlyphColor}" /&gt;
                            &lt;/Path.Stroke&gt;
                        &lt;/Path&gt;
                    &lt;/Grid&gt;
                &lt;/Border&gt;
            &lt;/ControlTemplate&gt;
        &lt;/Setter.Value&gt;
    &lt;/Setter&gt;
&lt;/Style&gt;
</pre>

```

```

        </Path.Stroke>
    </Path>

    <ContentPresenter ContentSource="Header"
                      Margin="24,5,50,5" />
</Grid>
</Border>
<ControlTemplate.Triggers>
    <Trigger Property="JournalEntryUnifiedViewConverter.JournalEntryPosition"
             Value="Current">
        <Setter TargetName="Glyph"
               Property="Data"
               Value="M 0,5 L 2.5,8 L 7,3 " />
    </Trigger>
    <Trigger Property="IsHighlighted"
             Value="true">
        <Setter Property="Background"
               TargetName="Border">
            <Setter.Value>
                <LinearGradientBrush EndPoint="0.5,1"
                                     StartPoint="0.5,0">
                    <GradientStop Color="Transparent"
                                  Offset="0" />
                    <GradientStop Color="{DynamicResource ControlMouseOverColor}"
                                  Offset="1" />
                </LinearGradientBrush>
            </Setter.Value>
        </Setter>
        <Setter Property="BorderBrush"
               TargetName="Border">
            <Setter.Value>
                <LinearGradientBrush EndPoint="0.5,1"
                                     StartPoint="0.5,0">
                    <GradientStop Color="{DynamicResource BorderMediumColor}"
                                  Offset="0" />
                    <GradientStop Color="Transparent"
                                  Offset="1" />
                </LinearGradientBrush>
            </Setter.Value>
        </Setter>
    </Trigger>
    <MultiTrigger>
        <MultiTrigger.Conditions>
            <Condition Property="IsHighlighted"
                       Value="true" />
            <Condition Property="JournalEntryUnifiedViewConverter.JournalEntryPosition"
                       Value="Forward" />
        </MultiTrigger.Conditions>
        <Setter TargetName="Glyph"
               Property="Data"
               Value="M 3 1 L 7 5 L 3 9 z" />
        <Setter TargetName="Glyph"
               Property="Stroke"
               Value="{x:Null}" />
        <Setter TargetName="Glyph"
               Property="Fill">
            <Setter.Value>
                <SolidColorBrush Color="{StaticResource GlyphColor}" />
            </Setter.Value>
        </Setter>
    </MultiTrigger>
    <MultiTrigger>
        <MultiTrigger.Conditions>
            <Condition Property="IsHighlighted"
                       Value="true" />
            <Condition Property="JournalEntryUnifiedViewConverter.JournalEntryPosition"
                       Value="Back" />
        </MultiTrigger.Conditions>
        <Setter TargetName="Glyph"

```

```

        Property="Data"
        Value="M 7 1 L 3 5 L 7 9 z" />
    <Setter TargetName="Glyph"
        Property="Stroke"
        Value="{x:Null}" />
    <Setter TargetName="Glyph"
        Property="Fill">
        <Setter.Value>
            <SolidColorBrush Color="{StaticResource GlyphColor}" />
        </Setter.Value>
    </Setter>
    </MultiTrigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<!-- Merges Back and Forward Navigation Stacks --&gt;

&lt;JournalEntryUnifiedViewConverter x:Key="JournalEntryUnifiedViewConverter" /&gt;

<!-- SimpleStyles: Frame --&gt;

&lt;Style x:Key="{x:Type Frame}"
    TargetType="{x:Type Frame}"&gt;
    &lt;Setter Property="SnapsToDevicePixels"
        Value="true" /&gt;
    &lt;Setter Property="Template"&gt;
        &lt;Setter.Value&gt;
            &lt;ControlTemplate TargetType="{x:Type Frame}"&gt;
                &lt;DockPanel&gt;
                    &lt;Border DockPanel.Dock="Top"
                        Height="22"
                        BorderThickness="1"&gt;
                        &lt;Border.BorderBrush&gt;
                            &lt;LinearGradientBrush EndPoint="0.5,1"
                                StartPoint="0.5,0"&gt;
                                &lt;GradientStop Color="{DynamicResource BorderLightColor}"
                                    Offset="0" /&gt;
                                &lt;GradientStop Color="{DynamicResource BorderDarkColor}"
                                    Offset="1" /&gt;
                            &lt;/LinearGradientBrush&gt;
                        &lt;/Border.BorderBrush&gt;
                    &lt;Grid&gt;
                        &lt;Grid.Background&gt;
                            &lt;LinearGradientBrush StartPoint="0,0"
                                EndPoint="0,1"&gt;
                                &lt;LinearGradientBrush.GradientStops&gt;
                                    &lt;GradientStopCollection&gt;
                                        &lt;GradientStop Color="{DynamicResource ControlLightColor}"
                                            Offset="0.0" /&gt;
                                        &lt;GradientStop Color="{DynamicResource ControlMediumColor}"
                                            Offset="1.0" /&gt;
                                    &lt;/GradientStopCollection&gt;
                                &lt;/LinearGradientBrush.GradientStops&gt;
                            &lt;/LinearGradientBrush&gt;
                        &lt;/Grid.Background&gt;
                    &lt;Grid.ColumnDefinitions&gt;
                        &lt;ColumnDefinition Width="Auto" /&gt;
                        &lt;ColumnDefinition Width="Auto" /&gt;
                        &lt;ColumnDefinition Width="16" /&gt;
                        &lt;ColumnDefinition Width="*" /&gt;
                    &lt;/Grid.ColumnDefinitions&gt;

                    &lt;Menu x:Name="NavMenu"
                        Grid.ColumnSpan="3" /&gt;
                &lt;/Grid&gt;
            &lt;/ControlTemplate&gt;
        &lt;/Setter.Value&gt;
    &lt;/Setter&gt;
&lt;/Style&gt;
</pre>

```

```

        Height="16"
        Margin="1,0,0,0"
        VerticalAlignment="Center"
        Style="{StaticResource FrameMenu}"
    <MenuItem Style="{StaticResource FrameHeaderMenuItem}"
        ItemContainerStyle="{StaticResource FrameSubmenuItem}"
        Is_submenuOpen="{Binding (MenuItem.Is_submenuOpen),"
        Mode=TwoWay, RelativeSource={RelativeSource TemplatedParent}}">
        <MenuItem.ItemsSource>
            <MultiBinding Converter="{StaticResource JournalEntryUnifiedViewConverter}">
                <Binding RelativeSource="{RelativeSource TemplatedParent}"
                    Path="BackStack" />
                <Binding RelativeSource="{RelativeSource TemplatedParent}"
                    Path="ForwardStack" />
            </MultiBinding>
        </MenuItem.ItemsSource>
    </MenuItem>
</Menu>

<Path Grid.Column="0"
    SnapsToDevicePixels="false"
    IsHitTestVisible="false"
    Margin="2,1.5,0,1.5"
    Grid.ColumnSpan="3"
    StrokeThickness="1"
    HorizontalAlignment="Left"
    VerticalAlignment="Center"
    Data="M15,14 Q18,12.9 20.9,14 A8.3,8.3,0,0,0,35.7,8.7 A8.3,8.3,0,0,0,
    25.2,0.6 Q18, 3.3 10.8,0.6 A8.3,8.3,0,0,0,0.3,8.7 A8.3,8.3,0,0,0,15,14 z"
    Stroke="{x:Null}">
    <Path.Fill>
        <LinearGradientBrush EndPoint="0.5,1"
            StartPoint="0.5,0">
            <GradientStop Color="{DynamicResource ControlMediumColor}"
                Offset="0" />
            <GradientStop Color="{DynamicResource ControlDarkColor}"
                Offset="1" />
        </LinearGradientBrush>
    </Path.Fill>
</Path>
<Button Style="{StaticResource FrameButtonStyle}"
    Command="NavigationCommands.BrowseBack"
    Content="M 4 0 L 0 4 L 4 8 Z"
    Margin="2.7,1.5,1.3,1.5"
    Grid.Column="0" />
<Button Style="{StaticResource FrameButtonStyle}"
    Command="NavigationCommands.BrowseForward"
    Content="M 4 0 L 0 4 L 4 8 Z"
    Margin="1.3,1.5,0,1.5"
    Grid.Column="1" />
</Grid>
</Border>
<Border BorderThickness="1">
    <Border.BorderBrush>
        <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
    </Border.BorderBrush>
    <ContentPresenter x:Name="PART_FrameCP"
        Height="458"
        Width="640" />
</Border>
</DockPanel>
<ControlTemplate.Triggers>
    <MultiTrigger>
        <MultiTrigger.Conditions>
            <Condition Property="CanGoForward"
                Value="false" />
            <Condition Property="CanGoBack"
                Value="false" />
        </MultiTrigger.Conditions>
    </MultiTrigger>

```

```

        <Setter TargetName="NavMenu"
            Property="IsEnabled"
            Value="false" />
    </MultiTrigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

The preceding example uses one or more of the following resources.

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"

```

```
        StartPoint="0,0"
        EndPoint="1,0">
<LinearGradientBrush.GradientStops>
    <GradientStopCollection>
        <GradientStop Color="#000000FF"
            Offset="0" />
        <GradientStop Color="#600000FF"
            Offset="0.4" />
        <GradientStop Color="#600000FF"
            Offset="0.6" />
        <GradientStop Color="#000000FF"
            Offset="1" />
    </GradientStopCollection>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# GroupBox Styles and Templates

2 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [GroupBox](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## GroupBox Parts

The [GroupBox](#) control does not have any named parts.

## GroupBox States

The following table lists the visual states for the [GroupBox](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## GroupBox ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [GroupBox](#) control.

```
<Style TargetType="GroupBox">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="GroupBox">
                <Grid>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto" />
                        <RowDefinition Height="*" />
                    </Grid.RowDefinitions>
                    <Border Grid.Row="0"
                           BorderThickness="1"
                           CornerRadius="2,2,0,0">
                        <Border.BorderBrush>
                            <LinearGradientBrush StartPoint="0,0"
                                                EndPoint="0,1">
                                <LinearGradientBrush.GradientStops>
                                    <GradientStopCollection>
                                        <GradientStop Color="{DynamicResource BorderLightColor}"
                                                      Offset="0.0" />
                                        <GradientStop Color="{DynamicResource BorderDarkColor}"
                                                      Offset="1.0" />
                                    </GradientStopCollection>
                                </LinearGradientBrush.GradientStops>
                            </LinearGradientBrush>
                        </Border.BorderBrush>
                    </Border>
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

```

        </LinearGradientBrush>
    </Border.BorderBrush>

    <Border.Background>
        <LinearGradientBrush StartPoint="0,0"
            EndPoint="0,1">
            <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                    <GradientStop Color="{DynamicResource ControlLightColor}"
                        Offset="0.0" />
                    <GradientStop Color="{DynamicResource ControlMediumColor}"
                        Offset="1.0" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Border.Background>

    <ContentPresenter Margin="4"
        ContentSource="Header"
        RecognizesAccessKey="True" />
</Border>

<Border Grid.Row="1"
    BorderThickness="1,0,1,1"
    CornerRadius="0,0,2,2">
    <Border.BorderBrush>
        <SolidColorBrush Color="{StaticResource BorderMediumColor}" />
    </Border.BorderBrush>
    <Border.Background>
        <LinearGradientBrush EndPoint="0.5,1"
            MappingMode="RelativeToBoundingBox"
            StartPoint="0.5,0">
            <GradientStop Color="{DynamicResource ContentAreaColorLight}"
                Offset="0" />
            <GradientStop Color="{DynamicResource ContentAreaColorDark}"
                Offset="1" />
        </LinearGradientBrush>
    </Border.Background>
    <ContentPresenter Margin="4" />
</Border>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

The [ControlTemplate](#) uses one or more of the following resources.

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

```

```

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# Label Styles and Templates

2 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [Label](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## Label Parts

The [Label](#) control does not have any named parts.

## Label States

The following table lists the visual states for the [Label](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## Label ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [Label](#) control.

```

<Style x:Key="{x:Type Label}"
    TargetType="Label">
    <Setter Property="HorizontalContentAlignment"
        Value="Left" />
    <Setter Property="VerticalContentAlignment"
        Value="Top" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="Label">
                <Border>
                    <ContentPresenter HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"
                        VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
                        RecognizesAccessKey="True" />
                </Border>
                <ControlTemplate.Triggers>
                    <Trigger Property="IsEnabled"
                        Value="false">
                        <Setter Property="Foreground">
                            <Setter.Value>
                                <SolidColorBrush Color="{DynamicResource DisabledForegroundColor}" />
                            </Setter.Value>
                        </Setter>
                    </Trigger>
                </ControlTemplate.Triggers>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

The [ControlTemplate](#) uses one or more of the following resources.

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

```

```

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# ListBox Styles and Templates

2 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [ListBox](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## ListBox Parts

The [ListBox](#) control does not have any named parts.

When you create a [ControlTemplate](#) for a [ListBox](#), your template might contain an [ItemsPresenter](#) within a [ScrollViewer](#). (The [ItemsPresenter](#) displays each item in the [ListBox](#); the [ScrollViewer](#) enables scrolling within the control). If the [ItemsPresenter](#) is not the direct child of the [ScrollViewer](#), you must give the [ItemsPresenter](#) the name, `ItemsPresenter`.

## ListBox States

The following table lists the visual states for the [ListBox](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Valid	ValidationStates	The control is valid.
InvalidFocused	ValidationStates	The control is not valid and has focus.
InvalidUnfocused	ValidationStates	The control is not valid and does not have focus.

## ListBoxItem Parts

The [ListBoxItem](#) control does not have any named parts.

## ListBoxItem States

The following table lists the visual states for the [ListBox](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
MouseOver	CommonStates	The mouse pointer is positioned over the control.
Disabled	CommonStates	The item is disabled.
Focused	FocusStates	The item has focus.
Unfocused	FocusStates	The item does not have focus.
Unselected	SelectionStates	The item is not selected.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Selected	SelectionStates	The item is currentlyplate selected.
SelectedUnfocused	SelectionStates	The item is selected, but does not have focus.
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## ListBox ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [ListBox](#) and [ListBoxItem](#) controls.

```

<Style x:Key="{x:Type ListBox}"
       TargetType="ListBox">
    <Setter Property="SnapsToDevicePixels"
           Value="true" />
    <Setter Property="OverridesDefaultStyle"
           Value="true" />
    <Setter Property="ScrollViewer.HorizontalScrollBarVisibility"
           Value="Auto" />
    <Setter Property="ScrollViewer.VerticalScrollBarVisibility"
           Value="Auto" />
    <Setter Property="ScrollViewer.CanContentScroll"
           Value="true" />
    <Setter Property="MinWidth"
           Value="120" />
    <Setter Property="MinHeight"
           Value="95" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="ListBox">
                <Border Name="Border"
                      BorderThickness="1"
                      CornerRadius="2">
                    <Border.Background>
                        <SolidColorBrush Color="{StaticResource ControlLightColor}" />
                    </Border.Background>
                    <Border.BorderBrush>
                        <SolidColorBrush Color="{StaticResource BorderMediumColor}" />
                    </Border.BorderBrush>
                    <ScrollViewer Margin="0"
                                 Focusable="false">
                        <StackPanel Margin="2"
                                   IsItemsHost="True" />
                    </ScrollViewer>
                </Border>
            <ControlTemplate.Triggers>
                <Trigger Property="IsEnabled"
                         Value="false">
                    <Setter TargetName="Border"
                           Property="Background">
                        <Setter.Value>
```

```

        <SolidColorBrush Color="{StaticResource DisabledControlLightColor}" />
    </Setter.Value>
</Setter>
<Setter TargetName="Border"
       Property="BorderBrush">
    <Setter.Value>
        <SolidColorBrush Color="{DynamicResource DisabledBorderLightColor}" />
    </Setter.Value>

    </Setter>
</Trigger>
<Trigger Property="IsGrouping"
         Value="true">
    <Setter Property="ScrollViewer.CanContentScroll"
           Value="false" />
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="{x:Type ListBoxItem}"
       TargetType="ListBoxItem">
    <Setter Property="SnapsToDevicePixels"
           Value="true" />
    <Setter Property="OverridesDefaultStyle"
           Value="true" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="ListBoxItem">
                <Border x:Name="Border"
                        Padding="2"
                        SnapsToDevicePixels="true">
                    <Border.Background>
                        <SolidColorBrush Color="Transparent" />
                    </Border.Background>
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="SelectionStates">
                            <VisualState x:Name="Unselected" />
                            <VisualState x:Name="Selected">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                                               Storyboard.TargetProperty="(Panel.Background).(
                                                               SolidColorBrush.Color)">
                                        <EasingColorKeyFrame KeyTime="0"
                                                               Value="{StaticResource SelectedBackgroundColor}" />
                                    </ColorAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="SelectedUnfocused">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                                               Storyboard.TargetProperty="(Panel.Background).(
                                                               SolidColorBrush.Color)">
                                        <EasingColorKeyFrame KeyTime="0"
                                                               Value="{StaticResource SelectedUnfocusedColor}" />
                                    </ColorAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                    <ContentPresenter />
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

The preceding example uses one or more of the following resources.

```
<!--Control colors-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FCFFFFFF</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

```
        Offset="1" />
    </GradientStopCollection>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# ListView Styles and Templates

4 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [ListView](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## ListView Parts

The [ListView](#) control does not have any named parts.

When you create a [ControlTemplate](#) for a [ListView](#), your template might contain an [ItemsPresenter](#) within a [ScrollViewer](#). (The [ItemsPresenter](#) displays each item in the [ListView](#); the [ScrollViewer](#) enables scrolling within the control). If the [ItemsPresenter](#) is not the direct child of the [ScrollViewer](#), you must give the [ItemsPresenter](#) the name, `ItemsPresenter`.

## ListView States

The following table lists the visual states for the [ListView](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## ListViewItem Parts

The [ListViewItem](#) control does not have any named parts.

## ListViewItem States

The following table lists the states for the [ListViewItem](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
Disabled	CommonStates	The control is disabled.
MouseOver	CommonStates	The mouse pointer is over the <a href="#">ComboBox</a> control.
Focused	FocusStates	The control has focus.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Unfocused	FocusStates	The control does not have focus.
Selected	SelectionStates	The item is currently selected.
Unselected	SelectionStates	The item is not selected.
SelectedUnfocused	SelectionStates	The item is selected, but does not have focus.
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## ListView ControlTemplate Examples

The following example shows how to define a [ControlTemplate](#) for the [ListView](#) control and its associated types.

```
<Style x:Key="{x:Static GridView.GridViewScrollViewerStyleKey}"
       TargetType="ScrollViewer">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="ScrollViewer">
                <Grid Background="{TemplateBinding Background}">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="Auto" />
                    </Grid.ColumnDefinitions>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="*" />
                        <RowDefinition Height="Auto" />
                    </Grid.RowDefinitions>

                    <DockPanel Margin="{TemplateBinding Padding}">
                        <ScrollViewer DockPanel.Dock="Top"
                                      HorizontalScrollBarVisibility="Hidden"
                                      VerticalScrollBarVisibility="Hidden"
                                      Focusable="false">
                            <GridViewHeaderRowPresenter Margin="2,0,2,0"
                                Columns="{Binding Path=TemplatedParent.View.Columns,
                                RelativeSource={RelativeSource TemplatedParent}}"
                                ColumnHeaderContainerStyle="{Binding
                                Path=TemplatedParent.View.ColumnHeaderContainerStyle,
                                RelativeSource={RelativeSource TemplatedParent}}"
                                ColumnHeaderTemplate="{Binding
                                Path=TemplatedParent.View.ColumnHeaderTemplate,
                                RelativeSource={RelativeSource TemplatedParent}}"
                                ColumnHeaderTemplateSelector="{Binding
                                Path=TemplatedParent.View.ColumnHeaderTemplateSelector,
                                RelativeSource={RelativeSource TemplatedParent}}"
                                AllowsColumnReorder="{Binding
                                Path=TemplatedParent.ViewAllowsColumnReorder,
                                RelativeSource={RelativeSource TemplatedParent}}"
                                Path=TemplatedParent.View.AllowColumnReorder,
                                RelativeSource={RelativeSource TemplatedParent}}>
```

```

        ColumnHeaderContextMenu="{Binding
            Path=TemplatedParent.View.ColumnHeaderContextMenu,
            RelativeSource={RelativeSource TemplatedParent}}"
            ColumnHeaderToolTip="{Binding
            Path=TemplatedParent.View.ColumnHeaderToolTip,
            RelativeSource={RelativeSource TemplatedParent}}"
            SnapsToDevicePixels="{TemplateBinding
            SnapsToDevicePixels}" />
    </ScrollViewer>

    <ScrollContentPresenter Name="PART_ScrollContentPresenter"
        KeyboardNavigation.DirectionalNavigation="Local"
        CanContentScroll="True"
        CanHorizontallyScroll="False"
        CanVerticallyScroll="False" />
</DockPanel>

<ScrollBar Name="PART_HorizontalScrollBar"
    Orientation="Horizontal"
    Grid.Row="1"
    Maximum="{TemplateBinding ScrollableWidth}"
    ViewportSize="{TemplateBinding ViewportWidth}"
    Value="{TemplateBinding HorizontalOffset}"
    Visibility="{TemplateBinding ComputedHorizontalScrollBarVisibility}" />

<ScrollBar Name="PART_VerticalScrollBar"
    Grid.Column="1"
    Maximum="{TemplateBinding ScrollableHeight}"
    ViewportSize="{TemplateBinding ViewportHeight}"
    Value="{TemplateBinding VerticalOffset}"
    Visibility="{TemplateBinding ComputedVerticalScrollBarVisibility}" />

</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="GridViewColumnHeaderGripper"
    TargetType="Thumb">
    <Setter Property="Width"
        Value="18" />
    <Setter Property="Background">
        <Setter.Value>
            <LinearGradientBrush StartPoint="0,0"
                EndPoint="0,1">
                <LinearGradientBrush.GradientStops>
                    <GradientStopCollection>
                        <GradientStop Color="{DynamicResource BorderLightColor}"
                            Offset="0.0" />
                        <GradientStop Color="{DynamicResource BorderDarkColor}"
                            Offset="1.0" />
                    </GradientStopCollection>
                </LinearGradientBrush.GradientStops>
            </LinearGradientBrush>
        </Setter.Value>
    </Setter>
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Thumb}">
                <Border Padding="{TemplateBinding Padding}"
                    Background="Transparent">
                    <Rectangle HorizontalAlignment="Center"
                        Width="1"
                        Fill="{TemplateBinding Background}" />
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>

```

```

<Setter Property="BorderBrush">
    <Setter.Value>
        <LinearGradientBrush EndPoint="0.5,1"
                            StartPoint="0.5,0">
            <GradientStop Color="Black"
                          Offset="0" />
            <GradientStop Color="White"
                          Offset="1" />
        </LinearGradientBrush>
    </Setter.Value>
</Setter>
</Style>

<Style x:Key="{x:Type GridViewColumnHeader}"
       TargetType="GridViewColumnHeader">
    <Setter Property="HorizontalContentAlignment"
           Value="Center" />
    <Setter Property="VerticalContentAlignment"
           Value="Center" />
    <Setter Property="Foreground"
           Value="{DynamicResource {x:Static SystemColors.ControlTextBrushKey}}" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="GridViewColumnHeader">
                <Grid>
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="Normal" />
                            <VisualState x:Name="MouseOver">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Panel.Background).(
                                        GradientBrush.GradientStops)[1].(GradientStop.Color)"
                                        Storyboard.TargetName="HeaderBorder">
                                        <EasingColorKeyFrame KeyTime="0"
                                            Value="{StaticResource ControlMouseOverColor}" />
                                    </ColorAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="Pressed" />
                            <VisualState x:Name="Disabled" />
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                    <Border x:Name="HeaderBorder"
                           BorderThickness="0,1,0,1"
                           Padding="2,0,2,0">
                        <Border.BorderBrush>
                            <LinearGradientBrush StartPoint="0,0"
                                                EndPoint="0,1">
                                <LinearGradientBrush.GradientStops>
                                    <GradientStopCollection>
                                        <GradientStop Color="{DynamicResource BorderLightColor}"
                                                      Offset="0.0" />
                                        <GradientStop Color="{DynamicResource BorderDarkColor}"
                                                      Offset="1.0" />
                                    </GradientStopCollection>
                                </LinearGradientBrush.GradientStops>
                            </LinearGradientBrush>
                        </Border.BorderBrush>
                        <Border.Background>
                            <LinearGradientBrush StartPoint="0,0"
                                                EndPoint="0,1">
                                <LinearGradientBrush.GradientStops>
                                    <GradientStopCollection>
                                        <GradientStop Color="{DynamicResource ControlLightColor}"
                                                      Offset="0.0" />
                                        <GradientStop Color="{DynamicResource ControlMediumColor}"
                                                      Offset="1.0" />
                                    </GradientStopCollection>
                                </LinearGradientBrush.GradientStops>
                            </LinearGradientBrush>
                        </Border.Background>
                    </Border>
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

```

        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>

</Border.Background>
<ContentPresenter x:Name="HeaderContent"
    Margin="0,0,0,1"
    RecognizesAccessKey="True"
    VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
    HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"
    SnapsToDevicePixels="{TemplateBinding SnapsToDevicePixels}"/>
</Border>
<Thumb x:Name="PART_HeaderGripper"
    HorizontalAlignment="Right"
    Margin="0,0,-9,0"
    Style="{StaticResource GridViewColumnHeaderGripper}" />
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
<Style.Triggers>
    <Trigger Property="Role"
        Value="Floating">
        <Setter Property="Opacity"
            Value="0.7" />
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="GridViewColumnHeader">
                    <Canvas Name="PART_FloatingHeaderCanvas">
                        <Rectangle Fill="#60000000"
                            Width="{TemplateBinding ActualWidth}"
                            Height="{TemplateBinding ActualHeight}" />
                    </Canvas>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Trigger>
    <Trigger Property="Role"
        Value="Padding">
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="GridViewColumnHeader">
                    <Border Name="HeaderBorder"
                        BorderThickness="0,1,0,1">
                        <Border.Background>
                            <SolidColorBrush Color="{DynamicResource ControlLightColor}" />
                        </Border.Background>
                        <Border.BorderBrush>
                            <LinearGradientBrush StartPoint="0,0"
                                EndPoint="0,1">
                                <LinearGradientBrush.GradientStops>
                                    <GradientStopCollection>
                                        <GradientStop Color="{DynamicResource BorderLightColor}"
                                            Offset="0.0" />
                                        <GradientStop Color="{DynamicResource BorderDarkColor}"
                                            Offset="1.0" />
                                    </GradientStopCollection>
                                </LinearGradientBrush.GradientStops>
                            </LinearGradientBrush>
                        </Border.BorderBrush>
                    </Border>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Trigger>
</Style.Triggers>
</Style>
<Style x:Key="svType_ListView1">

```

```

<Style x:Key="{x:Type ListView}">
    <Setter Property="SnapsToDevicePixels"
        Value="true" />
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="ScrollViewer.HorizontalScrollBarVisibility"
        Value="Auto" />
    <Setter Property="ScrollViewer.VerticalScrollBarVisibility"
        Value="Auto" />
    <Setter Property="ScrollViewer.CanContentScroll"
        Value="true" />
    <Setter Property="VerticalContentAlignment"
        Value="Center" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="ListView">
                <Border Name="Border"
                    BorderThickness="1">
                    <Border.Background>
                        <SolidColorBrush Color="{StaticResource ControlLightColor}" />
                    </Border.Background>
                    <Border.BorderBrush>
                        <SolidColorBrush Color="{StaticResource BorderMediumColor}" />
                    </Border.BorderBrush>
                    <ScrollViewer Style="{DynamicResource
                        {x:Static GridView.GridViewScrollViewerStyleKey}}">
                        <ItemsPresenter />
                    </ScrollViewer>
                </Border>
                <ControlTemplate.Triggers>
                    <Trigger Property="IsGrouping"
                        Value="true">
                        <Setter Property="ScrollViewer.CanContentScroll"
                            Value="false" />
                    </Trigger>
                    <Trigger Property="IsEnabled"
                        Value="false">
                        <Setter TargetName="Border"
                            Property="Background">
                            <Setter.Value>
                                <SolidColorBrush Color="{DynamicResource DisabledBorderLightColor}" />
                            </Setter.Value>
                        </Setter>
                    </Trigger>
                </ControlTemplate.Triggers>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

<Style x:Key="{x:Type ListViewItem}"
    TargetType="ListViewItem">
    <Setter Property="SnapsToDevicePixels"
        Value="true" />
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="ListBoxItem">
                <Border x:Name="Border"
                    Padding="2"
                    SnapsToDevicePixels="true"
                    Background="Transparent">
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="Normal" />
                            <VisualState x:Name="MouseOver" />
                            <VisualState x:Name="Disabled" />
                        </VisualStateGroup>
                    </VisualStateManager>
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

```

</visualStateGroup>
<VisualStateGroup x:Name="SelectionStates">
    <VisualState x:Name="Unselected" />
    <VisualState x:Name="Selected">
        <Storyboard>
            <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                Storyboard.TargetProperty="(Panel.Background).(
                    SolidColorBrush.Color)">
                <EasingColorKeyFrame KeyTime="0"
                    Value="{StaticResource SelectedBackgroundColor}" />
            </ColorAnimationUsingKeyFrames>
        </Storyboard>
    </VisualState>
    <VisualState x:Name="SelectedUnfocused">
        <Storyboard>
            <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                Storyboard.TargetProperty="(Panel.Background).(
                    SolidColorBrush.Color)">
                <EasingColorKeyFrame KeyTime="0"
                    Value="{StaticResource SelectedUnfocusedColor}" />
            </ColorAnimationUsingKeyFrames>
        </Storyboard>
    </VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<GridViewRowPresenter VerticalAlignment="{TemplateBinding VerticalContentAlignment}" />
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

The [ControlTemplate](#) examples use one or more of the following resources.

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

```

```

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# Menu Styles and Templates

6 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [Menu](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## Menu Parts

The [Menu](#) control does not have any named parts.

When you create a [ControlTemplate](#) for a [Menu](#), your template might contain an [ItemsPresenter](#) within a [ScrollViewer](#). (The [ItemsPresenter](#) displays each item in the [Menu](#); the [ScrollViewer](#) enables scrolling within the control). If the [ItemsPresenter](#) is not the direct child of the [ScrollViewer](#), you must give the [ItemsPresenter](#) the name, `ItemsPresenter`.

## Menu States

The following table lists the visual states for the [Menu](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## MenuItem Parts

The following table lists the named parts for the [Menu](#) control.

PART	TYPE	DESCRIPTION
PART_Popup	Popup	The area for the submenu.

When you create a [ControlTemplate](#) for a [MenuItem](#), your template might contain an [ItemsPresenter](#) within a [ScrollViewer](#). (The [ItemsPresenter](#) displays each item in the [MenuItem](#); the [ScrollViewer](#) enables scrolling within the control). If the [ItemsPresenter](#) is not the direct child of the [ScrollViewer](#), you must give the [ItemsPresenter](#) the name, `ItemsPresenter`.

## MenuItem States

The following table lists the visual states for the [MenuItem](#) control.

VisualState Name	VisualStateGroup Name	Description
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## Menu and MenuItem ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [Menu](#) control.

```
<Style x:Key="{x:Type Menu}"
    TargetType="{x:Type Menu}">
<Setter Property="OverridesDefaultStyle"
    Value="True" />
<Setter Property="SnapsToDevicePixels"
    Value="True" />
<Setter Property="Template">
    <Setter.Value>
        <ControlTemplate TargetType="{x:Type Menu}">
            <Border BorderThickness="1">
                <Border.BorderBrush>
                    <LinearGradientBrush StartPoint="0,0"
                        EndPoint="0,1">
                        <LinearGradientBrush.GradientStops>
                            <GradientStopCollection>
                                <GradientStop Color="{DynamicResource BorderLightColor}"
                                    Offset="0.0" />
                                <GradientStop Color="{DynamicResource BorderDarkColor}"
                                    Offset="1.0" />
                            </GradientStopCollection>
                        </LinearGradientBrush.GradientStops>
                    </LinearGradientBrush>
                </Border.BorderBrush>
                <Border.Background>
                    <LinearGradientBrush EndPoint="0.5,1"
                        StartPoint="0.5,0">
                        <GradientStop Color="{DynamicResource ControlLightColor}"
                            Offset="0" />
                        <GradientStop Color="{DynamicResource ControlMediumColor}"
                            Offset="1" />
                    </LinearGradientBrush>
                </Border.Background>
                <StackPanel ClipToBounds="True"
                    Orientation="Horizontal"
                    IsItemsHost="True" />
            </Border>
        </ControlTemplate>
    </Setter.Value>
</Setter>
</Style>
```

The following example shows how to define a [ControlTemplate](#) for the [MenuItem](#) control.

```
<Style x:Key="{x:Static MenuItem.SeparatorStyleKey}">
```

```

    <Style Key="Type Separator Style">
        <Setter Property="TargetType" Value="{x:Type Separator}" />
        <Setter Property="Height" Value="1" />
        <Setter Property="Margin" Value="0,4,0,4" />
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="{x:Type Separator}">
                    <Border BorderThickness="1">
                        <Border.BorderBrush>
                            <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
                        </Border.BorderBrush>
                    </Border>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>

    <!-- TopLevelHeader -->
    <ControlTemplate x:Key="{x:Static MenuItem.TopLevelHeaderTemplateKey}" TargetType="{x:Type MenuItem}">
        <Border x:Name="Border">
            <Grid>
                <ContentPresenter Margin="6,3,6,3" ContentSource="Header" RecognizesAccessKey="True" />
                <Popup x:Name="Popup" Placement="Bottom" IsOpen="{TemplateBinding IsSubmenuOpen}" AllowsTransparency="True" Focusable="False" PopupAnimation="Fade">
                    <Border x:Name="SubmenuBorder" SnapsToDevicePixels="True" BorderThickness="1" Background="{DynamicResource MenuPopupBrush}">
                        <Border.BorderBrush>
                            <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
                        </Border.BorderBrush>
                    </Border>
                    <ScrollViewer CanContentScroll="True" Style="{StaticResource MenuScrollViewer}">
                        <StackPanel IsItemsHost="True" KeyboardNavigation.Directionality="Cycle" />
                    </ScrollViewer>
                </Popup>
            </Grid>
        </Border>
        <ControlTemplate.Triggers>
            <Trigger Property="IsSuspendedPopupAnimation" Value="true">
                <Setter TargetName="Popup" Property="PopupAnimation" Value="None" />
            </Trigger>
            <Trigger Property="IsHighlighted" Value="true">
                <Setter TargetName="Border" Property="BorderBrush" Value="Transparent" />
                <Setter Property="Background" TargetName="Border">
                    <Setter.Value>
                        <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
                            <LinearGradientBrush.GradientStops>
                                <GradientStop Collection>
                                    <GradientStop Color="{StaticResource ControlLightColor}" />
                                    <GradientStop Color="{StaticResource ControlMouseOverColor}" />
                                </GradientStop Collection>
                            </LinearGradientBrush.GradientStops>
                        </LinearGradientBrush>
                    </Setter.Value>
                </Setter>
            </Trigger>
        </ControlTemplate.Triggers>
    </ControlTemplate>

```

```

        <GradientStop Color="{StaticResource ControlMouseOverColor}"
                      Offset="1.0" />
    </GradientStopCollection>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>

</Setter.Value>
</Setter>
</Trigger>
<Trigger SourceName="Popup"
         Property="AllowsTransparency"
         Value="True">
    <Setter TargetName="SubmenuBorder"
            Property="CornerRadius"
            Value="0,0,4,4" />
    <Setter TargetName="SubmenuBorder"
            Property="Padding"
            Value="0,0,0,3" />
</Trigger>
<Trigger Property="IsEnabled"
         Value="False">
    <Setter Property="Foreground">
        <Setter.Value>
            <SolidColorBrush Color="{StaticResource DisabledForegroundColor}" />
        </Setter.Value>
    </Setter>
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>

<!-- TopLevelItem -->
<ControlTemplate x:Key="{x:Static MenuItem.TopLevelItemTemplateKey}"
                 TargetType="{x:Type MenuItem}">
    <Border x:Name="Border">
        <Grid>
            <ContentPresenter Margin="6,3,6,3"
                              ContentSource="Header"
                              RecognizesAccessKey="True" />
        </Grid>
    </Border>
    <ControlTemplate.Triggers>
        <Trigger Property="IsHighlighted"
                 Value="true">
            <Setter Property="Background"
                    TargetName="Border">
                <Setter.Value>
                    <LinearGradientBrush StartPoint="0,0"
                                         EndPoint="0,1">
                        <LinearGradientBrush.GradientStops>
                            <GradientStopCollection>
                                <GradientStop Color="{StaticResource ControlLightColor}" />
                                <GradientStop Color="{StaticResource ControlMouseOverColor}"
                                              Offset="1.0" />
                            </GradientStopCollection>
                        </LinearGradientBrush.GradientStops>
                    </LinearGradientBrush>
                </Setter.Value>
            </Setter>
        </Trigger>
        <Trigger Property="IsEnabled"
                 Value="False">
            <Setter Property="Foreground">
                <Setter.Value>
                    <SolidColorBrush Color="{StaticResource DisabledForegroundColor}" />
                </Setter.Value>
            </Setter>
        </Trigger>
    </ControlTemplate.Triggers>
</ControlTemplate>

```

```

</ControlTemplate>

<!-- SubmenuItem -->
<ControlTemplate x:Key="{x:Static MenuItem.SubmenuItemTemplateKey}"
    TargetType="{x:Type MenuItem}">
    <Border x:Name="Border"
        BorderThickness="1">
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto"
                    SharedSizeGroup="Icon" />
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="Auto"
                    SharedSizeGroup="Shortcut" />
                <ColumnDefinition Width="13" />
            </Grid.ColumnDefinitions>
            <ContentPresenter x:Name="Icon"
                Margin="6,0,6,0"
                VerticalAlignment="Center"
                ContentSource="Icon" />
            <Border x:Name="Check"
                Width="13"
                Height="13"
                Visibility="Collapsed"
                Margin="6,0,6,0"
                BorderThickness="1">
                <Border.BorderBrush>
                    <LinearGradientBrush StartPoint="0,0"
                        EndPoint="0,1">
                        <LinearGradientBrush.GradientStops>
                            <GradientStopCollection>
                                <GradientStop Color="{DynamicResource BorderLightColor}"
                                    Offset="0.0" />
                                <GradientStop Color="{DynamicResource BorderDarkColor}"
                                    Offset="1.0" />
                            </GradientStopCollection>
                        </LinearGradientBrush.GradientStops>
                    </LinearGradientBrush>
                </Border.BorderBrush>
                <Border.Background>
                    <LinearGradientBrush StartPoint="0,0"
                        EndPoint="0,1">
                        <LinearGradientBrush.GradientStops>
                            <GradientStopCollection>
                                <GradientStop Color="{DynamicResource ControlLightColor}" />
                                <GradientStop Color="{DynamicResource ControlMediumColor}"
                                    Offset="1.0" />
                            </GradientStopCollection>
                        </LinearGradientBrush.GradientStops>
                    </LinearGradientBrush>
                </Border.Background>
            <Path x:Name="CheckMark"
                Width="7"
                Height="7"
                Visibility="Hidden"
                SnapsToDevicePixels="False"
                StrokeThickness="2"
                Data="M 0 0 L 7 7 M 0 7 L 7 0">
                <Path.Stroke>
                    <SolidColorBrush Color="{DynamicResource GlyphColor}" />
                </Path.Stroke>
            </Path>
        </Border>
        <ContentPresenter x:Name="HeaderHost"
            Grid.Column="1"
            ContentSource="Header"
            RecognizesAccessKey="True" />
        <TextBlock x:Name="InputGestureText">

```

```

        Grid.Column="2"
        Text="{TemplateBinding InputGestureText}"
        Margin="5,2,0,2"
        DockPanel.Dock="Right" />
    </Grid>
</Border>
<ControlTemplate.Triggers>
    <Trigger Property="ButtonBase.Command"
        Value="{x:Null}" />
    <Trigger Property="Icon"
        Value="{x:Null}">
        <Setter TargetName="Icon"
            Property="Visibility"
            Value="Hidden" />
    </Trigger>
    <Trigger Property="IsChecked"
        Value="true">
        <Setter TargetName="CheckMark"
            Property="Visibility"
            Value="Visible" />
    </Trigger>
    <Trigger Property="IsCheckable"
        Value="true">
        <Setter TargetName="Check"
            Property="Visibility"
            Value="Visible" />
        <Setter TargetName="Icon"
            Property="Visibility"
            Value="Hidden" />
    </Trigger>
    <Trigger Property="IsHighlighted"
        Value="true">
        <Setter Property="Background"
            TargetName="Border">
            <Setter.Value>
                <LinearGradientBrush EndPoint="0.5,1"
                    StartPoint="0.5,0">
                    <GradientStop Color="Transparent"
                        Offset="0" />
                    <GradientStop Color="{DynamicResource ControlMouseOverColor}"
                        Offset="1" />
                </LinearGradientBrush>
            </Setter.Value>
        </Setter>
        <Setter Property="BorderBrush"
            TargetName="Border">
            <Setter.Value>
                <LinearGradientBrush EndPoint="0.5,1"
                    StartPoint="0.5,0">
                    <GradientStop Color="{DynamicResource BorderMediumColor}"
                        Offset="0" />
                    <GradientStop Color="Transparent"
                        Offset="1" />
                </LinearGradientBrush>
            </Setter.Value>
        </Setter>
    </Trigger>
    <Trigger Property="IsEnabled"
        Value="false">
        <Setter Property="Foreground">
            <Setter.Value>
                <SolidColorBrush Color="{StaticResource DisabledForegroundColor}" />
            </Setter.Value>
        </Setter>
    </Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>

<ControlTemplate x:Key="{x:Static MenuItem.SubmenuHeaderTemplateKey}">

```

```

        TargetType="{x:Type MenuItem}">
<Border x:Name="Border"
        BorderThickness="1">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"
                SharedSizeGroup="Icon" />
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="Auto"
                SharedSizeGroup="Shortcut" />
            <ColumnDefinition Width="13" />
        </Grid.ColumnDefinitions>
        <ContentPresenter x:Name="Icon"
            Margin="6,0,6,0"
            VerticalAlignment="Center"
            ContentSource="Icon" />
        <ContentPresenter x:Name="HeaderHost"
            Grid.Column="1"
            ContentSource="Header"
            RecognizesAccessKey="True" />
        <TextBlock x:Name="InputGestureText"
            Grid.Column="2"
            Text="{TemplateBinding InputGestureText}"
            Margin="5,2,2,2"
            DockPanel.Dock="Right" />
        <Path Grid.Column="3"
            HorizontalAlignment="Center"
            VerticalAlignment="Center"
            Data="M 0 0 L 0 7 L 4 3.5 Z">
            <Path.Fill>
                <SolidColorBrush Color="{DynamicResource GlyphColor}" />
            </Path.Fill>
        </Path>
        <Popup x:Name="Popup"
            Placement="Right"
            HorizontalOffset="-4"
            IsOpen="{TemplateBinding Is_submenuOpen}"
            AllowsTransparency="True"
            Focusable="False"
            PopupAnimation="Fade">
            <Border x:Name="SubmenuBorder"
                SnapsToDevicePixels="True"
                Background="{DynamicResource MenuPopupBrush}"
                BorderThickness="1">
                <Border.BorderBrush>
                    <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
                </Border.BorderBrush>
                <ScrollViewer CanContentScroll="True"
                    Style="{StaticResource MenuScrollViewer}">
                    <StackPanel IsItemsHost="True"
                        KeyboardNavigation.DirectionalNavigation="Cycle" />
                </ScrollViewer>
            </Border>
        </Popup>
    </Grid>
</Border>
<ControlTemplate.Triggers>
    <Trigger Property="Icon"
        Value="{x:Null}">
        <Setter TargetName="Icon"
            Property="Visibility"
            Value="Collapsed" />
    </Trigger>
    <Trigger Property="IsHighlighted"
        Value="true">
        <Setter Property="Background"
            TargetName="Border">
            <Setter.Value>
                <LinearGradientBrush EndPoint="0.5,1"

```

```

        StartPoint="0.5,0">
    <GradientStop Color="Transparent"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMouseOverColor}"
        Offset="1" />
</LinearGradientBrush>
</Setter.Value>
</Setter>
<Setter Property="BorderBrush"
    TargetName="Border">
<Setter.Value>
    <LinearGradientBrush EndPoint="0.5,1"
        StartPoint="0.5,0">
        <GradientStop Color="{DynamicResource BorderMediumColor}"
            Offset="0" />
        <GradientStop Color="Transparent"
            Offset="1" />
    </LinearGradientBrush>
</Setter.Value>
</Setter>
</Trigger>
<Trigger SourceName="Popup"
    Property="AllowsTransparency"
    Value="True">
<Setter TargetName="SubmenuBorder"
    Property="CornerRadius"
    Value="4" />
<Setter TargetName="SubmenuBorder"
    Property="Padding"
    Value="0,3,0,3" />
</Trigger>
<Trigger Property="IsEnabled"
    Value="false">
<Setter Property="Foreground">
<Setter.Value>
    <SolidColorBrush Color="{StaticResource DisabledForegroundColor}" />
</Setter.Value>
</Setter>
</Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>

<!-- MenuItem Style --&gt;
&lt;Style x:Key="{x:Type MenuItem}"
    TargetType="{x:Type MenuItem}"&gt;
    &lt;Setter Property="OverridesDefaultCellStyle"
        Value="True" /&gt;
&lt;Style.Triggers&gt;
    &lt;Trigger Property="Role"
        Value="TopLevelHeader"&gt;
        &lt;Setter Property="Template"
            Value="{StaticResource {x:Static MenuItem.TopLevelHeaderTemplateKey}}" /&gt;
        &lt;Setter Property="Grid.IsSharedSizeScope"
            Value="true" /&gt;
    &lt;/Trigger&gt;
    &lt;Trigger Property="Role"
        Value="TopLevelItem"&gt;
        &lt;Setter Property="Template"
            Value="{StaticResource {x:Static MenuItem.TopLevelItemTemplateKey}}" /&gt;
    &lt;/Trigger&gt;
    &lt;Trigger Property="Role"
        Value="SubMenuHeader"&gt;
        &lt;Setter Property="Template"
            Value="{StaticResource {x:Static MenuItem.SubmenuHeaderTemplateKey}}" /&gt;
    &lt;/Trigger&gt;
    &lt;Trigger Property="Role"
        Value="MenuItem"&gt;
        &lt;Setter Property="Template"
            Value="{StaticResource {x:Static MenuItem.SubmenuItemTemplateKey}}" /&gt;
    &lt;/Trigger&gt;
&lt;/Style.Triggers&gt;
</pre>

```

```

    </Trigger>
</Style.Triggers>
</Style>

```

The following example defines the `MenuScrollViewer`, which is used in the previous example.

```

<!--ScrollView for a MenuItem-->
<MenuScrollingVisibilityConverter x:Key="MenuScrollingVisibilityConverter" />

<Style x:Key="MenuScrollViewer"
    TargetType="{x:Type ScrollViewer}"
    BasedOn="{x:Null}">
    <Setter Property="HorizontalScrollBarVisibility"
        Value="Hidden" />
    <Setter Property="VerticalScrollBarVisibility"
        Value="Auto" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type ScrollViewer}">
                <Grid SnapsToDevicePixels="True">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="*" />
                    </Grid.ColumnDefinitions>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto" />
                        <RowDefinition Height="*" />
                        <RowDefinition Height="Auto" />
                    </Grid.RowDefinitions>
                    <Border Grid.Row="1"
                        Grid.Column="0">
                        <ScrollContentPresenter Margin="{TemplateBinding Padding}" />
                    </Border>
                    <RepeatButton Style="{StaticResource MenuScrollButton}"
                        Grid.Row="0"
                        Grid.Column="0"
                        Command="{x:Static ScrollBar.LineUpCommand}"
                        CommandTarget="{Binding RelativeSource={RelativeSource TemplatedParent}}"
                        Focusable="False">
                        <RepeatButton.Visibility>
                            <MultiBinding FallbackValue="Visibility.Collapsed"
                                Converter="{StaticResource MenuScrollingVisibilityConverter}"
                                ConverterParameter="0">
                                <Binding RelativeSource="{RelativeSource TemplatedParent}"
                                    Path="ComputedVerticalScrollBarVisibility" />
                                <Binding RelativeSource="{RelativeSource TemplatedParent}"
                                    Path="VerticalOffset" />
                                <Binding RelativeSource="{RelativeSource TemplatedParent}"
                                    Path="ExtentHeight" />
                                <Binding RelativeSource="{RelativeSource TemplatedParent}"
                                    Path="ViewportHeight" />
                            </MultiBinding>
                        </RepeatButton.Visibility>
                        <Path Fill="{DynamicResource {x:Static SystemColors.MenuTextBrushKey}}"
                            Data="{StaticResource UpArrow}" />
                    </RepeatButton>
                    <RepeatButton Style="{StaticResource MenuScrollButton}"
                        Grid.Row="2"
                        Grid.Column="0"
                        Command="{x:Static ScrollBar.LineDownCommand}"
                        CommandTarget="{Binding RelativeSource={RelativeSource TemplatedParent}}"
                        Focusable="False">
                        <RepeatButton.Visibility>
                            <MultiBinding FallbackValue="Visibility.Collapsed"
                                Converter="{StaticResource MenuScrollingVisibilityConverter}"
                                ConverterParameter="100">
                                <Binding RelativeSource="{RelativeSource TemplatedParent}"
                                    Path="ComputedVerticalScrollBarVisibility" />
                            </MultiBinding>
                        </RepeatButton.Visibility>
                    </RepeatButton>
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

```

        <MultiBinding>
            <Binding RelativeSource="{RelativeSource TemplatedParent}"
                    Path="VerticalOffset" />
            <Binding RelativeSource="{RelativeSource TemplatedParent}"
                    Path="ExtentHeight" />
            <Binding RelativeSource="{RelativeSource TemplatedParent}"
                    Path="ViewportHeight" />
        </MultiBinding>
    </RepeatButton.Visibility>
    <Path Fill="{DynamicResource {x:Static SystemColors.MenuTextBrushKey}}"
          Data="{StaticResource DownArrow}" />
</RepeatButton>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

The [ControlTemplate](#) examples use one or more of the following resources.

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />

```

```
<GradientStop Color="{DynamicResource ControlMediumColor}"
    Offset="0.5" />
<GradientStop Color="{DynamicResource ControlLightColor}"
    Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
<LinearGradientBrush.GradientStops>
    <GradientStopCollection>
        <GradientStop Color="#000000FF"
            Offset="0" />
        <GradientStop Color="#600000FF"
            Offset="0.4" />
        <GradientStop Color="#600000FF"
            Offset="0.6" />
        <GradientStop Color="#000000FF"
            Offset="1" />
    </GradientStopCollection>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# NavigationWindow Styles and Templates

5 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [NavigationWindow](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

# NavigationWindow Parts

The following table lists the named parts for the [NavigationWindow](#) control.

PART	TYPE	DESCRIPTION
PART_NavWinCP	ContentPresenter	The area for the content.

## NavigationWindow States

The following table lists the visual states for the [NavigationWindow](#) control.

VisualState Name	VisualStateGroup Name	Description
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

# NavigationWindow ControlTemplate Example

Although this example contains all of the elements that are defined in the [ControlTemplate](#) of a [NavigationWindow](#) by default, the specific values should be thought of as examples.

```
<Style x:Key="NavWinButtonStyle"
    TargetType="{x:Type Button}>
        <Setter Property="OverridesDefaultStyle"
            Value="true" />
        <Setter Property="Command"
            Value="NavigationCommands.BrowseBack" />
        <Setter Property="Focusable"
            Value="false" />
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="{x:Type Button}">
                    <Grid>
                        <VisualStateManager.VisualStateGroups>
                            <VisualStateGroup x:Name="CommonStates">
                                <VisualState x:Name="Normal" />
                                <VisualState x:Name="MouseOver" />
                                <VisualState x:Name="Pressed" />
                                <VisualState x:Name="Disabled" />
                            </VisualStateGroup>
                        </VisualStateManager.VisualStateGroups>
                    </Grid>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
```

```

<Storyboard>
    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Ellipse"
        Storyboard.TargetProperty="(Shape.Fill).{StaticResource ControlMouseOverColor}" />
    <EasingColorKeyFrame KeyTime="0"
        Value="{StaticResource ControlMouseOverColor}" />
</ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Pressed">
    <Storyboard>
        <ColorAnimationUsingKeyFrames Storyboard.TargetName="Ellipse"
            Storyboard.TargetProperty="(Shape.Fill).{StaticResource ControlPressedColor}" />
        <EasingColorKeyFrame KeyTime="0"
            Value="{StaticResource ControlPressedColor}" />
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Disabled">
    <Storyboard>
        <ColorAnimationUsingKeyFrames Storyboard.TargetName="Ellipse"
            Storyboard.TargetProperty="(Shape.Fill).{StaticResource DisabledControlDarkColor}" />
        <ColorAnimationUsingKeyFrames Storyboard.TargetName="Arrow"
            Storyboard.TargetProperty="(Shape.Fill).{StaticResource DisabledForegroundColor}" />
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateManager.VisualStateGroups>
<Ellipse x:Name="Ellipse">
    <StrokeThickness>1</StrokeThickness>
    <Width>24</Width>
    <Height>24</Height>
    <Stroke>
        <SolidColorBrush Color="{DynamicResource NavButtonFrameColor}" />
    </Stroke>
    <Fill>
        <LinearGradientBrush StartPoint="0,0"
            EndPoint="0,1">
            <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                    <GradientStop Color="{DynamicResource ControlLightColor}" />
                    <GradientStop Color="{DynamicResource ControlMediumColor}"
                        Offset="1.0" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Fill>
</Ellipse>
<Path x:Name="Arrow">
    <Margin>0,0,3,0</Margin>
    <HorizontalAlignment>Center</HorizontalAlignment>
    <VerticalAlignment>Center</VerticalAlignment>
    <Data>M 6 0 L 0 6 L 6 12 Z</Data>
    <Fill>
        <SolidColorBrush Color="{DynamicResource GlyphColor}" />
    </Fill>
</Path>
</Grid>

```

```

<ControlTemplate.Triggers>
    <Trigger Property="Command"
        Value="{x:Static NavigationCommands.BrowseForward}">
        <Setter TargetName="Arrow"
            Property="Data"
            Value="M 0 0 L 6 6 L 0 12 z" />
        <Setter TargetName="Arrow"
            Property="Margin"
            Value="3,0,0,0" />
    </Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<!-- NavWin Menu Style -->
<Style x:Key="NavWinMenu"
    TargetType="{x:Type Menu}">
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="KeyboardNavigation.TabNavigation"
        Value="None" />
    <Setter Property="IsMainMenu"
        Value="false" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Menu}">
                <DockPanel IsItemsHost="true" />
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

<!-- NavWin Menu Header Style -->
<Style x:Key="NavWinHeaderMenuItem"
    TargetType="{x:Type MenuItem}">
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type MenuItem}">
                <Grid>
                    <Popup x:Name="PART_Popup"
                        Placement="Bottom"
                        VerticalOffset="2"
                        IsOpen="{TemplateBinding IsSubmenuOpen}"
                        AllowsTransparency="True"
                        Focusable="False"
                        PopupAnimation="Fade">
                        <Border x:Name="SubMenuBorder"
                            Background="{DynamicResource MenuPopupBrush}"
                            BorderThickness="1">
                            <Border.BorderBrush>
                                <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
                            </Border.BorderBrush>
                            <StackPanel IsItemsHost="true"
                                Margin="2"
                                KeyboardNavigation.TabNavigation="Cycle"
                                KeyboardNavigation.DirectionalNavigation="Cycle" />
                        </Border>
                    </Popup>
                <Grid x:Name="Panel"
                    Width="24"
                    Background="Transparent"
                    HorizontalAlignment="Right"
                    VerticalAlignment="Stretch"
                    d:IsHidden="True">

```

```

<Border Visibility="Hidden"
        x:Name="HighlightBorder"
        BorderThickness="1"
        CornerRadius="2">
    <Border.BorderBrush>
        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1">
            <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                    <GradientStop Color="{DynamicResource BorderLightColor}"
                                  Offset="0.0" />
                    <GradientStop Color="{DynamicResource BorderDarkColor}"
                                  Offset="1.0" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Border.BorderBrush>
    <Border.Background>
        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1">
            <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                    <GradientStop Color="{DynamicResource ControlLightColor}" />
                    <GradientStop Color="{DynamicResource ControlMouseOverColor}"
                                  Offset="1.0" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Border.Background>
</Border>
<Path x:Name="Arrow"
      SnapsToDevicePixels="false"
      HorizontalAlignment="Right"
      VerticalAlignment="Center"
      Margin="0,2,4,0"
      StrokeLineJoin="Round"
      Data="M 0 0 L 4 4 L 8 0 Z">
    <Path.Fill>
        <SolidColorBrush Color="{DynamicResource GlyphColor}" />
    </Path.Fill>
</Path>
</Grid>
</Grid>
<ControlTemplate.Triggers>
    <Trigger Property="IsHighlighted"
              Value="true">
        <Setter TargetName="HighlightBorder"
                  Property="Visibility"
                  Value="Visible" />
    </Trigger>
    <Trigger Property="Is_submenuOpen"
              Value="true">
        <Setter TargetName="HighlightBorder"
                  Property="BorderBrush">
            <Setter.Value>
                <LinearGradientBrush StartPoint="0,0"
                                    EndPoint="0,1">
                    <GradientBrush.GradientStops>
                        <GradientStopCollection>
                            <GradientStop Color="{DynamicResource BorderDarkColor}"
                                          Offset="0.0" />
                            <GradientStop Color="{DynamicResource BorderMediumColor}"
                                          Offset="1.0" />
                        </GradientStopCollection>
                    </GradientBrush.GradientStops>
                </LinearGradientBrush>
            </Setter.Value>
        </Setter>
    </Trigger>
</ControlTemplate.Triggers>

```

```

        </Setter.Value>
    </Setter>
    <Setter Property="Background"
        TargetName="HighlightBorder">
        <Setter.Value>

            <LinearGradientBrush EndPoint="0,1"
                StartPoint="0,0">
                <GradientStop Color="{DynamicResource ControlLightColor}"
                    Offset="0" />
                <GradientStop Color="{DynamicResource ControlPressedColor}"
                    Offset="0.984" />
            </LinearGradientBrush>
        </Setter.Value>
    </Setter>
    </Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<!-- NavWin Menu Item Style --&gt;

&lt;Style x:Key="NavWinSubmenuItem"
    TargetType="{x:Type MenuItem}"&gt;
    &lt;Setter Property="OverridesDefaultStyle"
        Value="true" /&gt;
    &lt;Setter Property="Header"
        Value="{Binding (JournalEntry.Name)}" /&gt;
    &lt;Setter Property="Command"
        Value="NavigationCommands.NavigateJournal" /&gt;
    &lt;Setter Property="CommandTarget"
        Value="{Binding TemplatedParent, RelativeSource={RelativeSource AncestorType={x:Type Menu}}}" /&gt;
    &lt;Setter Property="CommandParameter"
        Value="{Binding RelativeSource={RelativeSource Self}}" /&gt;
    &lt;Setter Property="JournalEntryUnifiedViewConverter.JournalEntryPosition"
        Value="{Binding (JournalEntryUnifiedViewConverter.JournalEntryPosition)}" /&gt;
    &lt;Setter Property="Template"&gt;
        &lt;Setter.Value&gt;
            &lt;ControlTemplate TargetType="{x:Type MenuItem}"&gt;
                &lt;Border Name="Border"
                    BorderThickness="1"&gt;
                    &lt;Grid x:Name="Panel"
                        Background="Transparent"
                        SnapsToDevicePixels="true"
                        Height="35"
                        Width="250"&gt;
                        &lt;Path x:Name="Glyph"
                            SnapsToDevicePixels="false"
                            Margin="7,5"
                            Width="10"
                            Height="10"
                            HorizontalAlignment="Left"
                            StrokeStartLineCap="Triangle"
                            StrokeEndLineCap="Triangle"
                            StrokeThickness="2"&gt;
                            &lt;Path.Stroke&gt;
                                &lt;SolidColorBrush Color="{DynamicResource GlyphColor}" /&gt;
                            &lt;/Path.Stroke&gt;
                        &lt;/Path&gt;
                        &lt;ContentPresenter ContentSource="Header"
                            Margin="24,5,50,5" /&gt;
                    &lt;/Grid&gt;
                &lt;/Border&gt;
                &lt;ControlTemplate.Triggers&gt;
                    &lt;Trigger Value="Current"
                        Property="JournalEntryUnifiedViewConverter.JournalEntryPosition"&gt;
</pre>

```

```

    Property="JournalEntryUnifiedViewConverter.JournalEntryPosition" />
<Setter TargetName="Glyph"
       Property="Data"
       Value="M 0,5 L 2.5,8 L 7,3 " />
</Trigger>
<Trigger Property="IsHighlighted"
         Value="true">
<Setter Property="Background"
        TargetName="Border">
<Setter.Value>
<LinearGradientBrush EndPoint="0.5,1"
                      StartPoint="0.5,0">
<GradientStop Color="Transparent"
               Offset="0" />
<GradientStop Color="{DynamicResource ControlMouseOverColor}"
               Offset="1" />
</LinearGradientBrush>
</Setter.Value>
</Setter>
<Setter Property="BorderBrush"
        TargetName="Border">
<Setter.Value>
<LinearGradientBrush EndPoint="0.5,1"
                      StartPoint="0.5,0">
<GradientStop Color="{DynamicResource BorderMediumColor}"
               Offset="0" />
<GradientStop Color="Transparent"
               Offset="1" />
</LinearGradientBrush>
</Setter.Value>
</Setter>
</Trigger>
<MultiTrigger>
<MultiTrigger.Conditions>
<Condition Property="IsHighlighted"
            Value="true" />
<Condition Value="Forward"
            Property="JournalEntryUnifiedViewConverter.JournalEntryPosition" />
</MultiTrigger.Conditions>
<Setter TargetName="Glyph"
       Property="Data"
       Value="M 3 1 L 7 5 L 3 9 z" />
<Setter TargetName="Glyph"
       Property="Fill">
<Setter.Value>
<SolidColorBrush Color="{StaticResource GlyphColor}" />
</Setter.Value>
</Setter>
<Setter TargetName="Glyph"
       Property="Stroke"
       Value="{x:Null}" />
</MultiTrigger>
<MultiTrigger>
<MultiTrigger.Conditions>
<Condition Property="IsHighlighted"
            Value="true" />
<Condition Value="Back"
            Property="JournalEntryUnifiedViewConverter.JournalEntryPosition" />
</MultiTrigger.Conditions>
<Setter TargetName="Glyph"
       Property="Data"
       Value="M 7 1 L 3 5 L 7 9 z" />
<Setter TargetName="Glyph"
       Property="Fill">
<Setter.Value>
<SolidColorBrush Color="{StaticResource GlyphColor}" />
</Setter.Value>
</Setter>
<Setter TargetName="Glyph"
       Property="Data"
       Value="M 0,5 L 2.5,8 L 7,3 " />
</Trigger>

```

```

        Property="Stroke"
        Value="{x:Null}" />
    </MultiTrigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<!-- Merges Back and Forward Navigation Stacks -->

<JournalEntryUnifiedViewConverter x:Key="JournalEntryUnifiedViewConverter" />

<!-- SimpleStyles: NavigationWindow -->

<Style x:Key="{x:Type NavigationWindow}"
       TargetType="{x:Type NavigationWindow}">
    <Setter Property="SnapsToDevicePixels"
           Value="true" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type NavigationWindow}">
                <DockPanel>
                    <DockPanel.Background>
                        <SolidColorBrush Color="{DynamicResource WindowColor}" />
                    </DockPanel.Background>
                    <Border DockPanel.Dock="Top"
                           Height="30"
                           BorderThickness="1">
                        <Border.BorderBrush>
                            <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
                        </Border.BorderBrush>
                    <Grid>
                        <Grid.Background>

                            <LinearGradientBrush StartPoint="0,0"
                                               EndPoint="0,1">
                                <LinearGradientBrush.GradientStops>
                                    <GradientStopCollection>
                                        <GradientStop Color="{DynamicResource ControlLightColor}"
                                                      Offset="0.0" />
                                        <GradientStop Color="{DynamicResource ControlMediumColor}"
                                                      Offset="1.0" />
                                    </GradientStopCollection>
                                </LinearGradientBrush.GradientStops>
                            </LinearGradientBrush>
                        </Grid.Background>
                    </Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto" />
                    <ColumnDefinition Width="Auto" />
                    <ColumnDefinition Width="16" />
                    <ColumnDefinition Width="*" />
                </Grid.ColumnDefinitions>

                <Menu x:Name="NavMenu"
                      Grid.ColumnSpan="3"
                      Height="20"
                      Margin="1,0,0,0"
                      VerticalAlignment="Center"
                      Style="{StaticResource NavWinMenu}">
                    <MenuItem Style="{StaticResource NavWinHeaderMenuItem}"
                              ItemContainerStyle="{StaticResource NavWinSubmenuItem}"
                              Is_submenuOpen="{Binding (MenuItem.Is_submenuOpen),
Mode=TwoWay, RelativeSource={RelativeSource TemplatedParent}}">
                        <MenuItem.ItemsSource>
                            <MultiBinding Converter="{StaticResource JournalEntryUnifiedViewConverter}">
                                <Binding RelativeSource="{RelativeSource TemplatedParent}"
Path="BackStack" />
                            </MultiBinding>
                        </MenuItem.ItemsSource>
                    </MenuItem>
                </Menu>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

```

        <Binding RelativeSource="{RelativeSource TemplatedParent}"
                  Path="ForwardStack" />
    </MultiBinding>
</MenuItem.ItemsSource>
</MenuItem>
</Menu>

<Path Grid.Column="0"
      SnapsToDevicePixels="false"
      IsHitTestVisible="false"
      Margin="2,1.5,0,1.5"
      Grid.ColumnSpan="3"
      StrokeThickness="1"
      HorizontalAlignment="Left"
      VerticalAlignment="Center"
      Data="M22.5767,21.035 Q27,19.37
            31.424,21.035 A12.5,12.5,0,0,0,53.5,13
            A12.5,12.5,0,0,0,37.765,0.926
            Q27,4.93 16.235,0.926
            A12.5,12.5,0,0,0,0.5,13
            A12.5,12.5,0,0,0,22.5767,21.035 z">
<Path.Stroke>
<SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
</Path.Stroke>
<Path.Fill>

<LinearGradientBrush EndPoint="0,1"
                      StartPoint="0,0">
    <GradientStop Color="{DynamicResource ControlMediumColor}"
                  Offset="0" />

    <GradientStop Color="{DynamicResource ControlDarkColor}"
                  Offset="0.984" />
</LinearGradientBrush>
</Path.Fill>
</Path>
<Button Style="{StaticResource NavWinButtonStyle}"
        Command="NavigationCommands.BrowseBack"
        Content="M 4 0 L 0 4 L 4 8 Z"
        Margin="3,1.5,2,1.5"
        Grid.Column="0" />
<Button Style="{StaticResource NavWinButtonStyle}"
        Command="NavigationCommands.BrowseForward"
        Content="M 4 0 L 0 4 L 4 8 Z"
        Margin="2,1.5,0,1.5"
        Grid.Column="1" />
</Grid>
</Border>
<Grid>
    <AdornerDecorator>
        <Border BorderThickness="1">
            <Border.BorderBrush>
<SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
            </Border.BorderBrush>
            <ContentPresenter x:Name="PART_NavWinCP"
                            ClipToBounds="true" />
        </Border>
    </AdornerDecorator>

    <ResizeGrip x:Name="WindowResizeGrip"
                HorizontalAlignment="Right"
                VerticalAlignment="Bottom"
                Visibility="Collapsed"
                IsTabStop="false" />
</Grid>
</DockPanel>
<ControlTemplate.Triggers>
    <Trigger Property="ResizeMode"
              Value="CanResizeWithGrip">

```

```

        <Setter TargetName="WindowResizeGrip"
               Property="Visibility"
               Value="Visible" />
    </Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

The preceding example uses one or more of the following resources.

```

<Style x:Key="{x:Type ResizeGrip}"
       TargetType="{x:Type ResizeGrip}">
    <Setter Property="OverridesDefaultStyle"
           Value="true" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type ResizeGrip}">
                <Border Background="Transparent"
                        SnapsToDevicePixels="True"
                        Width="16"
                        Height="16">
                    <Rectangle Margin="2">
                        <Rectangle.Fill>
                            <DrawingBrush Viewport="0,0,4,4"
                                         ViewportUnits="Absolute"
                                         Viewbox="0,0,8,8"
                                         ViewboxUnits="Absolute"
                                         TileMode="Tile">
                                <DrawingBrush.Drawing>
                                    <DrawingGroup>
                                        <DrawingGroup.Children>
                                            <GeometryDrawing Brush="#FFE8EDF9"
                                                              Geometry="M 4 4 L 4 8 L
                                                               8 8 L 8 4 z" />
                                        </DrawingGroup.Children>
                                    </DrawingGroup>
                                </DrawingBrush.Drawing>
                            </DrawingBrush>
                        </Rectangle.Fill>
                    </Rectangle>
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF5921A0</Color>

```

```

<Color x:Key="ControlPressedColor">#FF7FFF99</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# PasswordBox Styles and Templates

2 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [PasswordBox](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## PasswordBox Parts

The following table lists the named parts for the [PasswordBox](#) control.

PART	TYPE	DESCRIPTION
PART_ContentHost	FrameworkElement	A visual element that can contain a <a href="#">FrameworkElement</a> . The text of the <a href="#">PasswordBox</a> is displayed in this element.

## PasswordBox States

The following table lists the visual states for the [PasswordBox](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
MouseOver	CommonStates	The mouse pointer is positioned over the control.
Disabled	CommonStates	The control is disabled.
Focused	FocusStates	The control has focus.
Unfocused	FocusStates	The control does not have focus.
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## PasswordBox ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [PasswordBox](#) control.

```

<Style x:Key="{x:Type PasswordBox}"
    TargetType="{x:Type PasswordBox}">
    <Setter Property="SnapsToDevicePixels"
        Value="true" />
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="KeyboardNavigation.TabNavigation"
        Value="None" />
    <Setter Property="FocusVisualStyle"
        Value="{x:Null}" />
    <Setter Property="FontFamily"
        Value="Verdana" />
    <Setter Property="PasswordChar"
        Value="*" />
    <Setter Property="MinWidth"
        Value="120" />
    <Setter Property="MinHeight"
        Value="20" />
    <Setter Property="AllowDrop"
        Value="true" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type PasswordBox}">
                <Border x:Name="Border"
                    CornerRadius="2"
                    Padding="2"
                    BorderThickness="1">
                    <Border.Background>
                        <SolidColorBrush Color="{DynamicResource ControlLightColor}" />
                    </Border.Background>
                    <Border.BorderBrush>
                        <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
                    </Border.BorderBrush>
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="Normal" />
                            <VisualState x:Name="Disabled" />
                            <VisualState x:Name="MouseOver" />
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                    <ScrollViewer x:Name="PART_ContentHost" />
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

The preceding example uses one or more of the following resources.

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>

```

```

<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)



# ProgressBar Styles and Templates

2 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [ProgressBar](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## ProgressBar Parts

The following table lists the named parts for the [ProgressBar](#) control.

PART	TYPE	DESCRIPTION
PART_Indicator	FrameworkElement	The object that indicates progress.
PART_Track	FrameworkElement	The object that defines the path of the progress indicator.
PART_GlowRect	FrameworkElement	An object that embellishes the progress bar.

## ProgressBar States

The following table lists the visual states for the [ProgressBar](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Determinate	CommonStates	<a href="#">ProgressBar</a> reports progress based on the <a href="#">Value</a> property.
Indeterminate	CommonStates	<a href="#">ProgressBar</a> reports generic progress with a repeating pattern.
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> if the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> if the control does not have focus.

## ProgressBar ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [ProgressBar](#) control.

```
<Style x:Key="{x:Type ProgressBar}"
       TargetType="{x:Type ProgressBar}">
  <Setter Property="Template">
```

```

<Setter.Value>
<ControlTemplate TargetType="{x:Type ProgressBar}">
    <Grid MinHeight="14"
          MinWidth="200"
          Background="{TemplateBinding Background}">
        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="CommonStates">
                <VisualState x:Name="Determinate" />
                <VisualState x:Name="Indeterminate">
                    <Storyboard>
                        <ObjectAnimationUsingKeyFrames Duration="00:00:00"
                            Storyboard.TargetName="PART_Indicator"
                            Storyboard.TargetProperty="Background">
                            <DiscreteObjectKeyFrame KeyTime="00:00:00">
                                <DiscreteObjectKeyFrame.Value>
                                    <SolidColorBrush>Transparent</SolidColorBrush>
                                </DiscreteObjectKeyFrame.Value>
                            </DiscreteObjectKeyFrame>
                        </ObjectAnimationUsingKeyFrames>

                        </Storyboard>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateManager.VisualStateGroups>
            <Border x:Name="PART_Track"
                    CornerRadius="2"
                    BorderThickness="1">
                <Border.BorderBrush>
                    <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
                </Border.BorderBrush>
            </Border>
            <Border x:Name="PART_Indicator"
                    CornerRadius="2"
                    BorderThickness="1"
                    HorizontalAlignment="Left"
                    Background="{TemplateBinding Foreground}"
                    Margin="0,-1,0,1">
                <Border.BorderBrush>
                    <LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1">
                        <GradientBrush.GradientStops>
                            <GradientStopCollection>
                                <GradientStop Color="{DynamicResource BorderLightColor}"
                                              Offset="0.0" />
                                <GradientStop Color="{DynamicResource BorderMediumColor}"
                                              Offset="1.0" />
                            </GradientStopCollection>
                        </GradientBrush.GradientStops>
                    </LinearGradientBrush>
                </Border.BorderBrush>
                <Grid ClipToBounds="True"
                      x:Name="Animation">
                    <Rectangle x:Name="PART_GlowRect"
                               Width="100"
                               HorizontalAlignment="Left"
                               Fill="{StaticResource ProgressBarIndicatorAnimatedFill}"
                               Margin="-100,0,0,0" />
                </Grid>
            </Border>
        </Grid>
    </ControlTemplate>
</Setter.Value>
</Setter>
<Setter Property="Background">
    <Setter.Value>
        <LinearGradientBrush EndPoint="0,1"
                            StartPoint="0,0">
            <GradientStop Color="{DynamicResource ControlLightColor}">

```

```

        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="1" />
</LinearGradientBrush>
</Setter.Value>
</Setter>
<Setter Property="Foreground">
    <Setter.Value>
        <LinearGradientBrush EndPoint="0.5,1"
            StartPoint="0.5,0">
            <GradientStop Color="{DynamicResource ControlMediumColor}"
                Offset="0" />
            <GradientStop Color="{DynamicResource ControlDarkColor}"
                Offset="1" />
        </LinearGradientBrush>
    </Setter.Value>
</Setter>
</Style>

```

The preceding example uses one or more of the following resources.

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">

```

```
<GradientStop Color="{DynamicResource ControlLightColor}"
    Offset="0" />
<GradientStop Color="{DynamicResource ControlMediumColor}"
    Offset="0.5" />
<GradientStop Color="{DynamicResource ControlLightColor}"
    Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# RadioButton Styles and Templates

2 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [RadioButton](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## RadioButton Parts

The [RadioButton](#) control does not have any named parts.

## RadioButton States

The following table lists the visual states for the [RadioButton](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
MouseOver	CommonStates	The mouse pointer is positioned over the control.
Pressed	CommonStates	The control is pressed.
Disabled	CommonStates	The control is disabled.
Focused	FocusStates	The control has focus.
Unfocused	FocusStates	The control does not have focus.
Checked	CheckStates	<code>IsChecked</code> is <code>true</code> .
Unchecked	CheckStates	<code>IsChecked</code> is <code>false</code> .
Indeterminate	CheckStates	<code>IsThreeState</code> is <code>true</code> , and <code>IsChecked</code> is <code>null</code> .
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> if the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> if the control does not have focus.

## RadioButton ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [RadioButton](#) control.

```
<Style x:Key="{x:Type RadioButton}"
    TargetType="{x:Type RadioButton}">
    <Setter Property="SnapsToDevicePixels"
        Value="true" />
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="FocusVisualStyle"
        Value="{DynamicResource RadioButtonFocusVisual}" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type RadioButton}">
                <BulletDecorator Background="Transparent">
                    <BulletDecorator.Bullet>
                        <Grid Width="13"
                            Height="13">
                            <Ellipse x:Name="Border"
                                StrokeThickness="1">
                                <Ellipse.Stroke>
                                    <LinearGradientBrush EndPoint="0.5,1"
                                        StartPoint="0.5,0">
                                        <GradientStop Color="{DynamicResource BorderLightColor}"
                                            Offset="0" />
                                        <GradientStop Color="{DynamicResource BorderDarkColor}"
                                            Offset="1" />
                                    </LinearGradientBrush>
                                </Ellipse.Stroke>
                            <Ellipse.Fill>
                                <LinearGradientBrush StartPoint="0,0"
                                    EndPoint="0,1">
                                    <LinearGradientBrush.GradientStops>
                                        <GradientStopCollection>
                                            <GradientStop Color="{DynamicResource ControlLightColor}" />
                                            <GradientStop Color="{DynamicResource ControlMediumColor}"
                                                Offset="1.0" />
                                        </GradientStopCollection>
                                    </LinearGradientBrush.GradientStops>
                                </LinearGradientBrush>
                            </Ellipse.Fill>
                        </Grid>
                    </BulletDecorator.Bullet>
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="Normal" />
                            <VisualState x:Name="MouseOver">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                        Storyboard.TargetProperty="(Shape.Fill)">
                                        <GradientBrush.GradientStops>
                                            <GradientStop>
                                                <EasingColorKeyFrame KeyTime="0"
                                                    Value="{StaticResource ControlMouseOverColor}" />
                                            </GradientStop>
                                        </GradientBrush.GradientStops>
                                    </ColorAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="Pressed">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                        Storyboard.TargetProperty="(Shape.Fill)">
                                        <GradientBrush.GradientStops>
                                            <GradientStop>
                                                <EasingColorKeyFrame KeyTime="0"
                                                    Value="{StaticResource ControlPressedColor}" />
                                            </GradientStop>
                                        </GradientBrush.GradientStops>
                                    </ColorAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                </BulletDecorator>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

```

        <EasingColorKeyFrame KeyTime="0"
            Value="{StaticResource ControlPressedColor}" />
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Disabled">
    <Storyboard>
        <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
            Storyboard.TargetProperty="(Shape.Fill).{GradientBrush.GradientStops}[1].(GradientStop.Color)">
            <EasingColorKeyFrame KeyTime="0"
                Value="{StaticResource ControlLightColor}" />
        </ColorAnimationUsingKeyFrames>
        <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
            Storyboard.TargetProperty="(Shape.Stroke).{GradientBrush.GradientStops}[1].(GradientStop.Color)">
            <EasingColorKeyFrame KeyTime="0"
                Value="#40000000" />
        </ColorAnimationUsingKeyFrames>
        <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
            Storyboard.TargetProperty="(Shape.Stroke).{GradientBrush.GradientStops}[0].(GradientStop.Color)">
            <EasingColorKeyFrame KeyTime="0"
                Value="#40000000" />
        </ColorAnimationUsingKeyFrames>
    </Storyboard>
</VisualState>
</VisualStateGroup>
<VisualStateGroup x:Name="CheckStates">
    <VisualState x:Name="Checked">
        <Storyboard>
            <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)">
                Storyboard.TargetName="CheckMark">
                <DiscreteObjectKeyFrame KeyTime="0"
                    Value="{x:Static Visibility.Visible}" />
            </ObjectAnimationUsingKeyFrames>
        </Storyboard>
    </VisualState>
    <VisualState x:Name="Unchecked" />
    <VisualState x:Name="Indeterminate" />
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<ContentPresenter Margin="4,0,0,0"
    VerticalAlignment="Center"
    HorizontalAlignment="Left"
    RecognizesAccessKey="True" />
</BulletDecorator>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

The preceding example uses one or more of the following resources.

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>

```

```

<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)

- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# RepeatButton Styles and Templates

2 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [RepeatButton](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## RepeatButton Parts

The [RepeatButton](#) control does not have any named parts.

## RepeatButton States

The following table lists the visual states for the [RepeatButton](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
MouseOver	CommonStates	The mouse pointer is positioned over the control.
Pressed	CommonStates	The control is pressed.
Disabled	CommonStates	The control is disabled.
Focused	FocusStates	The control has focus.
Unfocused	FocusStates	The control does not have focus.
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## RepeatButton ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [RepeatButton](#) control.

```
<Style x:Key="ScrollBarLineButton"
    TargetType="{x:Type RepeatButton}">
    <Setter Property="SnapsToDevicePixels"
        Value="True" />
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="Focusable"
```

```

        <Setter Property="Focusable"
            Value="false" />
<Setter Property="Template">
    <Setter.Value>
        <ControlTemplate TargetType="{x:Type RepeatButton}">
            <Border x:Name="Border"
                Margin="1"
                CornerRadius="2"
                BorderThickness="1">
                <Border.BorderBrush>
                    <LinearGradientBrush StartPoint="0,0"
                        EndPoint="0,1">
                        <LinearGradientBrush.GradientStops>
                            <GradientStopCollection>
                                <GradientStop Color="{DynamicResource BorderMediumColor}"
                                    Offset="0.0" />
                                <GradientStop Color="{DynamicResource BorderDarkColor}"
                                    Offset="1.0" />
                            </GradientStopCollection>
                        </LinearGradientBrush.GradientStops>
                    </LinearGradientBrush>
                </Border.BorderBrush>
                <Border.Background>
                    <LinearGradientBrush StartPoint="0,0"
                        EndPoint="0,1">
                        <LinearGradientBrush.GradientStops>
                            <GradientStopCollection>
                                <GradientStop Color="{DynamicResource ControlLightColor}"/>
                                <GradientStop Color="{DynamicResource ControlMediumColor}"
                                    Offset="1.0" />
                            </GradientStopCollection>
                        </LinearGradientBrush.GradientStops>
                    </LinearGradientBrush>
                </Border.Background>
                <VisualStateManager.VisualStateGroups>
                    <VisualStateGroup x:Name="CommonStates">
                        <VisualState x:Name="Normal" />
                        <VisualState x:Name="MouseOver" />
                        <VisualState x:Name="Pressed">
                            <Storyboard>
                                <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                    Storyboard.TargetProperty="(Panel.Background)."
                                    (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                                    <EasingColorKeyFrame KeyTime="0"
                                        Value="{StaticResource ControlPressedColor}" />
                                </ColorAnimationUsingKeyFrames>
                            </Storyboard>
                        </VisualState>
                        <VisualState x:Name="Disabled">
                            <Storyboard>
                                <ColorAnimationUsingKeyFrames Storyboard.TargetName="Arrow"
                                    Storyboard.TargetProperty="(Shape.Fill)."
                                    (SolidColorBrush.Color)">
                                    <EasingColorKeyFrame KeyTime="0"
                                        Value="{StaticResource DisabledForegroundColor}" />
                                </ColorAnimationUsingKeyFrames>
                            </Storyboard>
                        </VisualState>
                    </VisualStateGroup>
                </VisualStateManager.VisualStateGroups>
                <Path x:Name="Arrow"
                    HorizontalAlignment="Center"
                    VerticalAlignment="Center"
                    Data="{Binding Content,
                    RelativeSource={RelativeSource TemplatedParent}}" >
                    <Path.Fill>
                        <SolidColorBrush Color="{DynamicResource GlyphColor}" />
                    </Path.Fill>
                </Path>
            </Border>
        </ControlTemplate>
    </Setter.Value>
</Setter>

```

```

        </Border>
    </ControlTemplate>
<Setter.Value>
</Setter>
</Style>

```

The preceding example uses one or more of the following resources.

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

```
<GradientStop Color="#000000FF"
    Offset="0" />
<GradientStop Color="#600000FF"
    Offset="0.4" />
<GradientStop Color="#600000FF"
    Offset="0.6" />
<GradientStop Color="#000000FF"
    Offset="1" />
</GradientStopCollection>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# ScrollBar Styles and Templates

3 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [ScrollBar](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## ScrollBar Parts

The following table lists the named parts for the [ScrollBar](#) control.

PART	TYPE	DESCRIPTION
PART_Track	Track	The container for the element that indicates the position of the <a href="#">ScrollBar</a> .

## ScrollBar States

The following table lists the visual states for the [ScrollBar](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
MouseOver	CommonStates	The mouse pointer is positioned over the control.
Disabled	CommonStates	The control is disabled.
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> and the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> and the control does not have focus.

## ScrollBar ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [ScrollBar](#) control.

```
<Style x:Key="ScrollBarLineButton"
    TargetType="{x:Type RepeatButton}>
    <Setter Property="SnapsToDevicePixels"
        Value="True" />
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="Focusable"
        Value="false" />
    <Setter Property="Template">
```

```

<Setter.Value>
    <ControlTemplate TargetType="{x:Type RepeatButton}">
        <Border x:Name="Border"
            Margin="1"
            CornerRadius="2"
            BorderThickness="1">
            <Border.BorderBrush>
                <LinearGradientBrush StartPoint="0,0"
                    EndPoint="0,1">
                    <LinearGradientBrush.GradientStops>
                        <GradientStopCollection>
                            <GradientStop Color="{DynamicResource BorderMediumColor}"
                                Offset="0.0" />
                            <GradientStop Color="{DynamicResource BorderDarkColor}"
                                Offset="1.0" />
                        </GradientStopCollection>
                    </LinearGradientBrush.GradientStops>
                </LinearGradientBrush>
            </Border.BorderBrush>
            <Border.Background>
                <LinearGradientBrush StartPoint="0,0"
                    EndPoint="0,1">
                    <LinearGradientBrush.GradientStops>
                        <GradientStopCollection>
                            <GradientStop Color="{DynamicResource ControlLightColor}"/>
                            <GradientStop Color="{DynamicResource ControlMediumColor}"
                                Offset="1.0" />
                        </GradientStopCollection>
                    </LinearGradientBrush.GradientStops>
                </LinearGradientBrush>
            </Border.Background>
        </VisualStateManager.VisualStateGroups>
        <VisualStateGroup x:Name="CommonStates">
            <VisualState x:Name="Normal" />
            <VisualState x:Name="MouseOver" />
            <VisualState x:Name="Pressed">
                <Storyboard>
                    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                        Storyboard.TargetProperty="(Panel.Background).(
                            GradientBrush.GradientStops)[1].(GradientStop.Color)">
                        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource ControlPressedColor}" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>
            <VisualState x:Name="Disabled">
                <Storyboard>
                    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Arrow"
                        Storyboard.TargetProperty="(Shape.Fill).(
                            SolidColorBrush.Color)">
                        <EasingColorKeyFrame KeyTime="0"
                            Value="{StaticResource DisabledForegroundColor}" />
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>
        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>
    <Path x:Name="Arrow"
        HorizontalAlignment="Center"
        VerticalAlignment="Center"
        Data="{Binding Content,
        RelativeSource={RelativeSource TemplatedParent}}" >
        <Path.Fill>
            <SolidColorBrush Color="{DynamicResource GlyphColor}"/>
        </Path.Fill>
    </Path>
</Border>
</ControlTemplate>
</Setter.Value>

```

```

        </Setter>
    </Style>

<Style x:Key="ScrollBarPageButton"
       TargetType="{x:Type RepeatButton}">
    <Setter Property="SnapsToDevicePixels"
           Value="True" />
    <Setter Property="OverridesDefaultStyle"
           Value="true" />
    <Setter Property="IsTabStop"
           Value="false" />
    <Setter Property="Focusable"
           Value="false" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type RepeatButton}">
                <Border Background="Transparent" />
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

<Style x:Key="ScrollBarThumb"
       TargetType="{x:Type Thumb}">
    <Setter Property="SnapsToDevicePixels"
           Value="True" />
    <Setter Property="OverridesDefaultStyle"
           Value="true" />
    <Setter Property="IsTabStop"
           Value="false" />
    <Setter Property="Focusable"
           Value="false" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Thumb}">
                <Border CornerRadius="2"
                       Background="{TemplateBinding Background}"
                       BorderBrush="{TemplateBinding BorderBrush}"
                       BorderThickness="1" />
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

<ControlTemplate x:Key="VerticalScrollBar"
                 TargetType="{x:Type ScrollBar}">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition MaxHeight="18" />
            <RowDefinition Height="0.0001*" />
            <RowDefinition MaxHeight="18" />
        </Grid.RowDefinitions>
        <Border Grid.RowSpan="3"
               CornerRadius="2"
               Background="#F0F0F0" />
        <RepeatButton Grid.Row="0"
                     Style="{StaticResource ScrollBarLineButton}"
                     Height="18"
                     Command="ScrollBar.LineUpCommand"
                     Content="M 0 4 L 8 4 L 4 0 Z" />
        <Track x:Name="PART_Track"
               Grid.Row="1"
               IsDirectionReversed="true">
            <Track.DecreaseRepeatButton>
                <RepeatButton Style="{StaticResource ScrollBarPageButton}"
                             Command="ScrollBar.PageUpCommand" />
            </Track.DecreaseRepeatButton>
            <Track.Thumb>
                <Thumb Style="{StaticResource ScrollBarThumb}" />
            </Track.Thumb>
        </Track>
    </Grid>
</ControlTemplate>

```

```

        Margin="1,0,1,0">
    <Thumb.BorderBrush>

        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="1,0">
            <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                    <GradientStop Color="{DynamicResource BorderLightColor}"
                                  Offset="0.0" />
                    <GradientStop Color="{DynamicResource BorderDarkColor}"
                                  Offset="1.0" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Thumb.BorderBrush>
    <Thumb.Background>

        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="1,0">
            <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                    <GradientStop Color="{DynamicResource ControlLightColor}"
                                  Offset="0.0" />
                    <GradientStop Color="{DynamicResource ControlMediumColor}"
                                  Offset="1.0" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Thumb.Background>
</Thumb>
</Track.Thumb>
<Track.IncreaseRepeatButton>
    <RepeatButton Style="{StaticResource ScrollBarPageButton}"
                  Command="ScrollBar.PageDownCommand" />
</Track.IncreaseRepeatButton>
</Track>
<RepeatButton Grid.Row="2"
              Style="{StaticResource ScrollBarLineButton}"
              Height="18"
              Command="ScrollBar.LineDownCommand"
              Content="M 0 0 L 4 4 L 8 0 Z" />
</Grid>
</ControlTemplate>

<ControlTemplate x:Key="HorizontalScrollBar"
                 TargetType="{x:Type ScrollBar}">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition MaxWidth="18" />
            <ColumnDefinition Width="0.0001*" />
            <ColumnDefinition MaxWidth="18" />
        </Grid.ColumnDefinitions>
        <Border Grid.ColumnSpan="3"
               CornerRadius="2"
               Background="#F0F0F0" />
        <RepeatButton Grid.Column="0"
                     Style="{StaticResource ScrollBarLineButton}"
                     Width="18"
                     Command="ScrollBar.LineLeftCommand"
                     Content="M 4 0 L 4 8 L 0 4 Z" />
        <Track x:Name="PART_Track"
               Grid.Column="1"
               IsDirectionReversed="False">
            <Track.DecreaseRepeatButton>
                <RepeatButton Style="{StaticResource ScrollBarPageButton}"
                              Command="ScrollBar.PageLeftCommand" />
            </Track.DecreaseRepeatButton>

```

```

    </ControlTemplate>
<Track.Thumb>
    <Thumb Style="{StaticResource ScrollBarThumb}"
        Margin="0,1,0,1">

        <Thumb.BorderBrush>

            <LinearGradientBrush StartPoint="0,0"
                EndPoint="1,0">
                <LinearGradientBrush.GradientStops>
                    <GradientStopCollection>
                        <GradientStop Color="{DynamicResource BorderLightColor}"
                            Offset="0.0" />
                        <GradientStop Color="{DynamicResource BorderDarkColor}"
                            Offset="1.0" />
                    </GradientStopCollection>
                </LinearGradientBrush.GradientStops>
            </LinearGradientBrush>

        </Thumb.BorderBrush>
        <Thumb.Background>

            <LinearGradientBrush StartPoint="0,0"
                EndPoint="0,1">
                <LinearGradientBrush.GradientStops>
                    <GradientStopCollection>
                        <GradientStop Color="{DynamicResource ControlLightColor}"
                            Offset="0.0" />
                        <GradientStop Color="{DynamicResource ControlMediumColor}"
                            Offset="1.0" />
                    </GradientStopCollection>
                </LinearGradientBrush.GradientStops>
            </LinearGradientBrush>

        </Thumb.Background>
    </Thumb>
</Track.Thumb>
<Track.IncreaseRepeatButton>
    <RepeatButton Style="{StaticResource ScrollBarPageButton}"
        Command="ScrollBar.PageRightCommand" />
</Track.IncreaseRepeatButton>
</Track>
<RepeatButton Grid.Column="2"
    Style="{StaticResource ScrollBarLineButton}"
    Width="18"
    Command="ScrollBar.LineRightCommand"
    Content="M 0 0 L 4 4 L 0 8 Z" />
</Grid>
</ControlTemplate>

<Style x:Key="{x:Type ScrollBar}"
    TargetType="{x:Type ScrollBar}">
    <Setter Property="SnapsToDevicePixels"
        Value="True" />
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Style.Triggers>
        <Trigger Property="Orientation"
            Value="Horizontal">
            <Setter Property="Width"
                Value="Auto" />
            <Setter Property="Height"
                Value="18" />
            <Setter Property="Template"
                Value="{StaticResource HorizontalScrollBar}" />
        </Trigger>
        <Trigger Property="Orientation"
            Value="Vertical">
            <Setter Property="Width"
                Value="18" />
            <Setter Property="Height"
                Value="Auto" />
        </Trigger>
    </Style.Triggers>
</Style>

```

```

        value="18" />
    <Setter Property="Height"
        Value="Auto" />
    <Setter Property="Template"
        Value="{StaticResource VerticalScrollBar}" />
</Trigger>
</Style.Triggers>
</Style>

```

The preceding example uses one or more of the following resources.

```

<!--Control colors-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">>sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">

```

```
    EndPoint="1,0">
<LinearGradientBrush.GradientStops>
  <GradientStopCollection>
    <GradientStop Color="#000000FF"
      Offset="0" />
    <GradientStop Color="#600000FF"
      Offset="0.4" />
    <GradientStop Color="#600000FF"
      Offset="0.6" />
    <GradientStop Color="#000000FF"
      Offset="1" />
  </GradientStopCollection>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# ScrollViewer Styles and Templates

2 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [ScrollViewer](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## ScrollViewer Parts

The following table lists the named parts for the [ScrollViewer](#) control.

PART	TYPE	DESCRIPTION
PART_ScrollContentPresenter	ScrollContentPresenter	The placeholder for content in the <a href="#">ScrollViewer</a> .
PART_HorizontalScrollBar	ScrollBar	The <a href="#">ScrollBar</a> used to scroll the content horizontally.
PART_VerticalScrollBar	ScrollBar	The <a href="#">ScrollBar</a> used to scroll the content vertically.

## ScrollViewer States

The following table lists the visual states for the [ScrollViewer](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## ScrollViewer ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [ScrollViewer](#) control.

```

<Style x:Key="LeftScrollViewer"
    TargetType="{x:Type ScrollViewer}">
    <Setter Property="OverridesDefaultStyle"
        Value="True" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type ScrollViewer}">
                <Grid>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="Auto" />
                        <ColumnDefinition />
                    </Grid.ColumnDefinitions>
                    <Grid.RowDefinitions>
                        <RowDefinition />
                        <RowDefinition Height="Auto" />
                    </Grid.RowDefinitions>
                    <Border Grid.Column="1"
                        BorderThickness="0,1,1,1">
                        <Border.BorderBrush>
                            <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
                        </Border.BorderBrush>
                        <ScrollContentPresenter />
                    </Border>
                    <ScrollBar x:Name="PART_VerticalScrollBar"
                        Value="{TemplateBinding VerticalOffset}"
                        Maximum="{TemplateBinding ScrollableHeight}"
                        ViewportSize="{TemplateBinding ViewportHeight}"
                        Visibility="{TemplateBinding ComputedVerticalScrollBarVisibility}"/>
                    <ScrollBar x:Name="PART_HorizontalScrollBar"
                        Orientation="Horizontal"
                        Grid.Row="1"
                        Grid.Column="1"
                        Value="{TemplateBinding HorizontalOffset}"
                        Maximum="{TemplateBinding ScrollableWidth}"
                        ViewportSize="{TemplateBinding ViewportWidth}"
                        Visibility="{TemplateBinding ComputedHorizontalScrollBarVisibility}"/>
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

The preceding example uses one or more of the following resources.

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>

```

```

<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# Slider Styles and Templates

4 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [Slider](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## Slider Parts

The following table lists the named parts for the [Slider](#) control.

PART	TYPE	DESCRIPTION
PART_Track	Track	The container for the element that indicates the position of the <a href="#">Slider</a> .
PART_SelectionRange	FrameworkElement	The element that displays a selection range along the <a href="#">Slider</a> . The selection range is visible only if the <a href="#">IsSelectionRangeEnabled</a> property is <code>true</code> .

## Slider States

The following table lists the visual states for the [Slider](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
MouseOver	CommonStates	The mouse pointer is positioned over the control.
Disabled	CommonStates	The control is disabled.
Focused	FocusStates	The control has focus.
Unfocused	FocusStates	The control does not have focus.
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## Slider ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [Slider](#) control.

```
<Style x:Key="SliderButtonStyle"
    TargetType="{x:Type RepeatButton}">
    <Setter Property="SnapsToDevicePixels"
        Value="true" />
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="IsTabStop"
        Value="false" />
    <Setter Property="Focusable"
        Value="false" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type RepeatButton}">
                <Border Background="Transparent" />
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

<Style x:Key="SliderThumbStyle"
    TargetType="{x:Type Thumb}">
    <Setter Property="SnapsToDevicePixels"
        Value="true" />
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="Height"
        Value="14" />
    <Setter Property="Width"
        Value="14" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Thumb}">
                <Ellipse x:Name="Ellipse"
                    StrokeThickness="1">
                    <Ellipse.Stroke>
                        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1">
                            <LinearGradientBrush.GradientStops>
                                <GradientStopCollection>
                                    <GradientStop Color="{DynamicResource BorderLightColor}"
                                        Offset="0.0" />
                                    <GradientStop Color="{DynamicResource BorderDarkColor}"
                                        Offset="1.0" />
                                </GradientStopCollection>
                            </LinearGradientBrush.GradientStops>
                        </LinearGradientBrush>
                    </Ellipse.Stroke>
                    <Ellipse.Fill>
                        <LinearGradientBrush EndPoint="0.5,1"
                            StartPoint="0.5,0">
                            <GradientStop Color="{DynamicResource ControlMediumColor}"
                                Offset="1" />
                            <GradientStop Color="{DynamicResource ControlLightColor}" />
                        </LinearGradientBrush>
                    </Ellipse.Fill>
                </Ellipse>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
    <VisualStateManager.VisualStateGroups>
        <VisualStateGroup x:Name="CommonStates">
            <VisualState x:Name="Normal" />
            <VisualState x:Name="MouseOver">
                <Storyboard>
                    <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Shape.Fill)."
                        (GradientBrush.GradientStops)[0].(GradientStop.Color)">
                        <Storyboard.TargetName="Ellipse">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource ControlMouseOverColor}" />
                        </Storyboard.TargetName>
                    </ColorAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>
        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>
</Style>
```

```

        </Storyboard>
    </VisualState>
    <VisualState x:Name="Pressed">
        <Storyboard>
            <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Shape.Fill).(
                GradientBrush.GradientStops)[0].(GradientStop.Color)" Storyboard.TargetName="Ellipse">
                <EasingColorKeyFrame KeyTime="0" Value="{StaticResource ControlPressedColor}" />
            </ColorAnimationUsingKeyFrames>
        </Storyboard>
    </VisualState>
    <VisualState x:Name="Disabled">
        <Storyboard>
            <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Shape.Fill).(
                GradientBrush.GradientStops)[0].(GradientStop.Color)" Storyboard.TargetName="Ellipse">
                <EasingColorKeyFrame KeyTime="0" Value="{StaticResource DisabledControlDarkColor}" />
            </ColorAnimationUsingKeyFrames>
        </Storyboard>
    </VisualState>
</VisualStateManager.VisualStateGroups>
</Ellipse>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<!--Template when the orientation of the Slider is Horizontal.--&gt;
&lt;ControlTemplate x:Key="HorizontalSlider"
    TargetType="{x:Type Slider}"&gt;
    &lt;Grid&gt;
        &lt;Grid.RowDefinitions&gt;
            &lt;RowDefinition Height="Auto" /&gt;
            &lt;RowDefinition Height="Auto"
                MinHeight="{TemplateBinding MinHeight}" /&gt;
            &lt;RowDefinition Height="Auto" /&gt;
        &lt;/Grid.RowDefinitions&gt;
        &lt;TickBar x:Name="TopTick"
            SnapsToDevicePixels="True"
            Placement="Top"
            Height="4"
            Visibility="Collapsed"&gt;
            &lt;TickBar.Fill&gt;
                &lt;SolidColorBrush Color="{DynamicResource GlyphColor}" /&gt;
            &lt;/TickBar.Fill&gt;
        &lt;/TickBar&gt;
        &lt;Border x:Name="TrackBackground"
            Margin="0"
            CornerRadius="2"
            Height="4"
            Grid.Row="1"
            BorderThickness="1"&gt;
            &lt;Border.BorderBrush&gt;
                &lt;LinearGradientBrush StartPoint="0,0"
                    EndPoint="0,1"&gt;
                    &lt;LinearGradientBrush.GradientStops&gt;
                        &lt;GradientStopCollection&gt;
                            &lt;GradientStop Color="{DynamicResource BorderLightColor}"
                                Offset="0.0" /&gt;
                            &lt;GradientStop Color="{DynamicResource BorderDarkColor}"
                                Offset="1.0" /&gt;
                        &lt;/GradientStopCollection&gt;
                    &lt;/LinearGradientBrush.GradientStops&gt;
                &lt;/LinearGradientBrush&gt;
            &lt;/Border.BorderBrush&gt;
            &lt;Border.Background&gt;
</pre>

```

```

<Border Padding="0" BorderThickness="1" Background="White" CornerRadius="2" Opacity="0.85" x:Name="Border">
    <Border.Background>
        <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
            <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                    <GradientStop Color="{DynamicResource ControlLightColor}" Offset="0.0" />
                    <GradientStop Color="{DynamicResource SliderTrackDarkColor}" Offset="1.0" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Border.Background>
</Border>
<Track Grid.Row="1" x:Name="PART_Track">
    <Track.DecreaseRepeatButton>
        <RepeatButton Style="{StaticResource SliderButtonStyle}" Command="Slider.DecreaseLarge" />
    </Track.DecreaseRepeatButton>
    <Track.Thumb>
        <Thumb Style="{StaticResource SliderThumbStyle}" />
    </Track.Thumb>
    <Track.IncreaseRepeatButton>
        <RepeatButton Style="{StaticResource SliderButtonStyle}" Command="Slider.IncreaseLarge" />
    </Track.IncreaseRepeatButton>
</Track>
<TickBar x:Name="BottomTick" SnapsToDevicePixels="True" Grid.Row="2" Fill="{TemplateBinding Foreground}" Placement="Bottom" Height="4" Visibility="Collapsed" />
</Grid>
<ControlTemplate.Triggers>
    <Trigger Property="TickPlacement" Value="TopLeft">
        <Setter TargetName="TopTick" Property="Visibility" Value="Visible" />
    </Trigger>
    <Trigger Property="TickPlacement" Value="BottomRight">
        <Setter TargetName="BottomTick" Property="Visibility" Value="Visible" />
    </Trigger>
    <Trigger Property="TickPlacement" Value="Both">
        <Setter TargetName="TopTick" Property="Visibility" Value="Visible" />
        <Setter TargetName="BottomTick" Property="Visibility" Value="Visible" />
    </Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>

<!--Template when the orientation of the Slider is Vertical.-->
<ControlTemplate x:Key="VerticalSlider" TargetType="{x:Type Slider}">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="Auto" MinWidth="{TemplateBinding MinWidth}" />
            <ColumnDefinition Width="Auto" />
        </Grid.ColumnDefinitions>
        <Border x:Name="Border" BorderThickness="1" CornerRadius="2" Opacity="0.85" Background="White" Padding="0" BorderBrush="Black" Grid.Column="1" Grid.Row="1" Grid.RowSpan="2" Grid.ZIndex="1" ClipToBounds="True" OpacityMask="Black" />
        <Track Grid.Column="1" Grid.Row="2" x:Name="PART_Track">
            <Track.DecreaseRepeatButton>
                <RepeatButton Style="{StaticResource SliderButtonStyle}" Command="Slider.DecreaseLarge" />
            </Track.DecreaseRepeatButton>
            <Track.Thumb>
                <Thumb Style="{StaticResource SliderThumbStyle}" />
            </Track.Thumb>
            <Track.IncreaseRepeatButton>
                <RepeatButton Style="{StaticResource SliderButtonStyle}" Command="Slider.IncreaseLarge" />
            </Track.IncreaseRepeatButton>
        </Track>
        <TickBar x:Name="BottomTick" SnapsToDevicePixels="True" Grid.Column="1" Grid.Row="2" Grid.RowSpan="2" Fill="{TemplateBinding Foreground}" Placement="Bottom" Height="4" Visibility="Collapsed" />
    </Grid>
    <ControlTemplate.Triggers>
        <Trigger Property="TickPlacement" Value="TopLeft">
            <Setter TargetName="TopTick" Property="Visibility" Value="Visible" />
        </Trigger>
        <Trigger Property="TickPlacement" Value="BottomRight">
            <Setter TargetName="BottomTick" Property="Visibility" Value="Visible" />
        </Trigger>
        <Trigger Property="TickPlacement" Value="Both">
            <Setter TargetName="TopTick" Property="Visibility" Value="Visible" />
            <Setter TargetName="BottomTick" Property="Visibility" Value="Visible" />
        </Trigger>
    </ControlTemplate.Triggers>
</ControlTemplate>

```

```

        <ColumnDefinition Width="Auto" />
    </Grid.ColumnDefinitions>
    <TickBar x:Name="TopTick"
        SnapsToDevicePixels="True"
        Placement="Left"
        Width="4"
        Visibility="Collapsed">
        <TickBar.Fill>
            <SolidColorBrush Color="{DynamicResource GlyphColor}" />
        </TickBar.Fill>
    </TickBar>

    <Border x:Name="TrackBackground"
        Margin="0"
        CornerRadius="2"
        Width="4"
        Grid.Column="1"
        BorderThickness="1">
        <Border.BorderBrush>
            <LinearGradientBrush StartPoint="0,0"
                EndPoint="1,0">
                <LinearGradientBrush.GradientStops>
                    <GradientStopCollection>
                        <GradientStop Color="{DynamicResource BorderLightColor}"
                            Offset="0.0" />
                        <GradientStop Color="{DynamicResource BorderDarkColor}"
                            Offset="1.0" />
                    </GradientStopCollection>
                </LinearGradientBrush.GradientStops>
            </LinearGradientBrush>
        </Border.BorderBrush>
        <Border.Background>
            <LinearGradientBrush EndPoint="1,0"
                StartPoint="0.25,0">
                <GradientStop Color="{DynamicResource ControlLightColor}"
                    Offset="0" />
                <GradientStop Color="{DynamicResource SliderTrackDarkColor}"
                    Offset="1" />
            </LinearGradientBrush>
        </Border.Background>
    </Border>
    <Track Grid.Column="1"
        x:Name="PART_Track">
        <Track.DecreaseRepeatButton>
            <RepeatButton Style="{StaticResource SliderButtonStyle}"
                Command="Slider.DecreaseLarge" />
        </Track.DecreaseRepeatButton>
        <Track.Thumb>
            <Thumb Style="{StaticResource SliderThumbStyle}" />
        </Track.Thumb>
        <Track.IncreaseRepeatButton>
            <RepeatButton Style="{StaticResource SliderButtonStyle}"
                Command="Slider.IncreaseLarge" />
        </Track.IncreaseRepeatButton>
    </Track>
    <TickBar x:Name="BottomTick"
        SnapsToDevicePixels="True"
        Grid.Column="2"
        Fill="{TemplateBinding Foreground}"
        Placement="Right"
        Width="4"
        Visibility="Collapsed" />
</Grid>
<ControlTemplate.Triggers>
    <Trigger Property="TickPlacement"
        Value="TopLeft">
        <Setter TargetName="TopTick"
            Property="Visibility"

```

```

        Value="Visible" />
    </Trigger>
    <Trigger Property="TickPlacement"
        Value="BottomRight">
        <Setter TargetName="BottomTick"
            Property="Visibility"
            Value="Visible" />
    </Trigger>
    <Trigger Property="TickPlacement"
        Value="Both">
        <Setter TargetName="TopTick"
            Property="Visibility"
            Value="Visible" />
        <Setter TargetName="BottomTick"
            Property="Visibility"
            Value="Visible" />
    </Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>

<Style TargetType="{x:Type Slider}">
    <Setter Property="SnapsToDevicePixels"
        Value="true" />
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Style.Triggers>
        <Trigger Property="Orientation"
            Value="Horizontal">
            <Setter Property="MinWidth"
                Value="104" />
            <Setter Property="MinHeight"
                Value="21" />
            <Setter Property="Template"
                Value="{StaticResource HorizontalSlider}" />
        </Trigger>
        <Trigger Property="Orientation"
            Value="Vertical">
            <Setter Property="MinWidth"
                Value="21" />
            <Setter Property="MinHeight"
                Value="104" />
            <Setter Property="Template"
                Value="{StaticResource VerticalSlider}" />
        </Trigger>
    </Style.Triggers>
</Style>
```

The preceding example uses one or more of the following resources.

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>
```

```

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# StatusBar Styles and Templates

2 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [StatusBar](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## StatusBar Parts

The [StatusBar](#) control does not have any named parts.

## StatusBar States

The following table lists the visual states for the [StatusBar](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## StatusBarItem Parts

The [StatusBarItem](#) control does not have any named parts.

## StatusBar States

The following table lists the visual states for the [StatusBarItem](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## StatusBar ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [StatusBar](#) control.

```
<Style x:Key="{x:Type StatusBar}">
    TargetType="{x:Type StatusBar}"
    <Setter Property="SnapsToDevicePixels"
        Value="True" />
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type StatusBar}">
                <Border Padding="1">
                    <Border.BorderBrush>
                        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1">
                            <LinearGradientBrush.GradientStops>
                                <GradientStopCollection>
                                    <GradientStop Color="{DynamicResource BorderLightColor}"
                                        Offset="0.0" />
                                    <GradientStop Color="{DynamicResource BorderDarkColor}"
                                        Offset="1.0" />
                                </GradientStopCollection>
                            </LinearGradientBrush.GradientStops>
                        </LinearGradientBrush>
                    </Border.BorderBrush>
                    <Border.Background>
                        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1">
                            <LinearGradientBrush.GradientStops>
                                <GradientStopCollection>
                                    <GradientStop Color="{DynamicResource ControlLightColor}"
                                        Offset="0.0" />
                                    <GradientStop Color="{DynamicResource ControlMediumColor}"
                                        Offset="1.0" />
                                </GradientStopCollection>
                            </LinearGradientBrush.GradientStops>
                        </LinearGradientBrush>
                    </Border.Background>
                    <ItemsPresenter />
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

<Style x:Key="{x:Static StatusBar.SeparatorStyleKey}">
    TargetType="{x:Type Separator}"
    <Setter Property="OverridesDefaultStyle"
        Value="True" />
    <Setter Property="SnapsToDevicePixels"
        Value="True" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Separator}">
                <Rectangle Width="1"
                    Margin="3">
                    <Rectangle.Fill>
                        <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
                    </Rectangle.Fill>
                </Rectangle>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

<Style x:Key="{x:Type StatusBarItem}">
    TargetType="{x:Type StatusBarItem}">
```

```

<Setter Property="OverridesDefaultStyle"
       Value="True" />
<Setter Property="Template">
  <Setter.Value>
    <ControlTemplate TargetType="{x:Type StatusBarItem}">
      <ContentPresenter Margin="3"
                        Name="ContentSite" />
      <ControlTemplate.Triggers>
        <Trigger Property="IsEnabled"
                  Value="false">
          <Setter Property="Foreground">
            <Setter.Value>
              <SolidColorBrush Color="{StaticResource DisabledForegroundColor}" />
            </Setter.Value>
          </Setter>
        </Trigger>
      </ControlTemplate.Triggers>
    </ControlTemplate>
  </Setter.Value>
</Setter>
</Style>

```

The [ControlTemplate](#) uses one or more of the following resources.

```

<!--Control colors-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

```

```
<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# TabControl Styles and Templates

3 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [TabControl](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## TabControl Parts

The following table lists the named parts for the [TabControl](#) control.

PART	TYPE	DESCRIPTION
PART_SelectedContentHost	ContentPresenter	The object that shows the content of the currently selected <a href="#">TabItem</a> .

When you create a [ControlTemplate](#) for a [TabControl](#), your template might contain an [ItemsPresenter](#) within a [ScrollViewer](#). (The [ItemsPresenter](#) displays each item in the [TabControl](#); the [ScrollViewer](#) enables scrolling within the control). If the [ItemsPresenter](#) is not the direct child of the [ScrollViewer](#), you must give the [ItemsPresenter](#) the name, `ItemsPresenter`.

## TabControl States

The following table lists the visual states for the [TabControl](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
Disabled	CommonStates	The control is disabled.
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## TabItem Parts

The [TabItem](#) control does not have any named parts.

## TabItem States

The following table lists the visual states for the [TabItem](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
MouseOver	CommonStates	The mouse pointer is positioned over the control.
Disabled	CommonStates	The control is disabled.
Focused	FocusStates	The control has focus.
Unfocused	FocusStates	The control does not have focus.
Selected	SelectionStates	The control is selected.
Unselected	SelectionStates	The control is not selected.
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## TabControl ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [TabControl](#) and [TabItem](#) controls.

```
<Style TargetType="{x:Type TabControl}">
    <Setter Property="OverridesDefaultStyle"
           Value="True" />
    <Setter Property="SnapsToDevicePixels"
           Value="True" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type TabControl}">
                <Grid KeyboardNavigation.TabNavigation="Local">
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto" />
                        <RowDefinition Height="*" />
                    </Grid.RowDefinitions>
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="Disabled">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                        Storyboard.TargetProperty="(Border.BorderBrush)".
                                        (SolidColorBrush.Color)">
                                        <EasingColorKeyFrame KeyTime="0"
                                            Value="#FFAAAAAA" />
                                    </ColorAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

```

<TabPanel x:Name="HeaderPanel"
          Grid.Row="0"
          Panel.ZIndex="1"
          Margin="0,0,4,-1"
          IsItemsHost="True"
          KeyboardNavigation.TabIndex="1"
          Background="Transparent" />
<Border x:Name="Border"
        Grid.Row="1"
        BorderThickness="1"
        CornerRadius="2"
        KeyboardNavigation.TabNavigation="Local"
        KeyboardNavigation.DirectionNavigation="Contained"
        KeyboardNavigation.TabIndex="2">
<Border.Background>
    <LinearGradientBrush EndPoint="0.5,1"
                         StartPoint="0.5,0">
        <GradientStop Color="{DynamicResource ContentAreaColorLight}"
                      Offset="0" />
        <GradientStop Color="{DynamicResource ContentAreaColorDark}"
                      Offset="1" />
    </LinearGradientBrush>
</Border.Background>
<Border.BorderBrush>
    <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
</Border.BorderBrush>
<ContentPresenter x:Name="PART_SelectedContentHost"
                  Margin="4"
                  ContentSource="SelectedContent" />
</Border>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style TargetType="{x:Type TabItem}">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type TabItem}">
                <Grid x:Name="Root">
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="SelectionStates">
                            <VisualState x:Name="Unselected" />
                            <VisualState x:Name="Selected">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                                               Storyboard.TargetProperty="(Panel.Background)."
                                                               (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                                        <EasingColorKeyFrame KeyTime="0"
                                                               Value="{StaticResource ControlPressedColor}" />
                                    </ColorAnimationUsingKeyFrames>
                                    <ThicknessAnimationUsingKeyFrames Storyboard.TargetProperty="(Border.BorderThickness)">
                                        <Storyboard.TargetName="Border">
                                            <EasingThicknessKeyFrame KeyTime="0"
                                                               Value="1,1,1,0" />
                                        </ThicknessAnimationUsingKeyFrames>
                                    </Storyboard>
                                </VisualState>
                            </VisualStateGroup>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="Normal" />
                            <VisualState x:Name="MouseOver" />
                            <VisualState x:Name="Disabled">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                                               Storyboard.TargetProperty="(Panel.Background)."
                                                               (GradientBrush.GradientStops)[1].(GradientStop.Color)">
                                        <EasingColorKeyFrame KeyTime="0"
                                                               Value="1,1,1,0" />
                                    </ColorAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

```

        Value="{StaticResource DisabledControlDarkColor}" />
    </ColorAnimationUsingKeyFrames>
    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                   Storyboard.TargetProperty="(Border.BorderBrush).(
SolidColorBrush.Color)">
        <EasingColorKeyFrame KeyTime="0"
                           Value="{StaticResource DisabledBorderLightColor}"/>
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Border x:Name="Border"
        Margin="0,0,-4,0"
        BorderThickness="1,1,1,1"
        CornerRadius="2,12,0,0">
    <Border.BorderBrush>
        <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
    </Border.BorderBrush>
    <Border.Background>

        <LinearGradientBrush StartPoint="0,0"
                           EndPoint="0,1">
            <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                    <GradientStop Color="{DynamicResource ControlLightColor}"
                                  Offset="0.0" />
                    <GradientStop Color="{DynamicResource ControlMediumColor}"
                                  Offset="1.0" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>

    </Border.Background>
    <ContentPresenter x:Name="ContentSite"
                      VerticalAlignment="Center"
                      HorizontalAlignment="Center"
                      ContentSource="Header"
                      Margin="12,2,12,2"
                      RecognizesAccessKey="True" />
</Border>
</Grid>
<ControlTemplate.Triggers>
    <Trigger Property="IsSelected"
             Value="True">
        <Setter Property="Panel.ZIndex"
               Value="100" />
    </Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

The preceding example uses one or more of the following resources.

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

```

```

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)

- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# TextBox Styles and Templates

2 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [TextBox](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## TextBox Parts

The following table lists the named parts for the [TextBox](#) control.

PART	TYPE	DESCRIPTION
PART_ContentHost	FrameworkElement	A visual element that can contain a <a href="#">FrameworkElement</a> . The text of the <a href="#">TextBox</a> is displayed in this element.

## TextBox States

The following table lists the visual states for the [TextBox](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
MouseOver	CommonStates	The mouse pointer is positioned over the control.
Disabled	CommonStates	The control is disabled.
ReadOnly	CommonStates	The user cannot change the text in the <a href="#">TextBox</a> .
Focused	FocusStates	The control has focus.
Unfocused	FocusStates	The control does not have focus.
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## TextBox ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [TextBox](#) control.

```

<Style TargetType="{x:Type TextBox}">
    <Setter Property="SnapsToDevicePixels"
        Value="True" />
    <Setter Property="OverridesDefaultStyle"
        Value="True" />
    <Setter Property="KeyboardNavigation.TabNavigation"
        Value="None" />
    <Setter Property="FocusVisualStyle"
        Value="{x:Null}" />
    <Setter Property="MinWidth"
        Value="120" />
    <Setter Property="MinHeight"
        Value="20" />
    <Setter Property="AllowDrop"
        Value="true" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type TextBoxBase}">
                <Border Name="Border"
                    CornerRadius="2"
                    Padding="2"
                    BorderThickness="1">
                    <Border.Background>
                        <SolidColorBrush Color="{DynamicResource ControlLightColor}" />
                    </Border.Background>
                    <Border.BorderBrush>
                        <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
                    </Border.BorderBrush>
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="Normal" />
                            <VisualState x:Name="Disabled">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                        Storyboard.TargetProperty="(Panel.Background).(
                                            SolidColorBrush.Color)">
                                        <EasingColorKeyFrame KeyTime="0"
                                            Value="{StaticResource DisabledControlLightColor}" />
                                    </ColorAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="ReadOnly">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                        Storyboard.TargetProperty="(Panel.Background).(
                                            SolidColorBrush.Color)">
                                        <EasingColorKeyFrame KeyTime="0"
                                            Value="{StaticResource DisabledControlDarkColor}" />
                                    </ColorAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="MouseOver" />
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                    <ScrollViewer Margin="0"
                        x:Name="PART_ContentHost" />
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

The preceding example uses one or more of the following resources.

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>

```

```

<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

---

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# Thumb Styles and Templates

2 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [Thumb](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## Thumb Parts

The [Thumb](#) control does not have any named parts.

## Thumb States

The following table lists the visual states for the [Thumb](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
MouseOver	CommonStates	The mouse pointer is positioned over the control.
Pressed	CommonStates	The control is pressed.
Disabled	CommonStates	The control is disabled.
Focused	FocusStates	The control has focus.
Unfocused	FocusStates	The control does not have focus.
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## Thumb ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [Thumb](#) control.

```
<Style x:Key="SliderThumbStyle"
    TargetType="{x:Type Thumb}">
    <Setter Property="SnapsToDevicePixels"
        Value="true" />
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="Height"
        Value="14" />
```

```

<Setter Property="Width"
       Value="14" />
<Setter Property="Template">
  <Setter.Value>
    <ControlTemplate TargetType="{x:Type Thumb}">
      <Ellipse x:Name="Ellipse"
                StrokeThickness="1">
        <Ellipse.Stroke>
          <LinearGradientBrush StartPoint="0,0"
                               EndPoint="0,1">
            <LinearGradientBrush.GradientStops>
              <GradientStopCollection>
                <GradientStop Color="{DynamicResource BorderLightColor}"
                              Offset="0.0" />
                <GradientStop Color="{DynamicResource BorderDarkColor}"
                              Offset="1.0" />
              </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
          </LinearGradientBrush>
        </Ellipse.Stroke>
        <Ellipse.Fill>
          <LinearGradientBrush EndPoint="0.5,1"
                               StartPoint="0.5,0">
            <GradientStop Color="{DynamicResource ControlMediumColor}"
                          Offset="1" />
            <GradientStop Color="{DynamicResource ControlLightColor}" />
          </LinearGradientBrush>
        </Ellipse.Fill>
      </VisualStateManager.VisualStateGroups>
      <VisualStateGroup x:Name="CommonStates">
        <VisualState x:Name="Normal" />
        <VisualState x:Name="MouseOver">
          <Storyboard>
            <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Shape.Fill).{GradientBrush.GradientStops}[0].{GradientStop.Color}" Storyboard.TargetName="Ellipse">
              <EasingColorKeyFrame KeyTime="0" Value="{StaticResource ControlMouseOverColor}" />
            </ColorAnimationUsingKeyFrames>
          </Storyboard>
        </VisualState>
        <VisualState x:Name="Pressed">
          <Storyboard>
            <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Shape.Fill).{GradientBrush.GradientStops}[0].{GradientStop.Color}" Storyboard.TargetName="Ellipse">
              <EasingColorKeyFrame KeyTime="0" Value="{StaticResource ControlPressedColor}" />
            </ColorAnimationUsingKeyFrames>
          </Storyboard>
        </VisualState>
        <VisualState x:Name="Disabled">
          <Storyboard>
            <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Shape.Fill).{GradientBrush.GradientStops}[0].{GradientStop.Color}" Storyboard.TargetName="Ellipse">
              <EasingColorKeyFrame KeyTime="0" Value="{StaticResource DisabledControlDarkColor}" />
            </ColorAnimationUsingKeyFrames>
          </Storyboard>
        </VisualState>
      </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>
  </Ellipse>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

The preceding example uses one or more of the following resources.

```
<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

```
<GradientStop Color="#000000FF"
    Offset="1" />
</GradientStopCollection>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# ToggleButton Styles and Templates

2 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [ToggleButton](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## ToggleButton Parts

The [ToggleButton](#) control does not have any named parts.

## ToggleButton States

The following table lists the visual states for the [ToggleButton](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.
MouseOver	CommonStates	The mouse pointer is positioned over the control.
Pressed	CommonStates	The control is pressed.
Disabled	CommonStates	The control is disabled.
Focused	FocusStates	The control has focus.
Unfocused	FocusStates	The control does not have focus.
Checked	CheckStates	<code>IsChecked</code> is <code>true</code> .
Unchecked	CheckStates	<code>IsChecked</code> is <code>false</code> .
Indeterminate	CheckStates	<code>IsThreeState</code> is <code>true</code> , and <code>IsChecked</code> is <code>null</code> .
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> if the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> if the control does not have focus.

**NOTE**

If the Indeterminate visual state does not exist in your control template, then the Unchecked visual state will be used as default visual state.

## ToggleButton ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [ToggleButton](#) control.

```
<ControlTemplate x:Key="ComboBoxToggleButton"
    TargetType="{x:Type ToggleButton}">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition />
            <ColumnDefinition Width="20" />
        </Grid.ColumnDefinitions>
        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="CommonStates">
                <VisualState x:Name="Normal" />
                <VisualState x:Name="MouseOver">
                    <Storyboard>
                        <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Panel.Background)."
                            (GradientBrush.GradientStops)[1].(GradientStop.Color)" 
                            Storyboard.TargetName="Border">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource ControlMouseOverColor}" />
                        </ColorAnimationUsingKeyFrames>
                    </Storyboard>
                </VisualState>
                <VisualState x:Name="Pressed" />
                <VisualState x:Name="Disabled">
                    <Storyboard>
                        <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Panel.Background)."
                            (GradientBrush.GradientStops)[1].(GradientStop.Color)" 
                            Storyboard.TargetName="Border">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource DisabledControlDarkColor}" />
                        </ColorAnimationUsingKeyFrames>
                        <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Shape.Fill)."
                            (SolidColorBrush.Color)" 
                            Storyboard.TargetName="Arrow">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource DisabledForegroundColor}" />
                        </ColorAnimationUsingKeyFrames>
                        <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Border.BorderBrush)."
                            (GradientBrush.GradientStops)[1].(GradientStop.Color)" 
                            Storyboard.TargetName="Border">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource DisabledBorderDarkColor}" />
                        </ColorAnimationUsingKeyFrames>
                    </Storyboard>
                </VisualState>
            </VisualStateGroup>
            <VisualStateGroup x:Name="CheckStates">
                <VisualState x:Name="Checked">
                    <Storyboard>
                        <ColorAnimationUsingKeyFrames Storyboard.TargetProperty="(Panel.Background)."
                            (GradientBrush.GradientStops)[1].(GradientStop.Color)" 
                            Storyboard.TargetName="Border">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource ControlPressedColor}" />
                        </ColorAnimationUsingKeyFrames>
                    </Storyboard>
                </VisualState>
                <VisualState x:Name="Unchecked" />
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>
    </Grid>

```

```

<VisualState x:Name="Indeterminate" />
</VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<Border x:Name="Border"
        Grid.ColumnSpan="2"
        CornerRadius="2"
        BorderThickness="1">
    <Border.BorderBrush>
        <LinearGradientBrush EndPoint="0,1"
                            StartPoint="0,0">
            <GradientStop Color="{DynamicResource BorderLightColor}"
                           Offset="0" />
            <GradientStop Color="{DynamicResource BorderDarkColor}"
                           Offset="1" />
        </LinearGradientBrush>
    </Border.BorderBrush>
    <Border.Background>

        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1">
            <LinearGradientBrush.GradientStops>
                <GradientStopCollection>
                    <GradientStop Color="{DynamicResource ControlLightColor}" />
                    <GradientStop Color="{DynamicResource ControlMediumColor}"
                                   Offset="1.0" />
                </GradientStopCollection>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Border.Background>
</Border>
<Border Grid.Column="0"
        CornerRadius="2,0,0,2"
        Margin="1" >
    <Border.Background>
        <SolidColorBrush Color="{DynamicResource ControlLightColor}" />
    </Border.Background>
</Border>
<Path x:Name="Arrow"
      Grid.Column="1"
      HorizontalAlignment="Center"
      VerticalAlignment="Center"
      Data="M 0 0 L 4 4 L 8 0 Z" >
    <Path.Fill>
        <SolidColorBrush Color="{DynamicResource GlyphColor}" />
    </Path.Fill>
</Path>
</Grid>
</ControlTemplate>

```

The preceding example uses one or more of the following resources.

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

```

```

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)

- [Styling and Templating](#)
- [Create a template for a control](#)

# ToolBar Styles and Templates

4 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [ToolBar](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## ToolBar Parts

The following table lists the named parts for the [ToolBar](#) control.

PART	TYPE	DESCRIPTION
PART_ToolBarPanel	<a href="#">ToolBarPanel</a>	The object that contains the controls on the <a href="#">ToolBar</a> .
PART_ToolBarOverflowPanel	<a href="#">ToolBarOverflowPanel</a>	The object that contains the controls that are in the overflow area of the <a href="#">ToolBar</a> .

When you create a [ControlTemplate](#) for a [ToolBar](#), your template might contain an [ItemsPresenter](#) within a [ScrollViewer](#). (The [ItemsPresenter](#) displays each item in the [ToolBar](#); the [ScrollViewer](#) enables scrolling within the control). If the [ItemsPresenter](#) is not the direct child of the [ScrollViewer](#), you must give the [ItemsPresenter](#) the name, `ItemsPresenter`.

## ToolBar States

The following table lists the visual states for the [ToolBar](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code>   has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code>   has the control does not have focus.

## ToolBar ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [ToolBar](#) control.

```
<Style x:Key="ToolBarButtonBaseStyle"
       TargetType="{x:Type ButtonBase}">
    <Setter Property="SnapsToDevicePixels"
           Value="true" />
    <Setter Property="OverridesDefaultStyle"
           Value="true" />
    <Setter Property="Template">
```

```

<Setter.Value>
<ControlTemplate TargetType="{x:Type ButtonBase}">
    <Border x:Name="Border"
        BorderThickness="1">
        <Border.BorderBrush>
            <LinearGradientBrush EndPoint="0.5,1"
                StartPoint="0.5,0">
                <GradientStop Color="{DynamicResource BorderLightColor}"
                    Offset="0" />
                <GradientStop Color="{DynamicResource BorderMediumColor}"
                    Offset="1" />
            </LinearGradientBrush>
        </Border.BorderBrush>
        <Border.Background>
            <LinearGradientBrush EndPoint="0.5,1"
                StartPoint="0.5,0">
                <GradientStop Color="{DynamicResource ControlLightColor}"
                    Offset="0" />
                <GradientStop Color="{DynamicResource ControlMediumColor}"
                    Offset="1" />
            </LinearGradientBrush>
        </Border.Background>
        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="CommonStates">
                <VisualState x:Name="Normal" />
                <VisualState x:Name="Pressed">
                    <Storyboard>
                        <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                            Storyboard.TargetProperty="(Panel.Background).(
                                GradientBrush.GradientStops)[1].(GradientStop.Color)">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource ControlPressedColor}" />
                        </ColorAnimationUsingKeyFrames>
                    </Storyboard>
                </VisualState>
                <VisualState x:Name="MouseOver">
                    <Storyboard>
                        <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                            Storyboard.TargetProperty="(Panel.Background).(
                                GradientBrush.GradientStops)[1].(GradientStop.Color)">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource ControlMouseOverColor}" />
                        </ColorAnimationUsingKeyFrames>
                    </Storyboard>
                </VisualState>
                <VisualState x:Name="Disabled">
                    <Storyboard>
                        <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                            Storyboard.TargetProperty="(Panel.Background).(
                                GradientBrush.GradientStops)[1].(GradientStop.Color)">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource DisabledControlDarkColor}" />
                        </ColorAnimationUsingKeyFrames>
                    </Storyboard>
                </VisualState>
            </VisualStateGroup>
            <VisualStateGroup x:Name="CheckStates">
                <VisualState x:Name="Checked">
                    <Storyboard>
                        <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                            Storyboard.TargetProperty="(Panel.Background).(
                                GradientBrush.GradientStops)[1].(GradientStop.Color)">
                            <EasingColorKeyFrame KeyTime="0"
                                Value="{StaticResource ControlPressedColor}" />
                        </ColorAnimationUsingKeyFrames>
                    </Storyboard>
                </VisualState>
                <VisualState x:Name="Unchecked" />
                <VisualState x:Name="Indeterminate" />
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>
    </Border>

```

```

        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>
<ContentPresenter Margin="2"
                  HorizontalAlignment="Center"
                  VerticalAlignment="Center"
                  RecognizesAccessKey="True" />
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="{x:StaticToolBar.ButtonStyleKey}"
       BasedOn="{StaticResourceToolBarButtonBaseStyle}"
       TargetType="{x:Type Button}" />
<Style x:Key="{x:StaticToolBar.ToggleButtonStyleKey}"
       BasedOn="{StaticResourceToolBarButtonBaseStyle}"
       TargetType="{x:Type ToggleButton}" />
<Style x:Key="{x:StaticToolBar.CheckBoxStyleKey}"
       BasedOn="{StaticResourceToolBarButtonBaseStyle}"
       TargetType="{x:Type CheckBox}" />
<Style x:Key="{x:StaticToolBar.RadioButtonStyleKey}"
       BasedOn="{StaticResourceToolBarButtonBaseStyle}"
       TargetType="{x:Type RadioButton}" />

<Style x:Key="{x:StaticToolBar.TextBoxStyleKey}"
       TargetType="{x:Type TextBox}">
<Setter Property="SnapsToDevicePixels"
       Value="True" />
<Setter Property="OverridesDefaultStyle"
       Value="True" />
<Setter Property="KeyboardNavigation.TabNavigation"
       Value="None" />
<Setter Property="FocusVisualStyle"
       Value="{x:Null}" />
<Setter Property="AllowDrop"
       Value="true" />
<Setter Property="Template">
<Setter.Value>
    <ControlTemplate TargetType="{x:Type TextBox}">
        <Border x:Name="Border"
               Padding="2"
               BorderThickness="1">
            <Border.Background>
                <SolidColorBrush Color="{DynamicResourceControlLightColor}" />
            </Border.Background>
            <Border.BorderBrush>
                <SolidColorBrush Color="{StaticResourceBorderMediumColor}" />
            </Border.BorderBrush>
            <VisualStateManager.VisualStateGroups>
                <VisualStateGroup x:Name="CommonStates">
                    <VisualState x:Name="Normal" />
                    <VisualState x:Name="Disabled">
                        <Storyboard>
                            <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                                          Storyboard.TargetProperty="(Panel.Background).{SolidColorBrush.Color}">
                                <EasingColorKeyFrame KeyTime="0"
                                                    Value="{StaticResourceDisabledControlDarkColor}" />
                            </ColorAnimationUsingKeyFrames>
                            <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                                          Storyboard.TargetProperty="(Border.BorderBrush).{SolidColorBrush.Color}">
                                <EasingColorKeyFrame KeyTime="0"
                                                    Value="{StaticResourceDisabledBorderLightColor}" />
                            </ColorAnimationUsingKeyFrames>
                        </Storyboard>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateManager.VisualStateGroups>
        </Border>
    </ControlTemplate>
</Setter.Value>
</Style>
<VisualState x:Name="ReadOnly" />

```

```

        <VisualState x:Name="MouseOver" />
    </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
<ScrollViewer Margin="0"
    x:Name="PART_ContentHost" />
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="ToolBarThumbStyle"
    TargetType="{x:Type Thumb}">
<Setter Property="OverridesDefaultStyle"
    Value="true" />
<Setter Property="Cursor"
    Value="SizeAll" />
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type Thumb}">
<Border Background="Transparent"
    SnapsToDevicePixels="True">
<Rectangle Margin="0,2">
<Rectangle.Fill>
<DrawingBrush Viewport="0,0,4,4"
    ViewportUnits="Absolute"
    Viewbox="0,0,8,8"
    ViewboxUnits="Absolute"
    TileMode="Tile">
<DrawingBrush.Drawing>
<DrawingGroup>
<GeometryDrawing Brush="#AAA"
    Geometry="M 4 4 L 4 8 L 8 8 L 8 4 z" />
</DrawingGroup>
</DrawingBrush.Drawing>
</DrawingBrush>
</Rectangle.Fill>
</Rectangle>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="ToolBarOverflowButtonStyle"
    TargetType="{x:Type ToggleButton}">
<Setter Property="OverridesDefaultStyle"
    Value="true" />
<Setter Property="Template">
<Setter.Value>
<ControlTemplate TargetType="{x:Type ToggleButton}">
<Border x:Name="Border"
    CornerRadius="0,3,3,0"
    SnapsToDevicePixels="true">
<Border.Background>
<LinearGradientBrush EndPoint="0.5,1"
    StartPoint="0.5,0">
<GradientStop Color="#00000000"
    Offset="0" />
<GradientStop Offset="1" />
</LinearGradientBrush>
</Border.Background>
</VisualStateManager.VisualStateGroups>
<VisualStateGroup x:Name="CommonStates">
<VisualState x:Name="Normal" />
<VisualState x:Name="Pressed">
<Storyboard>
<ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
    Storyboard.TargetProperty="(Panel.Background)">

```

```

        (GradientBrush.GradientStops)[1].(GradientStop.Color)">
        <EasingColorKeyFrame KeyTime="0"
            Value="{StaticResource ControlPressedColor}" />
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="MouseOver">
    <Storyboard>
        <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
            Storyboard.TargetProperty="(Panel.Background).(
                GradientBrush.GradientStops)[1].(GradientStop.Color)">
        <EasingColorKeyFrame KeyTime="0"
            Value="{StaticResource ControlMouseOverColor}" />
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
<VisualState x:Name="Disabled">
    <Storyboard>
        <ColorAnimationUsingKeyFrames Storyboard.TargetName="Border"
            Storyboard.TargetProperty="(Panel.Background).(
                GradientBrush.GradientStops)[1].(GradientStop.Color)">
        <EasingColorKeyFrame KeyTime="0"
            Value="{StaticResource DisabledBorderLightColor}" />
    </ColorAnimationUsingKeyFrames>
</Storyboard>
</VisualState>
</VisualStateManager.VisualStateGroups>
<Grid>
    <Path x:Name="Arrow"
        Fill="Black"
        VerticalAlignment="Bottom"
        Margin="2,3"
        Data="M -0.5 3 L 5.5 3 L 2.5 6 Z" />
    <ContentPresenter />
</Grid>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="{x:TypeToolBar}"
    TargetType="{x:TypeToolBar}">
    <Setter Property="SnapsToDevicePixels"
        Value="true" />
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:TypeToolBar}">
                <Border x:Name="Border"
                    CornerRadius="2"
                    BorderThickness="1">
                    <Border.BorderBrush>
                        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1">
                            <LinearGradientBrush.GradientStops>
                                <GradientStopCollection>
                                    <GradientStop Color="{DynamicResource BorderMediumColor}"
                                        Offset="0.0" />
                                    <GradientStop Color="{DynamicResource BorderDarkColor}"
                                        Offset="1.0" />
                                </GradientStopCollection>
                            </LinearGradientBrush.GradientStops>
                        </LinearGradientBrush>
                    </Border.BorderBrush>
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

```

<Border .Background>
    <LinearGradientBrush StartPoint="0,0"
        EndPoint="0,1">
        <LinearGradientBrush.GradientStops>
            <GradientStopCollection>
                <GradientStop Color="#FFF"
                    Offset="0.0" />
                <GradientStop Color="{DynamicResource ControlMediumColor}"
                    Offset="1.0" />
            </GradientStopCollection>
        </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
</Border.Background>
<DockPanel>
    <ToggleButton DockPanel.Dock="Right"
        IsEnabled="{TemplateBinding HasOverflowItems}"
        Style="{StaticResourceToolBarOverflowButtonStyle}"
        ClickMode="Press"
        IsChecked="{Binding IsOverflowOpen, Mode=TwoWay,
        RelativeSource={RelativeSource TemplatedParent}}">
        <Popup x:Name="OverflowPopup"
            AllowsTransparency="true"
            Placement="Bottom"
            StaysOpen="false"
            Focusable="false"
            PopupAnimation="Slide"
            IsOpen="{Binding IsOverflowOpen,
            RelativeSource={RelativeSource TemplatedParent}}">
            <Border x:Name="DropDownBorder"
                BorderThickness="1">
                <Border.BorderBrush>
                    <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
                </Border.BorderBrush>
                <Border.Background>
                    <LinearGradientBrush EndPoint="0.5,1"
                        StartPoint="0.5,0">
                        <GradientStop Color="{DynamicResource ControlLightColor}" />
                        <GradientStop Color="{DynamicResource ControlMediumColor}"
                            Offset="1" />
                    </LinearGradientBrush>
                </Border.Background>
                <ToolBarOverflowPanel x:Name="PART_ToolBarOverflowPanel"
                    Margin="2"
                    WrapWidth="200"
                    Focusable="true"
                    FocusVisualStyle="{x:Null}"
                    KeyboardNavigation.TabNavigation="Cycle"
                    KeyboardNavigationDirectionalNavigation="Cycle" />
            </Border>
        </Popup>
    </ToggleButton>

    <Thumb x:Name="ToolBarThumb"
        Style="{StaticResourceToolBarThumbStyle}"
        Width="10" />
<ToolBarPanel x:Name="PART_ToolBarPanel"
    IsItemsHost="true"
    Margin="0,1,2,2" />
</DockPanel>
</Border>
<ControlTemplate.Triggers>
    <Trigger Property="IsOverflowOpen"
        Value="true">
        <Setter TargetName="ToolBarThumb"
            Property="IsEnabled"
            Value="false" />
    </Trigger>
    <Trigger Property="ToolBarTray.IsLocked"
        Value="true">

```

```

        <Setter TargetName="ToolBarThumb"
            Property="Visibility"
            Value="Collapsed" />
    </Trigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="{x:TypeToolBarTray}"
    TargetType="{x:TypeToolBarTray}">
    <Setter Property="Background">
        <Setter.Value>
            <LinearGradientBrush StartPoint="0,0"
                EndPoint="1,0">
                <LinearGradientBrush.GradientStops>
                    <GradientStopCollection>
                        <GradientStop Color="#FFF"
                            Offset="0.0" />
                        <GradientStop Color="{DynamicResource WindowColor}"
                            Offset="1.0" />
                    </GradientStopCollection>
                </LinearGradientBrush.GradientStops>
            </LinearGradientBrush>
        </Setter.Value>
    </Setter>
</Style>

```

The preceding example uses one or more of the following resources.

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FCCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

```

```

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# ToolTip Styles and Templates

2 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [ToolTip](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## ToolTip Parts

The [ToolTip](#) control does not have any named parts.

## ToolTip States

The following table lists the visual states for the [ToolTip](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Closed	OpenStates	The default state.
Open	OpenStates	The <a href="#">ToolTip</a> is visible.
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## ToolTip ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [ToolTip](#) control.

```

<Style x:Key="{x:Type ToolTip}"
    TargetType="ToolTip">
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="HasDropShadow"
        Value="True" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="ToolTip">
                <Border Name="Border"
                    BorderThickness="1"
                    Width="{TemplateBinding Width}"
                    Height="{TemplateBinding Height}">
                    <Border.Background>
                        <LinearGradientBrush StartPoint="0,0"
                            EndPoint="0,1">
                            <LinearGradientBrush.GradientStops>
                                <GradientStopCollection>
                                    <GradientStop Color="{DynamicResource ControlLightColor}"
                                        Offset="0.0" />
                                    <GradientStop Color="{DynamicResource ControlMediumColor}"
                                        Offset="1.0" />
                                </GradientStopCollection>
                            </LinearGradientBrush.GradientStops>
                        </LinearGradientBrush>
                    </Border.Background>
                    <Border.BorderBrush>
                        <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
                    </Border.BorderBrush>
                    <ContentPresenter Margin="4"
                        HorizontalAlignment="Left"
                        VerticalAlignment="Top" />
                </Border>
                <ControlTemplate.Triggers>
                    <Trigger Property="HasDropShadow"
                        Value="true">
                        <Setter TargetName="Border"
                            Property="CornerRadius"
                            Value="4" />
                        <Setter TargetName="Border"
                            Property="SnapsToDevicePixels"
                            Value="true" />
                    </Trigger>
                </ControlTemplate.Triggers>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

The preceding example uses one or more of the following resources.

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>

```

```

<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)

- [Styling and Templating](#)
- [Create a template for a control](#)

# TreeView Styles and Templates

4 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [TreeView](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## TreeView Parts

The [TreeView](#) control does not have any named parts.

When you create a [ControlTemplate](#) for an [TreeView](#), your template might contain a [ItemsPresenter](#) within a [ScrollViewer](#). (The [ItemsPresenter](#) displays each item in the [TreeView](#); the [ScrollViewer](#) enables scrolling within the control). If the [ItemsPresenter](#) is not the direct child of the [ScrollViewer](#), you must give the [ItemsPresenter](#) the name, `ItemsPresenter`.

## TreeView States

The following table lists the visual states for the [TreeView](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## TreeViewItem Parts

The following table lists the named parts for the [TreeViewItem](#) control.

PART	TYPE	DESCRIPTION
PART_Header	FrameworkElement	A visual element that contains that header content of the <a href="#">TreeView</a> control.

## TreeViewItem States

The following table lists the visual states for [TreeViewItem](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Normal	CommonStates	The default state.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
MouseOver	CommonStates	The mouse pointer is positioned over the <a href="#">TreeViewItem</a> .
Disabled	CommonStates	The <a href="#">TreeViewItem</a> is disabled.
Focused	FocusStates	The <a href="#">TreeViewItem</a> has focus.
Unfocused	FocusStates	The <a href="#">TreeViewItem</a> does not have focus.
Expanded	ExpansionStates	The <a href="#">TreeViewItem</a> control is expanded.
Collapsed	ExpansionStates	The <a href="#">TreeViewItem</a> control is collapsed.
HasItems	HasItemsStates	The <a href="#">TreeViewItem</a> has items.
NoItems	HasItemsStates	The <a href="#">TreeViewItem</a> does not have items.
Selected	SelectionStates	The <a href="#">TreeViewItem</a> is selected.
SelectedInactive	SelectionStates	The <a href="#">TreeViewItem</a> is selected but not active.
Unselected	SelectionStates	The <a href="#">TreeViewItem</a> is not selected.
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> if the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> if the control does not have focus.

## TreeView ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [TreeView](#) control and its associated types.

```
<Style x:Key="{x:Type TreeView}"
       TargetType="TreeView">
    <Setter Property="OverridesDefaultStyle"
           Value="True" />
    <Setter Property="SnapsToDevicePixels"
           Value="True" />
    <Setter Property="ScrollViewer.HorizontalScrollBarVisibility"
           Value="Auto" />
    <Setter Property="ScrollViewer.VerticalScrollBarVisibility"
           Value="Auto" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="TreeView">
                <Border Name="Border"
                       CornerRadius="1"
                       BorderThickness="1">
```

```

<Border.BorderBrush>
    <SolidColorBrush Color="{DynamicResource BorderMediumColor}" />
</Border.BorderBrush>
<Border.Background>
    <SolidColorBrush Color="{DynamicResource ControlLightColor}" />
</Border.Background>
<ScrollViewer Focusable="False"
    CanContentScroll="False"
    Padding="4">
    <ItemsPresenter />
</ScrollViewer>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

<Style x:Key="ExpandCollapseToggleStyle"
    TargetType="ToggleButton">
    <Setter Property="Focusable"
        Value="False" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="ToggleButton">
                <Grid Width="15"
                    Height="13"
                    Background="Transparent">
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CheckStates">
                            <VisualState x:Name="Checked">
                                <Storyboard>
                                    <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)"
                                        Storyboard.TargetName="Collapsed">
                                        <DiscreteObjectKeyFrame KeyTime="0"
                                            Value="{x:Static Visibility.Hidden}" />
                                    <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)"
                                        Storyboard.TargetName="Expanded">
                                        <DiscreteObjectKeyFrame KeyTime="0"
                                            Value="{x:Static Visibility.Visible}" />
                                    </ObjectAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="Unchecked" />
                            <VisualState x:Name="Indeterminate" />
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                    <Path x:Name="Collapsed"
                        HorizontalAlignment="Left"
                        VerticalAlignment="Center"
                        Margin="1,1,1,1"
                        Data="M 4 0 L 8 4 L 4 8 Z">
                        <Path.Fill>
                            <SolidColorBrush Color="{DynamicResource GlyphColor}" />
                        </Path.Fill>
                    </Path>
                    <Path x:Name="Expanded"
                        HorizontalAlignment="Left"
                        VerticalAlignment="Center"
                        Margin="1,1,1,1"
                        Data="M 0 4 L 8 4 L 4 8 Z"
                        Visibility="Hidden">
                        <Path.Fill>
                            <SolidColorBrush Color="{DynamicResource GlyphColor}" />
                        </Path.Fill>
                    </Path>
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

```

    </Setter>
</Style>
<Style x:Key="TreeViewItemFocusVisual">
    <Setter Property="Control.Template">
        <Setter.Value>
            <ControlTemplate>
                <Border>
                    <Rectangle Margin="0,0,0,0"
                               StrokeThickness="5"
                               Stroke="Black"
                               StrokeDashArray="1 2"
                               Opacity="0" />
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
<Style x:Key="{x:Type TreeViewItem}">
    TargetType="{x:Type TreeViewItem}"
    <Setter Property="Background"
           Value="Transparent" />
    <Setter Property="HorizontalContentAlignment"
           Value="{Binding Path=HorizontalContentAlignment,
                           RelativeSource={RelativeSource AncestorType={x:Type ItemsControl}}}" />
    <Setter Property="VerticalContentAlignment"
           Value="{Binding Path=VerticalContentAlignment,
                           RelativeSource={RelativeSource AncestorType={x:Type ItemsControl}}}" />
    <Setter Property="Padding"
           Value="1,0,0,0" />
    <Setter Property="Foreground"
           Value="{DynamicResource {x:Static SystemColors.ControlTextBrushKey}}" />
    <Setter Property="FocusVisualStyle"
           Value="{StaticResource TreeViewItemFocusVisual}" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type TreeViewItem}">
                <Grid>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition MinWidth="19"
                                         Width="Auto" />
                        <ColumnDefinition Width="Auto" />
                        <ColumnDefinition Width="*" />
                    </Grid.ColumnDefinitions>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto" />
                        <RowDefinition />
                    </Grid.RowDefinitions>
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="SelectionStates">
                            <VisualState x:Name="Selected">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Bd"
                                                               Storyboard.TargetProperty="(Panel.Background).(
SolidColorBrush.Color)">
                                        <EasingColorKeyFrame KeyTime="0"
                                                               Value="{StaticResource SelectedBackgroundColor}" />
                                    </ColorAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="Unselected" />
                            <VisualState x:Name="SelectedInactive">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames Storyboard.TargetName="Bd"
                                                               Storyboard.TargetProperty="(Panel.Background).(
SolidColorBrush.Color)">
                                        <EasingColorKeyFrame KeyTime="0"
                                                               Value="{StaticResource SelectedUnfocusedColor}" />
                                    </ColorAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

```
</ControlTemplate>
</ControlTemplate>
<ControlTemplate.Triggers>
<Trigger Property="HasItems"
Value="false">
<Setter TargetName="Expander"
Property="Visibility"
Value="Hidden" />
</Trigger>
<MultiTrigger>
<MultiTrigger.Conditions>
<Condition Property="HasHeader"
Value="false" />
<Condition Property="Width"
Value="Auto" />
</MultiTrigger.Conditions>
<Setter TargetName="PART_Header"
Property="MinWidth"
Value="75" />
</MultiTrigger>
<MultiTrigger>
<MultiTrigger.Conditions>
<Condition Property="HasHeader"
Value="false" />
<Condition Property="Height"
Value="Auto" />
</MultiTrigger.Conditions>
<Setter TargetName="PART_Header"
Property="MinHeight"
Value="19" />
</MultiTrigger>
</ControlTemplate.Triggers>
</ControlTemplate>
</ControlTemplate>
```

```

        </Setter.Value>
    </Setter>
</Style>

```

The preceding example uses one or more of the following resources.

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>

```

```
<GradientStop Color="#000000FF"
    Offset="0.4" />
<GradientStop Color="#600000FF"
    Offset="0.6" />
<GradientStop Color="#000000FF"
    Offset="1" />
</GradientStopCollection>
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# Window Styles and Templates

2 minutes to read • [Edit Online](#)

This topic describes the styles and templates for the [Window](#) control. You can modify the default [ControlTemplate](#) to give the control a unique appearance. For more information, see [Create a template for a control](#).

## Window Parts

The [Window](#) control does not have any named parts.

## Window States

The following table lists the visual states for the [Window](#) control.

VISUALSTATE NAME	VISUALSTATEGROUP NAME	DESCRIPTION
Valid	ValidationStates	The control uses the <a href="#">Validation</a> class and the <a href="#">HasError</a> attached property is <code>false</code> .
InvalidFocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control has focus.
InvalidUnfocused	ValidationStates	The <a href="#">HasError</a> attached property is <code>true</code> has the control does not have focus.

## Window ControlTemplate Example

The following example shows how to define a [ControlTemplate](#) for the [Window](#) control.

```
<Style x:Key="{x:Type Window}">
    TargetType="{x:Type Window}"
    <Setter Property="SnapsToDevicePixels"
        Value="true" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Window}">
                <Grid>
                    <Grid.Background>
                        <SolidColorBrush Color="{DynamicResource WindowColor}"/>
                    </Grid.Background>
                    <AdornerDecorator>
                        <ContentPresenter />
                    </AdornerDecorator>
                    <ResizeGrip x:Name="WindowResizeGrip"
                        HorizontalAlignment="Right"
                        VerticalAlignment="Bottom"
                        Visibility="Collapsed"
                        IsTabStop="false" />
                </Grid>
                <ControlTemplate.Triggers>
                    <Trigger Property="ResizeMode"
                        Value="CanResizeWithGrip">
                        <Setter TargetName="WindowResizeGrip"
                            Property="Visibility"
                            Value="Visible" />
                    </Trigger>
                </ControlTemplate.Triggers>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

The [ControlTemplate](#) uses one or more of the following resources.

```

<Style x:Key="{x:Type ResizeGrip}"
    TargetType="{x:Type ResizeGrip}">
    <Setter Property="OverridesDefaultStyle"
        Value="true" />
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type ResizeGrip}">
                <Border Background="Transparent"
                    SnapsToDevicePixels="True"
                    Width="16"
                    Height="16">
                    <Rectangle Margin="2">
                        <Rectangle.Fill>
                            <DrawingBrush Viewport="0,0,4,4"
                                ViewportUnits="Absolute"
                                Viewbox="0,0,8,8"
                                ViewboxUnits="Absolute"
                                TileMode="Tile">
                                <DrawingBrush.Drawing>
                                    <DrawingGroup>
                                        <DrawingGroup.Children>
                                            <GeometryDrawing Brush="#FFE8EDF9"
                                                Geometry="M 4 4 L 4 8 L
                                                8 8 L 8 4 z" />
                                        </DrawingGroup.Children>
                                    </DrawingGroup>
                                </DrawingBrush.Drawing>
                            </DrawingBrush>
                        </Rectangle.Fill>
                    </Rectangle>
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

```

```

<!--Control colors.-->
<Color x:Key="WindowColor">#FFE8EDF9</Color>
<Color x:Key="ContentAreaColorLight">#FFC5CBF9</Color>
<Color x:Key="ContentAreaColorDark">#FF7381F9</Color>

<Color x:Key="DisabledControlLightColor">#FFE8EDF9</Color>
<Color x:Key="DisabledControlDarkColor">#FFC5CBF9</Color>
<Color x:Key="DisabledForegroundColor">#FF888888</Color>

<Color x:Key="SelectedBackgroundColor">#FFC5CBF9</Color>
<Color x:Key="SelectedUnfocusedColor">#FFDDDDDD</Color>

<Color x:Key="ControlLightColor">White</Color>
<Color x:Key="ControlMediumColor">#FF7381F9</Color>
<Color x:Key="ControlDarkColor">#FF211AA9</Color>

<Color x:Key="ControlMouseOverColor">#FF3843C4</Color>
<Color x:Key="ControlPressedColor">#FF211AA9</Color>

<Color x:Key="GlyphColor">#FF444444</Color>
<Color x:Key="GlyphMouseOver">sc#1, 0.004391443, 0.002428215, 0.242281124</Color>

<!--Border colors-->
<Color x:Key="BorderLightColor">#FFCCCCCC</Color>
<Color x:Key="BorderMediumColor">#FF888888</Color>
<Color x:Key="BorderDarkColor">#FF444444</Color>

<Color x:Key="PressedBorderLightColor">#FF888888</Color>
<Color x:Key="PressedBorderDarkColor">#FF444444</Color>

```

```

<Color x:Key="DisabledBorderLightColor">#FFAAAAAA</Color>
<Color x:Key="DisabledBorderDarkColor">#FF888888</Color>

<Color x:Key="DefaultBorderBrushDarkColor">Black</Color>

<!--Control-specific resources.-->
<Color x:Key="HeaderTopColor">#FFC5CBF9</Color>
<Color x:Key="DatagridCurrentCellBorderColor">Black</Color>
<Color x:Key="SliderTrackDarkColor">#FFC5CBF9</Color>

<Color x:Key="NavButtonFrameColor">#FF3843C4</Color>

<LinearGradientBrush x:Key="MenuPopupBrush"
    EndPoint="0.5,1"
    StartPoint="0.5,0">
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="0" />
    <GradientStop Color="{DynamicResource ControlMediumColor}"
        Offset="0.5" />
    <GradientStop Color="{DynamicResource ControlLightColor}"
        Offset="1" />
</LinearGradientBrush>

<LinearGradientBrush x:Key="ProgressBarIndicatorAnimatedFill"
    StartPoint="0,0"
    EndPoint="1,0">
    <LinearGradientBrush.GradientStops>
        <GradientStopCollection>
            <GradientStop Color="#000000FF"
                Offset="0" />
            <GradientStop Color="#600000FF"
                Offset="0.4" />
            <GradientStop Color="#600000FF"
                Offset="0.6" />
            <GradientStop Color="#000000FF"
                Offset="1" />
        </GradientStopCollection>
    </LinearGradientBrush.GradientStops>
</LinearGradientBrush>

```

For the complete sample, see [Styling with ControlTemplates Sample](#).

## See also

- [Style](#)
- [ControlTemplate](#)
- [Control Styles and Templates](#)
- [Control Customization](#)
- [Styling and Templating](#)
- [Create a template for a control](#)

# UI Automation of a WPF Custom Control

8 minutes to read • [Edit Online](#)

Microsoft UI Automation provides a single, generalized interface that automation clients can use to examine or operate the user interfaces of a variety of platforms and frameworks. UI Automation enables both quality-assurance (test) code and accessibility applications such as screen readers to examine user-interface elements and simulate user interaction with them from other code. For information about UI Automation across all platforms, see Accessibility.

This topic describes how to implement a server-side UI Automation provider for a custom control that runs in a WPF application. WPF supports UI Automation through a tree of peer automation objects that parallels the tree of user interface elements. Test code and applications that provide accessibility features can use automation peer objects directly (for in-process code) or through the generalized interface provided by UI Automation.

## Automation Peer Classes

WPF controls support UI Automation through a tree of peer classes that derive from [AutomationPeer](#). By convention, peer class names begin with the control class name and end with "AutomationPeer". For example, [ButtonAutomationPeer](#) is the peer class for the [Button](#) control class. The peer classes are roughly equivalent to UI Automation control types but are specific to WPF elements. Automation code that accesses WPF applications through the UI Automation interface does not use automation peers directly, but automation code in the same process space can use automation peers directly.

## Built-in Automation Peer Classes

Elements implement an automation peer class if they accept interface activity from the user, or if they contain information needed by users of screen-reader applications. Not all WPF visual elements have automation peers. Examples of classes that implement automation peers are [Button](#), [TextBox](#), and [Label](#). Examples of classes that do not implement automation peers are classes that derive from [Decorator](#), such as [Border](#), and classes based on [Panel](#), such as [Grid](#) and [Canvas](#).

The base [Control](#) class does not have a corresponding peer class. If you need a peer class to correspond to a custom control that derives from [Control](#), you should derive the custom peer class from [FrameworkElementAutomationPeer](#).

## Security Considerations for Derived Peers

Automation peers must run in a partial-trust environment. Code in the [UIAutomationClient](#) assembly is not configured to run in a partial-trust environment, and automation peer code should not reference that assembly. Instead, you should use the classes in the [UIAutomationTypes](#) assembly. For example, you should use the [AutomationElementIdentifiers](#) class from the [UIAutomationTypes](#) assembly, which corresponds to the [AutomationElement](#) class in the [UIAutomationClient](#) assembly. It is safe to reference the [UIAutomationTypes](#) assembly in automation peer code.

## Peer Navigation

After locating an automation peer, in-process code can navigate the peer tree by calling the object's [GetChildren](#) and [GetParent](#) methods. Navigation among WPF elements within a control is supported by the peer's implementation of the [GetChildrenCore](#) method. The UI Automation system calls this method to build up a tree of subelements contained within a control; for example, list items in a list box. The default

`UIElementAutomationPeer.GetChildrenCore` method traverses the visual tree of elements to build the tree of automation peers. Custom controls override this method to expose children elements to automation clients, returning the automation peers of elements that convey information or allow user interaction.

## Customizations in a Derived Peer

All classes that derive from `UIElement` and `ContentElement` contain the protected virtual method `OnCreateAutomationPeer`. WPF calls `OnCreateAutomationPeer` to get the automation peer object for each control. Automation code can use the peer to get information about a control's characteristics and features and to simulate interactive use. A custom control that supports automation must override `OnCreateAutomationPeer` and return an instance of a class that derives from `AutomationPeer`. For example, if a custom control derives from the `ButtonBase` class, then the object returned by `OnCreateAutomationPeer` should derive from `ButtonBaseAutomationPeer`.

When implementing a custom control, you must override the "Core" methods from the base automation peer class that describe behavior unique and specific to your custom control.

### Override OnCreateAutomationPeer

Override the `OnCreateAutomationPeer` method for your custom control so that it returns your provider object, which must derive directly or indirectly from `AutomationPeer`.

### Override GetPattern

Automation peers simplify some implementation aspects of server-side UI Automation providers, but custom control automation peers must still handle pattern interfaces. Like non-WPF providers, peers support control patterns by providing implementations of interfaces in the `System.Windows.Automation.Provider` namespace, such as `IInvokeProvider`. The control pattern interfaces can be implemented by the peer itself or by another object. The peer's implementation of `GetPattern` returns the object that supports the specified pattern. UI Automation code calls the `GetPattern` method and specifies a `PatternInterface` enumeration value. Your override of `GetPattern` should return the object that implements the specified pattern. If your control does not have a custom implementation of a pattern, you can call the base type's implementation of `GetPattern` to retrieve either its implementation or null if the pattern is not supported for this control type. For example, a custom `NumericUpDown` control can be set to a value within a range, so its UI Automation peer would implement the `IRangeValueProvider` interface. The following example shows how the peer's `GetPattern` method is overridden to respond to a `PatternInterface.RangeValue` value.

```
public override object GetPattern(PatternInterface patternInterface)
{
    if (patternInterface == PatternInterface.RangeValue)
    {
        return this;
    }
    return base.GetPattern(patternInterface);
}
```

```
Public Overrides Function GetPattern(ByVal patternInterface As PatternInterface) As Object
    If patternInterface = PatternInterface.RangeValue Then
        Return Me
    End If
    Return MyBase.GetPattern(patternInterface)
End Function
```

A `GetPattern` method can also specify a subelement as a pattern provider. The following code shows how `ItemsControl` transfers scroll pattern handling to the peer of its internal `ScrollViewer` control.

```

public override object GetPattern(PatternInterface patternInterface)
{
    if (patternInterface == PatternInterface.Scroll)
    {
        ItemsControl owner = (ItemsControl) base.Owner;

        // ScrollHost is internal to the ItemsControl class
        if (owner.ScrollHost != null)
        {
            AutomationPeer peer = UIElementAutomationPeer.CreatePeerForElement(owner.ScrollHost);
            if ((peer != null) && (peer is IScrollProvider))
            {
                peer.EventsSource = this;
                return (IScrollProvider) peer;
            }
        }
    }
    return base.GetPattern(patternInterface);
}

```

```

Public Class Class1
    Public Overrides Function GetPattern(ByVal patternInterface_1 As PatternInterface) As Object
        If patternInterface1 = PatternInterface.Scroll Then
            Dim owner As ItemsControl = DirectCast(MyBase.Owner, ItemsControl)

            ' ScrollHost is internal to the ItemsControl class
            If owner.ScrollHost IsNot Nothing Then
                Dim peer As AutomationPeer = UIElementAutomationPeer.CreatePeerForElement(owner.ScrollHost)
                If (peer IsNot Nothing) AndAlso (TypeOf peer Is IScrollProvider) Then
                    peer.EventsSource = Me
                    Return DirectCast(peer, IScrollProvider)
                End If
            End If
            Return MyBase.GetPattern(patternInterface1)
        End Function
    End Class

```

To specify a subelement for pattern handling, this code gets the subelement object, creates a peer by using the [CreatePeerForElement](#) method, sets the [EventsSource](#) property of the new peer to the current peer, and returns the new peer. Setting [EventsSource](#) on a subelement prevents the subelement from appearing in the automation peer tree and designates all events raised by the subelement as originating from the control specified in [EventsSource](#). The [ScrollViewer](#) control does not appear in the automation tree, and scrolling events that it generates appear to originate from the [ItemsControl](#) object.

### Override "Core" Methods

Automation code gets information about your control by calling public methods of the peer class. To provide information about your control, override each method whose name ends with "Core" when your control implementation differs from that of that provided by the base automation peer class. At a minimum, your control must implement the [GetClassNameCore](#) and [GetAutomationControlTypeCore](#) methods, as shown in the following example.

```

protected override string GetClassNameCore()
{
    return "NumericUpDown";
}

protected override AutomationControlType GetAutomationControlTypeCore()
{
    return AutomationControlType.Spinner;
}

```

```

Protected Overrides Function GetClassNameCore() As String
    Return "NumericUpDown"
End Function

Protected Overrides Function GetAutomationControlTypeCore() As AutomationControlType
    Return AutomationControlType.Spinner
End Function

```

Your implementation of [GetAutomationControlTypeCore](#) describes your control by returning a [ControlType](#) value. Although you can return [ControlType.Custom](#), you should return one of the more specific control types if it accurately describes your control. A return value of [ControlType.Custom](#) requires extra work for the provider to implement UI Automation, and UI Automation client products are unable to anticipate the control structure, keyboard interaction, and possible control patterns.

Implement the [IsContentElementCore](#) and [IsControlElementCore](#) methods to indicate whether your control contains data content or fulfills an interactive role in the user interface (or both). By default, both methods return `true`. These settings improve the usability of automation tools such as screen readers, which may use these methods to filter the automation tree. If your [GetPattern](#) method transfers pattern handling to a subelement peer, the subelement peer's [IsControlElementCore](#) method can return `false` to hide the subelement peer from the automation tree. For example, scrolling in a [ListBox](#) is handled by a [ScrollViewer](#), and the automation peer for [PatternInterface.Scroll](#) is returned by the [GetPattern](#) method of the [ScrollViewerAutomationPeer](#) that is associated with the [ListBoxAutomationPeer](#). Therefore, the [IsControlElementCore](#) method of the [ScrollViewerAutomationPeer](#) returns `false`, so that the [ScrollViewerAutomationPeer](#) does not appear in the automation tree.

Your automation peer should provide appropriate default values for your control. Note that XAML that references your control can override your peer implementations of core methods by including [AutomationProperties](#) attributes. For example, the following XAML creates a button that has two customized UI Automation properties.

```
<Button AutomationProperties.Name="Special"
    AutomationProperties.HelpText="This is a special button."/>
```

## Implement Pattern Providers

The interfaces implemented by a custom provider are explicitly declared if the owning element derives directly from [Control](#). For example, the following code declares a peer for a [Control](#) that implements a range value.

```
public class RangePeer1 : FrameworkElementAutomationPeer, IRangeValueProvider { }
```

```

Public Class RangePeer1
    Inherits FrameworkElementAutomationPeer
    Implements IRangeValueProvider
End Class

```

If the owning control derives from a specific type of control such as [RangeBase](#), the peer can be derived from an

equivalent derived peer class. In this case, the peer would derive from [RangeBaseAutomationPeer](#), which supplies a base implementation of [IRangeValueProvider](#). The following code shows the declaration of such a peer.

```
public class RangePeer2 : RangeBaseAutomationPeer { }
```

```
Public Class RangePeer2
    Inherits RangeBaseAutomationPeer
End Class
```

For an example implementation, see the [C#](#) or [Visual Basic](#) source code that implements and consumes a [NumericUpDown](#) custom control.

## Raise Events

Automation clients can subscribe to automation events. Custom controls must report changes to control state by calling the [RaiseAutomationEvent](#) method. Similarly, when a property value changes, call the [RaisePropertyChangedEvent](#) method. The following code shows how to get the peer object from within the control code and call a method to raise an event. As an optimization, the code determines if there are any listeners for this event type. Raising the event only when there are listeners avoids unnecessary overhead and helps the control remain responsive.

```
if (AutomationPeer.ListenerExists(AutomationEvents.PropertyChanged))
{
    NumericUpDownAutomationPeer peer =
        UIElementAutomationPeer.FromElement(nudCtrl) as NumericUpDownAutomationPeer;

    if (peer != null)
    {
        peer.RaisePropertyChangedEvent(
            RangeValuePatternIdentifiers.ValueProperty,
            (double)oldValue,
            (double)newValue);
    }
}
```

```
If AutomationPeer.ListenerExists(AutomationEvents.PropertyChanged) Then
    Dim peer As NumericUpDownAutomationPeer = TryCast(UIElementAutomationPeer.FromElement(nudCtrl),
    NumericUpDownAutomationPeer)

    If peer IsNot Nothing Then
        peer.RaisePropertyChangedEvent(RangeValuePatternIdentifiers.ValueProperty, CDbL(oldValue),
    CDbL(newValue))
    End If
End If
```

## See also

- [UI Automation Overview](#)
- [Server-Side UI Automation Provider Implementation](#)
- [NumericUpDown custom control \(C#\) on GitHub](#)
- [NumericUpDown custom control \(Visual Basic\) on GitHub](#)

# Walkthroughs: Create a Custom Animated Button

2 minutes to read • [Edit Online](#)

As its name suggests, Windows Presentation Foundation (WPF) is great for making rich presentation experiences for customers. These walkthroughs show you how to customize the look and behavior of a button (including animations). This customization is done using a style and template so that you can apply this custom button easily to any buttons in your application. The following illustration shows the customized button that you will create.



The vector graphics that make up the appearance of the button are created by using Extensible Application Markup Language (XAML). XAML is similar to HTML except it is more powerful and extensible. Extensible Application Markup Language (XAML) can be typed in manually using Visual Studio or Notepad, or you can use a visual design tool such as Blend for Visual Studio. Blend works by creating underlying XAML code, so both methods create the same graphics.

## In This Section

[Create a Button by Using Microsoft Expression Blend](#) Demonstrates how to create buttons with custom behavior by using the designer features of Expression Blend.

[Create a Button by Using XAML](#) Demonstrates how to create buttons with custom behavior by using XAML and Visual Studio.

## Related Sections

[Styling and Templating](#) Describes how styles and templates can be used to determine the appearance and behavior of controls.

[Animation Overview](#) Describes how objects can be animated by using the WPF animation and timing system.

[Painting with Solid Colors and Gradients Overview](#) Describes how to use brush objects to paint with solid colors, linear gradients, and radial gradients.

[Bitmap Effects Overview](#) Describes the bitmap effects supported by WPF and explains how to apply them.

# Walkthrough: Create a Button by Using Microsoft Expression Blend

10 minutes to read • [Edit Online](#)

This walkthrough steps you through the process of creating a WPF customized button using Microsoft Expression Blend.

## IMPORTANT

Microsoft Expression Blend works by generating Extensible Application Markup Language (XAML) that is then compiled to make the executable program. If you would rather work with XAML directly, there is another walkthrough that creates the same application as this one using XAML with Visual Studio rather than Blend. See [Create a Button by Using XAML](#) for more information.

The following illustration shows the customized button that you will create.



## Convert a Shape to a Button

In the first part of this walkthrough you create the custom look of the custom button. To do this, you first convert a rectangle to a button. You then add additional shapes to the template of the button, creating a more complex looking button. Why not start with a regular button and customize it? Because a button has built-in functionality that you do not need; for custom buttons, it is easier to start with a rectangle.

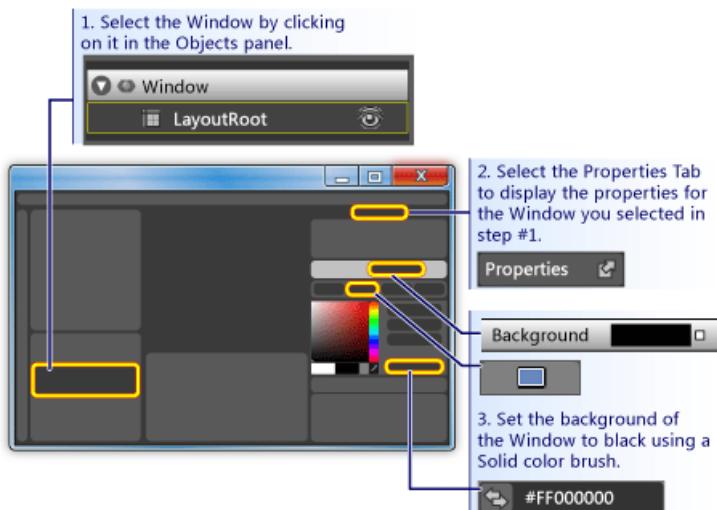
### To create a new project in Expression Blend

1. Start Expression Blend. (Click **Start**, point to **All Programs**, point to **Microsoft Expression**, and then click **Microsoft Expression Blend**.)
2. Maximize the application if needed.
3. On the **File** menu, click **New Project**.
4. Select **Standard Application (.exe)**.
5. Name the project `CustomButton` and press **OK**.

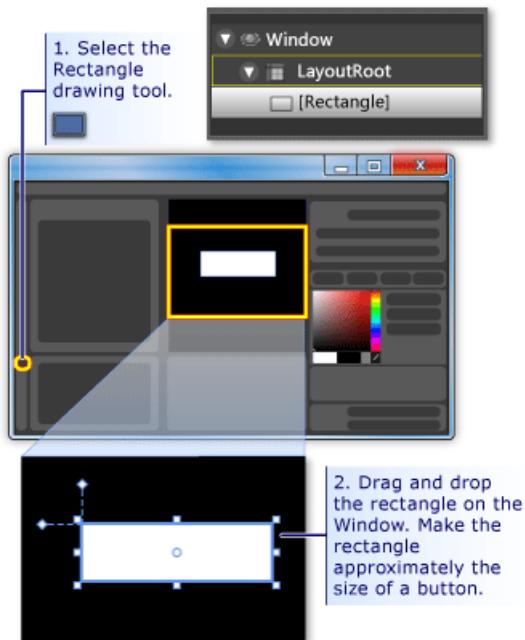
At this point you have a blank WPF project. You can press F5 to run the application. As you might expect, the application consists of only a blank window. Next, you create a rounded rectangle and convert it into a button.

## To convert a Rectangle to a Button

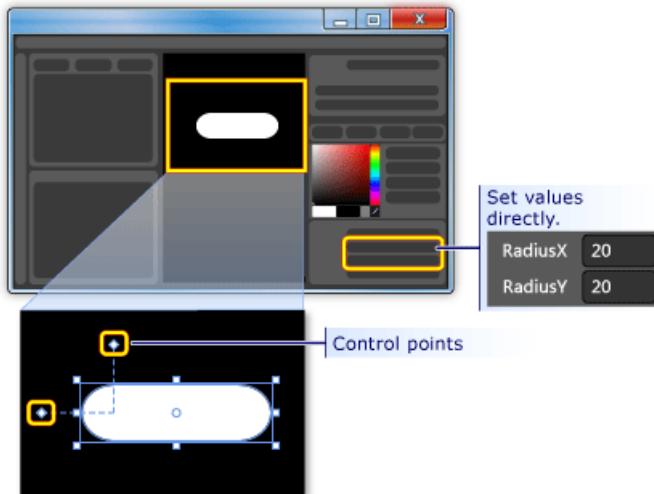
1. Set the Window Background property to black: Select the Window, click the **Properties Tab**, and set the **Background** property to **Black**.



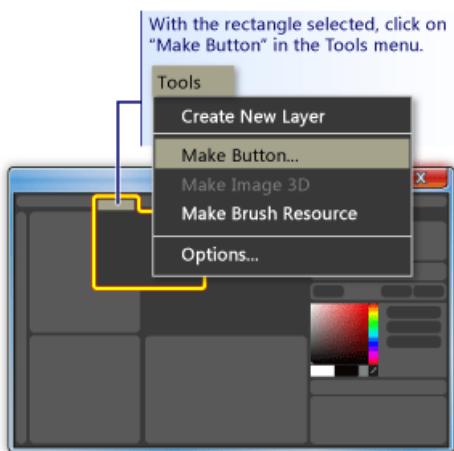
2. Draw a rectangle approximately the size of a button on the Window: Select the rectangle tool on the left-hand tool panel and drag the rectangle onto the Window.



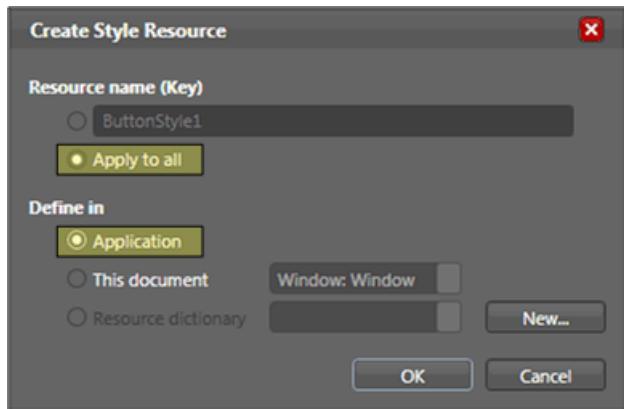
3. Round out the corners of the rectangle: Either drag the control points of the rectangle or directly set the **RadiusX** and **RadiusY** properties. Set the values of **RadiusX** and **RadiusY** to 20.



4. **Change the rectangle into a button:** Select the rectangle. On the **Tools** menu, click **Make Button**.



5. **Specify the scope of the style/template:** A dialog box like the following appears.



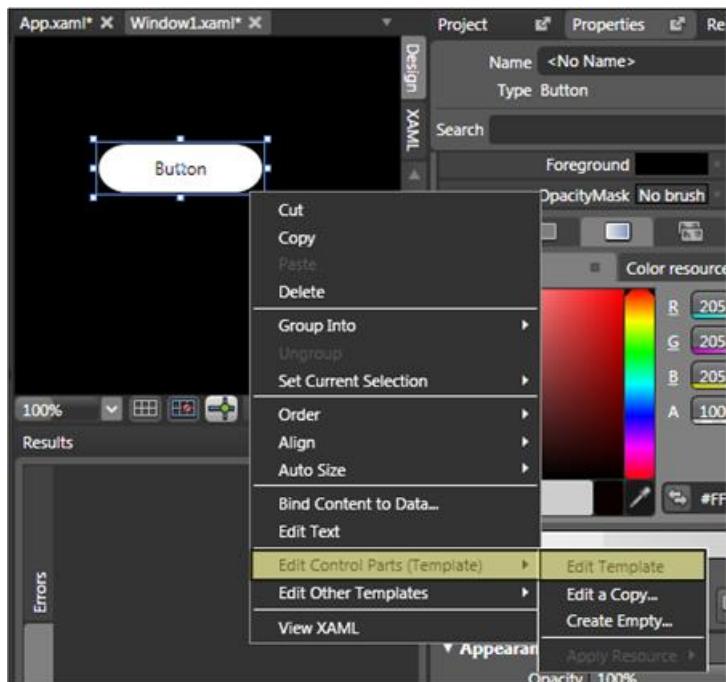
For **Resource name (Key)**, select **Apply to all**. This will make the resulting style and button template apply to all objects that are buttons. For **Define in**, select **Application**. This will make the resulting style and button template have scope over the entire application. When you set the values in these two boxes, the button style and template apply to all buttons within the entire application and any button you create in the application will, by default, use this template.

## Edit the Button Template

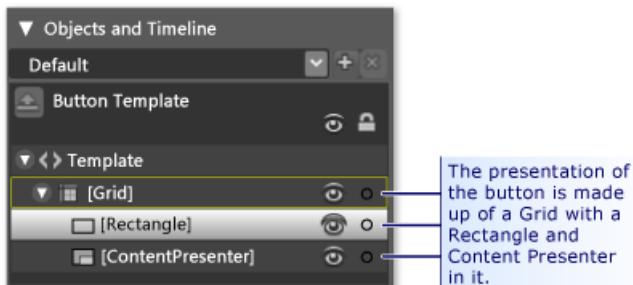
You now have a rectangle that has been changed to a button. In this section, you'll modify the template of the button and further customize how it looks.

### To edit the button template to change the button appearance

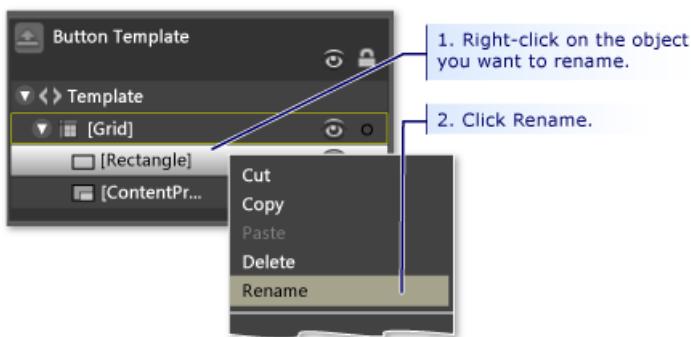
- Go into edit template view:** To further customize the look of our button, we need to edit the button template. This template was created when we converted the rectangle into a button. To edit the button template, right-click the button and select **Edit Control Parts (Template)** and then **Edit Template**.



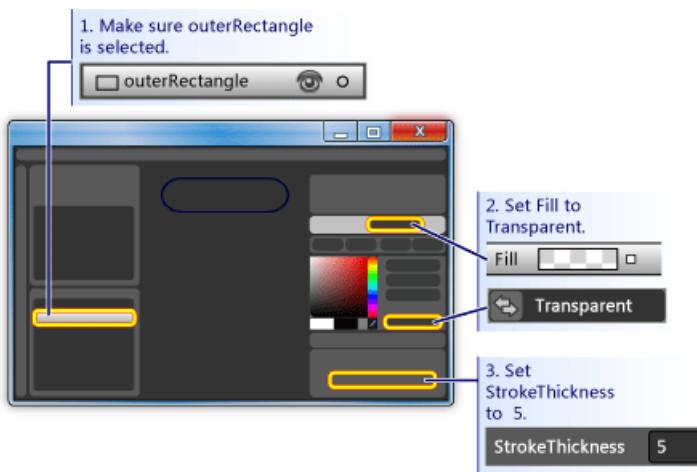
In the template editor, notice that the button is now separated into a [Rectangle](#) and the [ContentPresenter](#). The [ContentPresenter](#) is used to present content within the button (for example, the string "Button"). Both the rectangle and [ContentPresenter](#) are laid out inside of a [Grid](#).



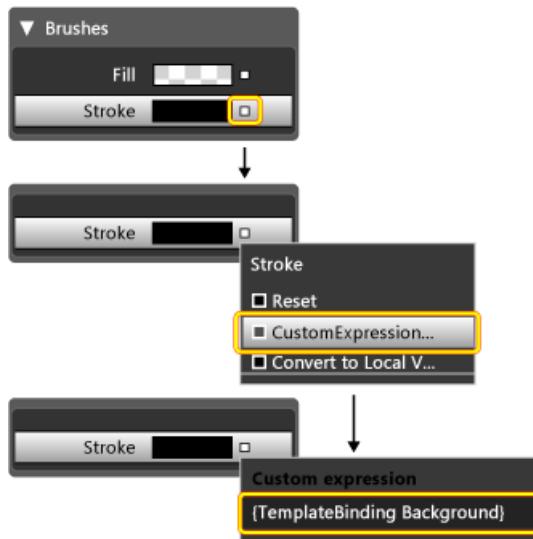
- Change the names of the template components:** Right-click the rectangle in the template inventory, change the [Rectangle](#) name from "[Rectangle]" to "outerRectangle", and change "[ContentPresenter]" to "myContentPresenter".



- Alter the rectangle so that it is empty inside (like a donut):** Select [outerRectangle](#) and set [Fill](#) to "Transparent" and [StrokeThickness](#) to 5.



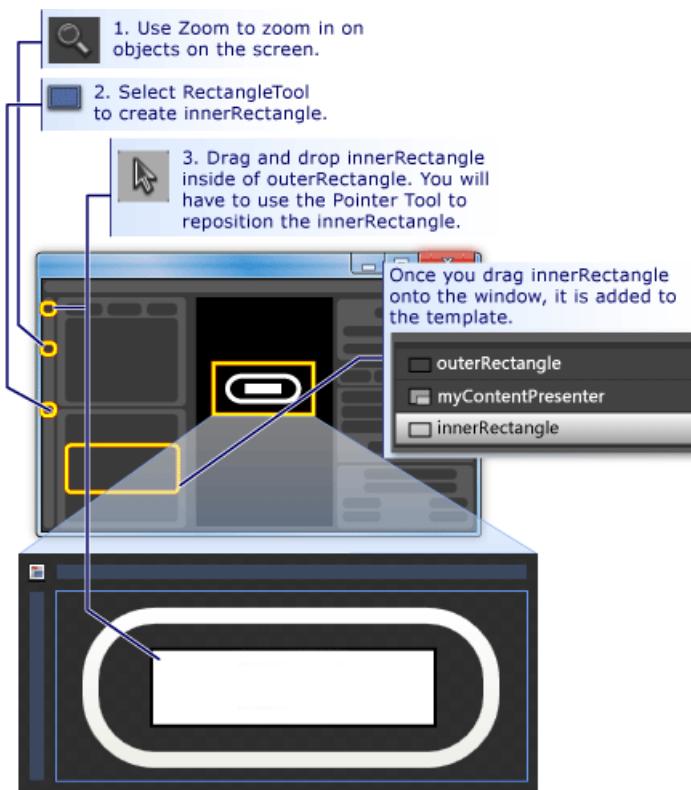
Then set the **Stroke** to the color of whatever the template will be. To do this, click the small white box next to **Stroke**, select **CustomExpression**, and type "{TemplateBinding Background}" in the dialog box.



4. **Create an inner rectangle:** Now, create another rectangle (name it "innerRectangle") and position it symmetrically on the inside of **outerRectangle**. For this kind of work, you will probably want to zoom to make the button larger in the editing area.

#### NOTE

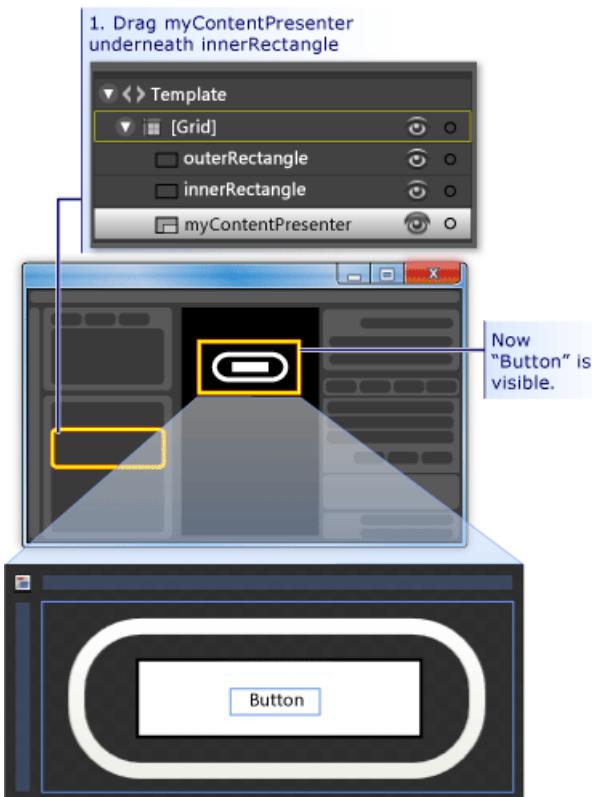
Your rectangle might look different than the one in the figure (for example, it might have rounded corners).



5. **Move ContentPresenter to the top:** At this point, it is possible that the text "Button" will not be visible any longer. If this is so, this is because **innerRectangle** is on top of the **myContentPresenter**. To fix this, drag **myContentPresenter** below **innerRectangle**. Reposition rectangles and **myContentPresenter** to look similar to below.

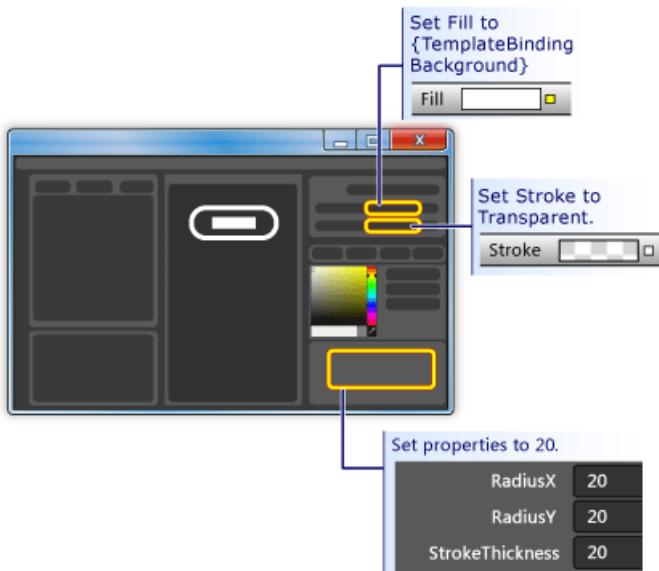
#### NOTE

Alternatively, you can also position **myContentPresenter** on top by right-clicking it and pressing **Send Forward**.

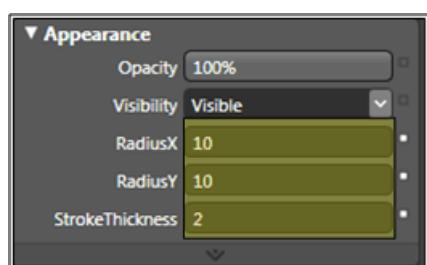


6. **Change the look of innerRectangle:** Set the **RadiusX**, **RadiusY**, and **StrokeThickness** values to 20. In addition, set the **Fill** to the background of the template using the custom expression "{TemplateBinding

Background} ) and set **Stroke** to "transparent". Notice that the settings for the **Fill** and **Stroke** of **innerRectangle** are the opposite of those for **outerRectangle**.



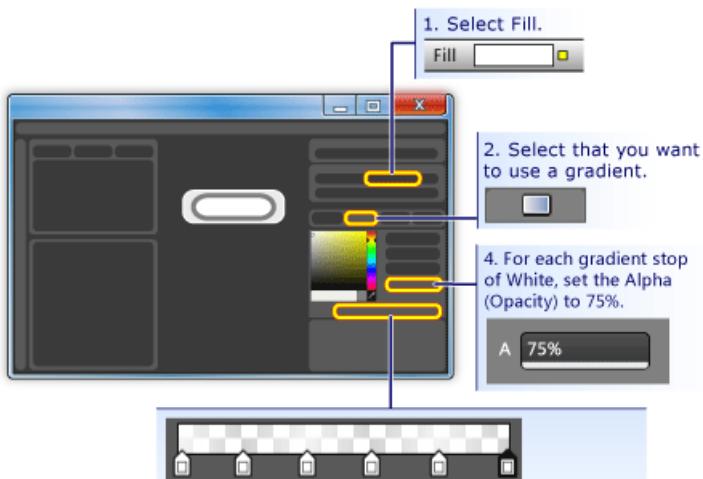
7. **Add a glass layer on top:** The final piece of customizing the look of the button is to add a glass layer on top. This glass layer consists of a third rectangle. Because the glass will cover the entire button, the glass rectangle is similar in dimensions to the **outerRectangle**. Therefore, create the rectangle by simply making a copy of the **outerRectangle**. Highlight **outerRectangle** and use CTRL+C and CTRL+V to make a copy. Name this new rectangle "glassCube".
8. **Reposition glassCube if necessary:** If **glassCube** is not already positioned so that it covers the entire button, drag it into position.
9. **Give glassCube a slightly different shape than outerRectangle:** Change the properties of **glassCube**. Start off by changing the **RadiusX** and **RadiusY** properties to 10 and the **StrokeThickness** to 2.



10. **Make glassCube look like glass:** Set the **Fill** to a glassy look by using a linear gradient that is 75% opaque and alternates between the color White and Transparent over 6 approximately evenly spaced intervals. This is what to set the gradient stops to:

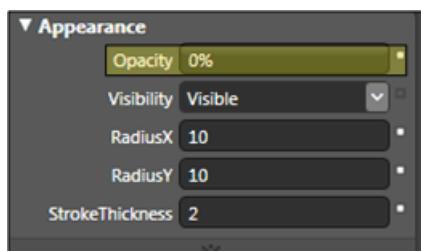
- Gradient Stop 1: White with Alpha value of 75%
- Gradient Stop 2: Transparent
- Gradient Stop 3: White with Alpha value of 75%
- Gradient Stop 4: Transparent
- Gradient Stop 5: White with Alpha value of 75%
- Gradient Stop 6: Transparent

This creates a "wavy" glass look.



3. For this gradient, there are 6 gradient stops alternating between the colors of White and Transparent. You create gradient stops by clicking on the gradient bar. Once you have created the gradient stops, click on each one to set the alternating colors of White and Transparent.

11. **Hide the glass layer:** Now that you see what the glassy layer looks like, go into the **Appearance pane** of the **Properties panel** and set the Opacity to 0% to hide it. In the sections ahead, we'll use property triggers and events to show and manipulate the glass layer.

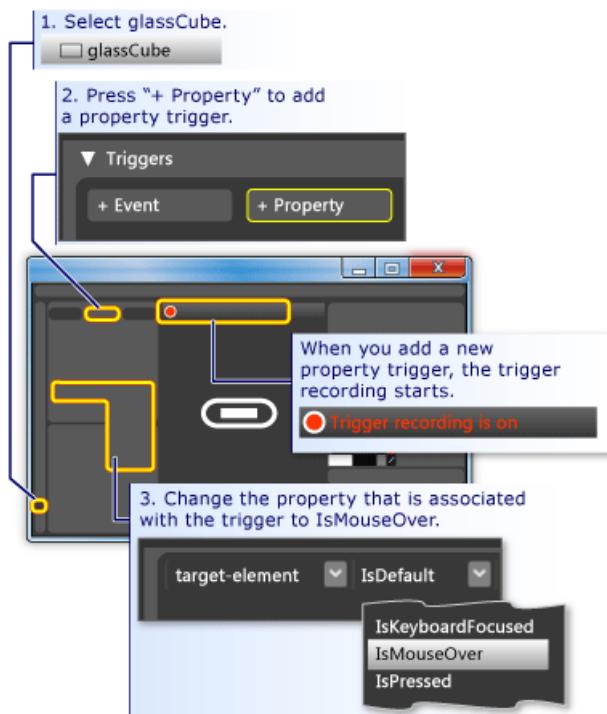


## Customize the Button Behavior

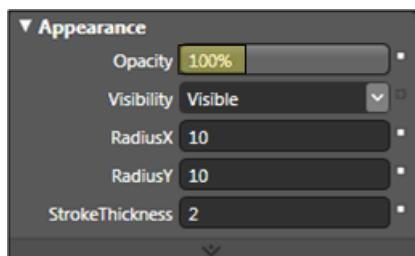
At this point, you have customized the presentation of the button by editing its template, but the button does not react to user actions as typical buttons do (for example, changing appearance upon mouse-over, receiving focus, and clicking.) The next two procedures show how to build behaviors like these into the custom button. We'll start with simple property triggers, and then add event triggers and animations.

### To set property triggers

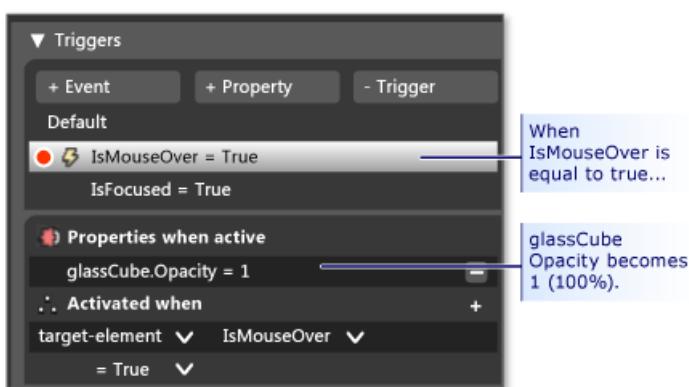
1. **Create a new property trigger:** With **GlassCube** selected, click **+ Property** in the **Triggers** panel (see the figure that follows the next step). This creates a property trigger with a default property trigger.
2. **Make IsMouseOver the property used by the trigger:** Change the property to **IsMouseOver**. This makes the property trigger activate when the **IsMouseOver** property is **true** (when the user points to the button with the mouse).



3. **IsMouseOver triggers opacity of 100% for glassCube:** Notice that the **Trigger recording is on** (see the preceding figure). This means that any changes you make to the property values of **glassCube** while recording is on will become an action that takes place when **IsMouseOver** is `true`. While recording, change the **Opacity** of **glassCube** to 100%.

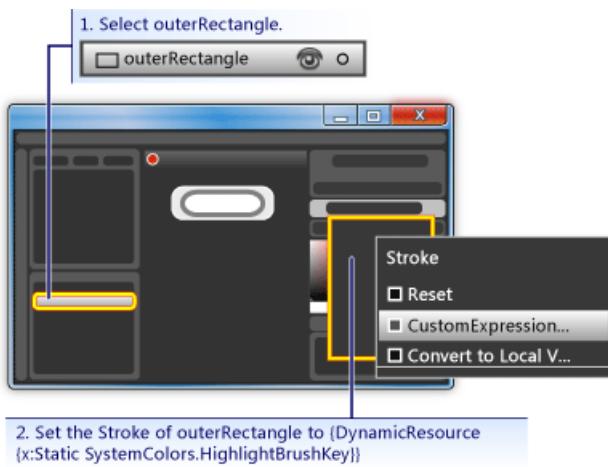


You have now created your first property trigger. Notice that the **Triggers panel** of the editor has recorded the **Opacity** being changed to 100%.

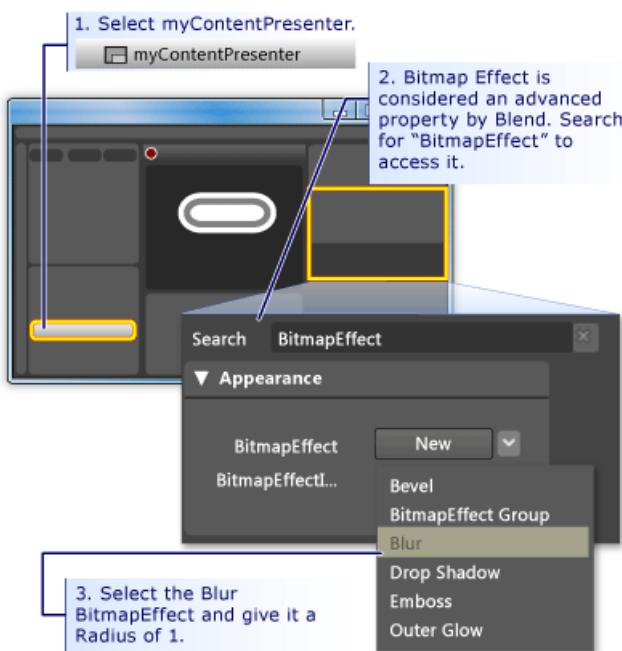


Press F5 to run the application, and move the mouse pointer over and off the button. You should see the glass layer appear when you mouse-over the button and disappear when the pointer leaves.

4. **IsMouseOver triggers stroke value change:** Let's associate some other actions with the **IsMouseOver** trigger. While recording continues, switch your selection from **glassCube** to **outerRectangle**. Then set the **Stroke** of **outerRectangle** to the custom expression of "`{DynamicResource {x:Static SystemColors.HighlightBrushKey}}`". This sets the **Stroke** to the typical highlight color used by buttons. Press F5 to see the effect when you mouse over the button.



5. **IsMouseOver triggers blurry text:** Let's associate one more action to the `IsMouseOver` property trigger. Make the content of the button appear a little blurry when the glass appears over it. To do this, we can apply a blur `BitmapEffect` to the `ContentPresenter` (`myContentPresenter`).



#### NOTE

To return the **Properties panel** back to what it was before you did the search for `BitmapEffect`, clear the text from the **Search box**.

At this point, we have used a property trigger with several associated actions to create highlighting behavior for when the mouse pointer enters and leaves the button area. Another typical behavior for a button is to highlight when it has focus (as after it is clicked). We can add such behavior by adding another property trigger for the `IsFocused` property.

6. **Create property trigger for IsFocused:** Using the same procedure as for `IsMouseOver` (see the first step of this section), create another property trigger for the `IsFocused` property. While **Trigger recording is on**, add the following actions to the trigger:

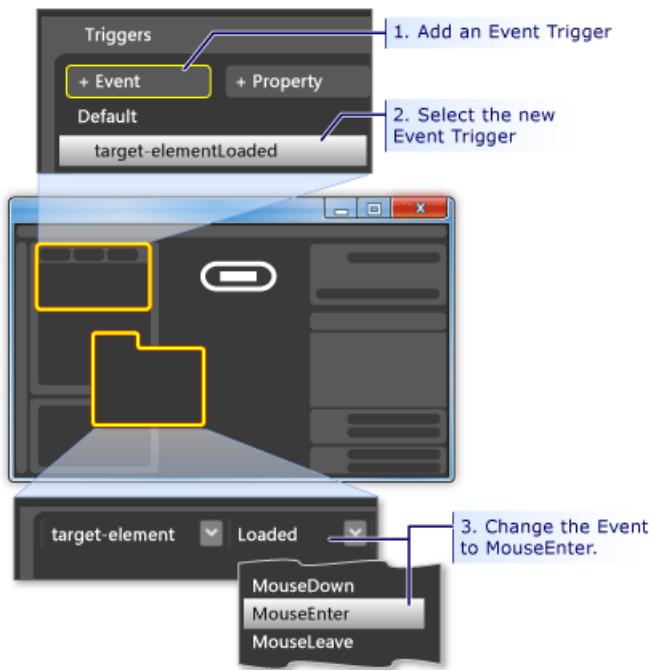
- **glassCube** gets an `Opacity` of 100%.
- **outerRectangle** gets a `Stroke` custom expression value of "`{DynamicResource {x:Static SystemColors.HighlightBrushKey}}`".

As the final step in this walkthrough, we will add animations to the button. These animations will be triggered by

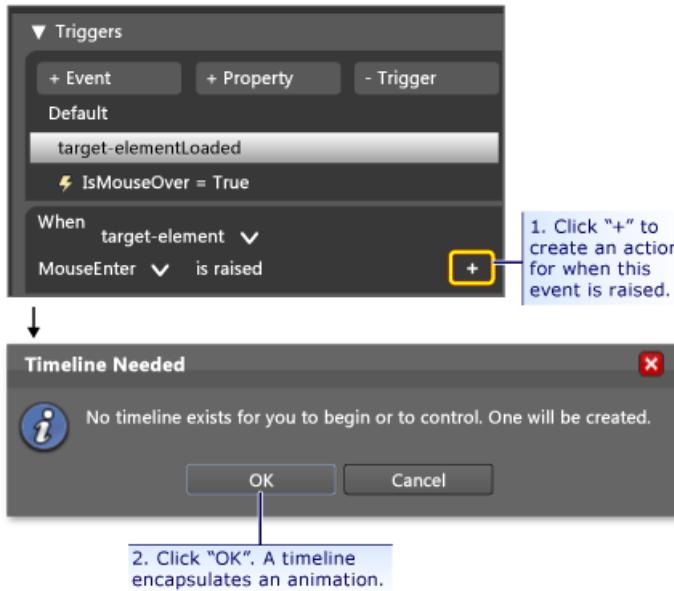
events—specifically, the [MouseEnter](#) and [Click](#) events.

## To use event triggers and animations to add interactivity

1. **Create a MouseEnter Event Trigger:** Add a new event trigger and select [MouseEnter](#) as the event to use in the trigger.



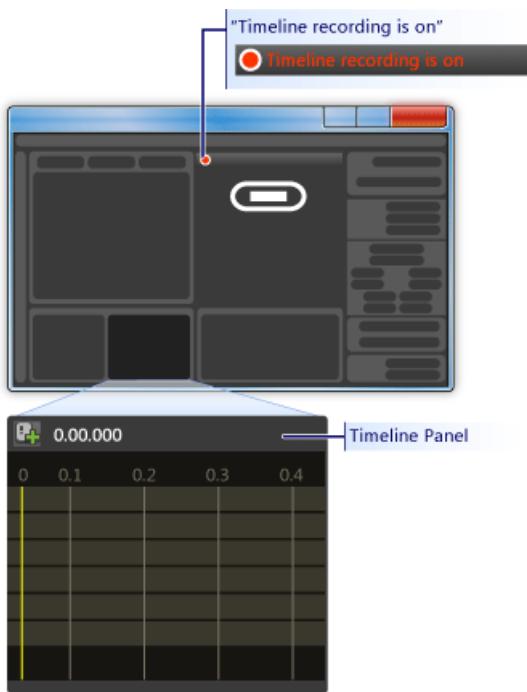
2. **Create an animation timeline:** Next, associate an animation timeline to the [MouseEnter](#) event.



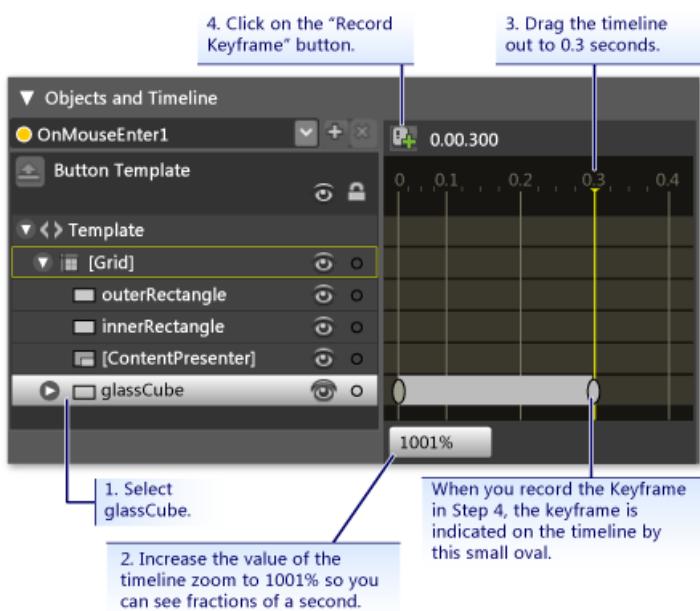
After you press **OK** to create a new timeline, a **Timeline Panel** appears and "Timeline recording is on" is visible in the design panel. This means we can start recording property changes in the timeline (animate property changes).

### NOTE

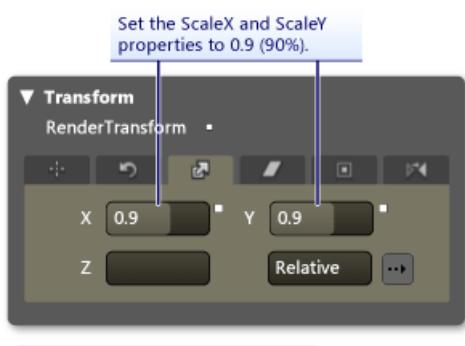
You may need to resize your window and/or panels to see the display.



- Create a keyframe:** To create an animation, select the object you want to animate, create two or more keyframes on the timeline, and for those keyframes, set the property values you want the animation to interpolate between. The following figure guides you through the creation of a keyframe.



- Shrink glassCube at this keyframe:** With the second keyframe selected, shrink the size of the **glassCube** to 90% of its full size using the **Size Transform**.

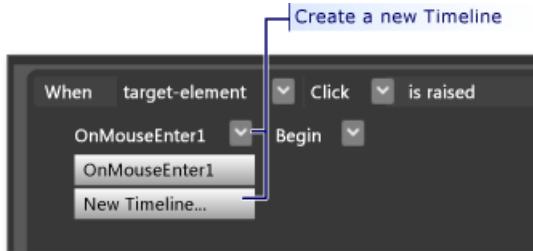


Press F5 to run the application. Move the mouse pointer over the button. Notice that the glass layer shrinks on top of the button.

5. **Create another Event Trigger and associate a different animation with it:** Let's add one more animation. Use a similar procedure to what you used to create the previous event trigger animation:

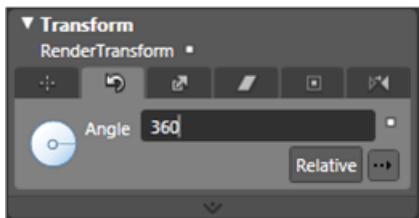
a. Create a new event trigger using the [Click](#) event.

b. Associate a new timeline with the [Click](#) event.



c. For this timeline, create two keyframes, one at 0.0 seconds and the second one at 0.3 seconds.

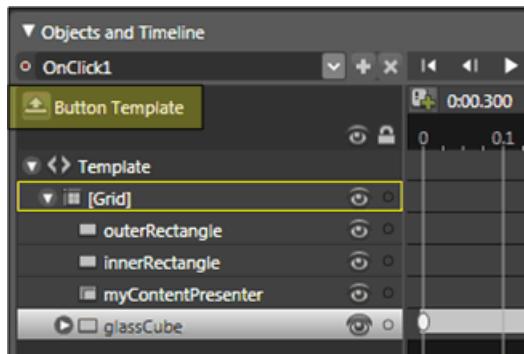
d. With the keyframe at 0.3 seconds highlighted, set the **Rotate Transform Angle** to 360 degrees.

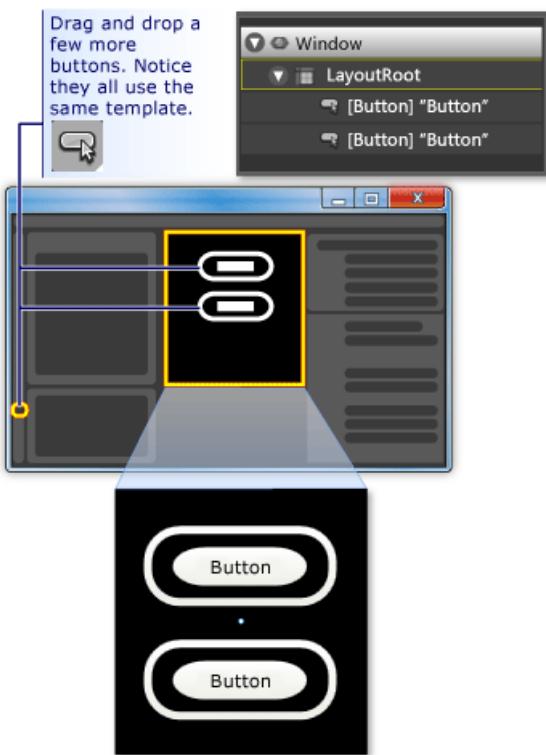


e. Press F5 to run the application. Click the button. Notice that the glass layer spins around.

## Conclusion

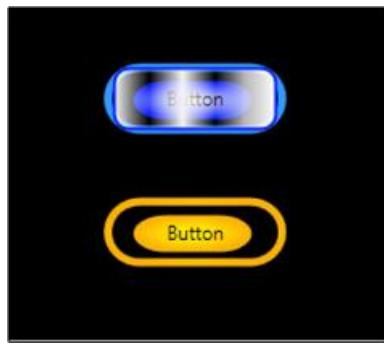
You have completed a customized button. You did this using a button template that was applied to all buttons in the application. If you leave the template editing mode (see the following figure) and create more buttons, you will see that they look and behave like your custom button rather than like the default button.





Press F5 to run the application. Click the buttons and notice how they all behave the same.

Remember that while you were customizing the template, you set the **Fill** property of **innerRectangle** and the **Stroke** property **outerRectangle** to the template background (**{TemplateBinding Background}**). Because of this, when you set the background color of the individual buttons, the background you set will be used for those respective properties. Try changing the backgrounds now. In the following figure, different gradients are used. Therefore, although a template is useful for overall customization of controls like button, controls with templates can still be modified to look different from each other.



In conclusion, in the process of customizing a button template you have learned how to do the following in Microsoft Expression Blend:

- Customize the look of a control.
- Set property triggers. Property triggers are very useful because they can be used on most objects, not just controls.
- Set event triggers. Event triggers are very useful because they can be used on most objects, not just controls.
- Create animations.
- Miscellaneous: create gradients, add BitmapEffects, use transforms, and set basic properties of objects.

## See also

- [Create a Button by Using XAML](#)
- [Styling and Templating](#)
- [Animation Overview](#)
- [Painting with Solid Colors and Gradients Overview](#)
- [Bitmap Effects Overview](#)

# Walkthrough: Create a Button by Using XAML

12 minutes to read • [Edit Online](#)

The objective of this walkthrough is to learn how to create an animated button for use in a Windows Presentation Foundation (WPF) application. This walkthrough uses styles and a template to create a customized button resource that allows reuse of code and separation of button logic from the button declaration. This walkthrough is written entirely in Extensible Application Markup Language (XAML).

## IMPORTANT

This walkthrough guides you through the steps for creating the application by typing or copying and pasting Extensible Application Markup Language (XAML) into Visual Studio. If you would prefer to learn how to use a designer to create the same application, see [Create a Button by Using Microsoft Expression Blend](#).

The following figure shows the finished buttons.



## Create Basic Buttons

Let's start by creating a new project and adding a few buttons to the window.

### To create a new WPF project and add buttons to the window

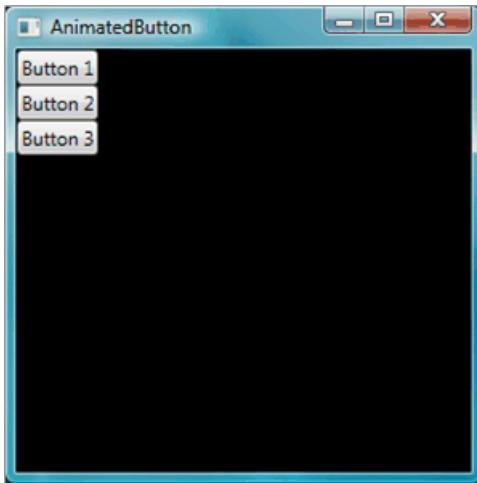
1. Start Visual Studio.
2. **Create a new WPF project:** On the **File** menu, point to **New**, and then click **Project**. Find the **Windows Application (WPF)** template and name the project "AnimatedButton". This will create the skeleton for the application.
3. **Add basic default buttons:** All the files you need for this walkthrough are provided by the template. Open the Window1.xaml file by double clicking it in Solution Explorer. By default, there is a **Grid** element in Window1.xaml. Remove the **Grid** element and add a few buttons to the Extensible Application Markup Language (XAML) page by typing or copy and pasting the following highlighted code to Window1.xaml:

```

<Window x:Class="AnimatedButton.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="AnimatedButton" Height="300" Width="300"
    Background="Black">
    <!-- Buttons arranged vertically inside a StackPanel. -->
    <StackPanel HorizontalAlignment="Left">
        <Button>Button 1</Button>
        <Button>Button 2</Button>
        <Button>Button 3</Button>
    </StackPanel>
</Window>

```

Press F5 to run the application; you should see a set of buttons that looks like the following figure.



Now that you have created the basic buttons, you are finished working in the Window1.xaml file. The rest of the walkthrough focuses on the app.xaml file, defining styles and a template for the buttons.

## Set Basic Properties

Next, let's set some properties on these buttons to control the button appearance and layout. Rather than setting properties on the buttons individually, you will use resources to define button properties for the entire application. Application resources are conceptually similar to external Cascading Style Sheets (CSS) for Web pages; however, resources are much more powerful than Cascading Style Sheets (CSS), as you will see by the end of this walkthrough. To learn more about resources, see [XAML Resources](#).

### To use styles to set basic properties on the buttons

- Define an Application.Resources block:** Open app.xaml and add the following highlighted markup if it is not already there:

```

<Application x:Class="AnimatedButton.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    StartupUri="Window1.xaml"
    >
    <Application.Resources>
        <!-- Resources for the entire application can be defined here. -->
    </Application.Resources>
</Application>

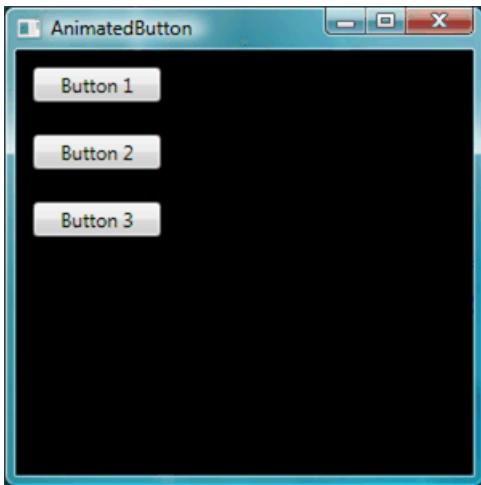
```

Resource scope is determined by where you define the resource. Defining resources in `Application.Resources` in the app.xaml file enables the resource to be used from anywhere in the application. To learn more about defining the scope of your resources, see [XAML Resources](#).

2. **Create a style and define basic property values with it:** Add the following markup to the `Application.Resources` block. This markup creates a [Style](#) that applies to all buttons in the application, setting the [Width](#) of the buttons to 90 and the [Margin](#) to 10:

```
<Application.Resources>
<Style TargetType="Button">
    <Setter Property="Width" Value="90" />
    <Setter Property="Margin" Value="10" />
</Style>
</Application.Resources>
```

The [TargetType](#) property specifies that the style applies to all objects of type [Button](#). Each [Setter](#) sets a different property value for the [Style](#). Therefore, at this point every button in the application has a width of 90 and a margin of 10. If you press F5 to run the application, you see the following window.



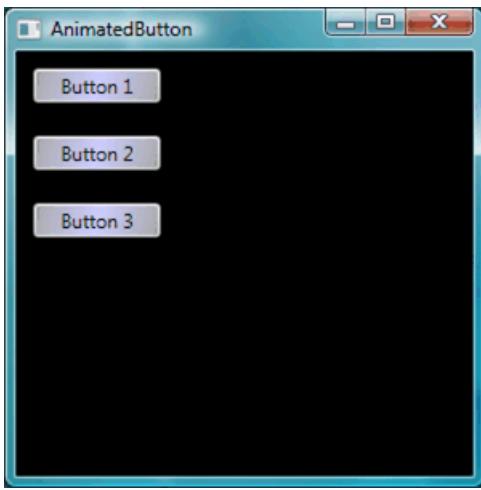
There is much more you can do with styles, including a variety of ways to fine-tune what objects are targeted, specifying complex property values, and even using styles as input for other styles. For more information, see [Styling and Templating](#).

3. **Set a style property value to a resource:** Resources enable a simple way to reuse commonly defined objects and values. It is especially useful to define complex values using resources to make your code more modular. Add the following highlighted markup to app.xaml.

```
<Application.Resources>
<LinearGradientBrush x:Key="GrayBlueGradientBrush" StartPoint="0,0" EndPoint="1,1">
    <GradientStop Color="DarkGray" Offset="0" />
    <GradientStop Color="#CCCCFF" Offset="0.5" />
    <GradientStop Color="DarkGray" Offset="1" />
</LinearGradientBrush>
<Style TargetType="{x:Type Button}">
    <Setter Property="Background" Value="{StaticResource GrayBlueGradientBrush}" />
    <Setter Property="Width" Value="80" />
    <Setter Property="Margin" Value="10" />
</Style>
</Application.Resources>
```

Directly under the `Application.Resources` block, you created a resource called "GrayBlueGradientBrush". This resource defines a horizontal gradient. This resource can be used as a property value from anywhere in the application, including inside the button style setter for the [Background](#) property. Now, all the buttons have a [Background](#) property value of this gradient.

Press F5 to run the application. It should look like the following.



## Create a Template That Defines the Look of the Button

In this section, you create a template that customizes the appearance (presentation) of the button. The button presentation is made up of several objects including rectangles and other components to give the button a unique look.

So far, the control of how buttons look in the application has been confined to changing properties of the button. What if you want to make more radical changes to the button's appearance? Templates enable powerful control over the presentation of an object. Because templates can be used within styles, you can apply a template to all objects that the style applies to (in this walkthrough, the button).

### To use the template to define the look of the button

1. **Set up the template:** Because controls like [Button](#) have a [Template](#) property, you can define the template property value just like the other property values we have set in a [Style](#) using a [Setter](#). Add the following highlighted markup to your button style.

```
<Application.Resources>
    <LinearGradientBrush x:Key="GrayBlueGradientBrush"
        StartPoint="0,0" EndPoint="1,1">
        <GradientStop Color="DarkGray" Offset="0" />
        <GradientStop Color="#CCCCFF" Offset="0.5" />
        <GradientStop Color="DarkGray" Offset="1" />
    </LinearGradientBrush>
    <Style TargetType="{x:Type Button}">
        <Setter Property="Background" Value="{StaticResource GrayBlueGradientBrush}" />
        <Setter Property="Width" Value="80" />
        <Setter Property="Margin" Value="10" />
        <Setter Property="Template">
            <Setter.Value>
                <!-- The button template is defined here. -->
            </Setter.Value>
        </Setter>
    </Style>
</Application.Resources>
```

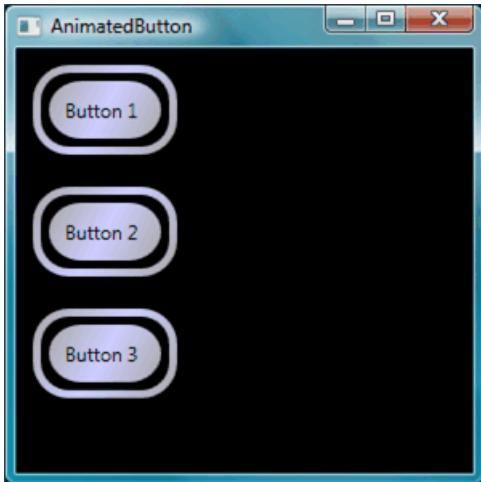
2. **Alter button presentation:** At this point, you need to define the template. Add the following highlighted markup. This markup specifies two [Rectangle](#) elements with rounded edges, followed by a [DockPanel](#). The [DockPanel](#) is used to host the [ContentPresenter](#) of the button. A [ContentPresenter](#) displays the content of the button. In this walkthrough, the content is text ("Button 1", "Button 2", "Button 3"). All of the template components (the rectangles and the [DockPanel](#)) are laid out inside of a [Grid](#).

```

<Setter.Value>
  <ControlTemplate TargetType="Button">
    <Grid Width="{TemplateBinding Width}" Height="{TemplateBinding Height}" ClipToBounds="True">
      <!-- Outer Rectangle with rounded corners. -->
      <Rectangle x:Name="outerRectangle" HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
      Stroke="{TemplateBinding Background}" RadiusX="20" RadiusY="20" StrokeThickness="5" Fill="Transparent"
      />
      <!-- Inner Rectangle with rounded corners. -->
      <Rectangle x:Name="innerRectangle" HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
      Stroke="Transparent" StrokeThickness="20" Fill="{TemplateBinding Background}" RadiusX="20" RadiusY="20"
      />
      <!-- Present Content (text) of the button. -->
      <ContentPresenter x:Name="myContentPresenter" Margin="20" Content="{TemplateBinding Content}"
      TextBlock.Foreground="Black" />
    </ControlTemplate>
  </Setter.Value>

```

Press F5 to run the application. It should look like the following.



3. **Add a glass effect to the template:** Next you will add the glass. First you create some resources that create a glass gradient effect. Add these gradient resources anywhere within the `Application.Resources` block:

```

<Application.Resources>
  <GradientStopCollection x:Key="MyGlassGradientStopsResource">
    <GradientStop Color="WhiteSmoke" Offset="0.2" />
    <GradientStop Color="Transparent" Offset="0.4" />
    <GradientStop Color="WhiteSmoke" Offset="0.5" />
    <GradientStop Color="Transparent" Offset="0.75" />
    <GradientStop Color="WhiteSmoke" Offset="0.9" />
    <GradientStop Color="Transparent" Offset="1" />
  </GradientStopCollection>
  <LinearGradientBrush x:Key="MyGlassBrushResource"
    StartPoint="0,0" EndPoint="1,1" Opacity="0.75"
    GradientStops="{StaticResource MyGlassGradientStopsResource}" />
  <!-- Styles and other resources below here. -->

```

These resources are used as the `Fill` for a rectangle that we insert into the `Grid` of the button template. Add the following highlighted markup to the template.

```

<Setter.Value>
<ControlTemplate TargetType="{x:Type Button}">
    <Grid Width="{TemplateBinding Width}" Height="{TemplateBinding Height}">
        <ClipToBounds>True</ClipToBounds>

        <!-- Outer Rectangle with rounded corners. -->
        <Rectangle x:Name="outerRectangle" HorizontalAlignment="Stretch"
            VerticalAlignment="Stretch" Stroke="{TemplateBinding Background}"
            RadiusX="20" RadiusY="20" StrokeThickness="5" Fill="Transparent" />

        <!-- Inner Rectangle with rounded corners. -->
        <Rectangle x:Name="innerRectangle" HorizontalAlignment="Stretch"
            VerticalAlignment="Stretch" Stroke="Transparent" StrokeThickness="20"
            Fill="{TemplateBinding Background}" RadiusX="20" RadiusY="20" />

        <!-- Glass Rectangle -->
        <Rectangle x:Name="glassCube" HorizontalAlignment="Stretch"
            VerticalAlignment="Stretch"
            StrokeThickness="2" RadiusX="10" RadiusY="10" Opacity="0"
            Fill="{StaticResource MyGlassBrushResource}"
            RenderTransformOrigin="0.5,0.5">
            <Rectangle.Stroke>
                <LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">
                    <LinearGradientBrush.GradientStops>
                        <GradientStop Offset="0.0" Color="LightBlue" />
                        <GradientStop Offset="1.0" Color="Gray" />
                    </LinearGradientBrush.GradientStops>
                </LinearGradientBrush>
            </Rectangle.Stroke>
            <!-- These transforms have no effect as they are declared here.
            The reason the transforms are included is to be targets
            for animation (see later). -->
            <Rectangle.RenderTransform>
                <TransformGroup>
                    <ScaleTransform />
                    <RotateTransform />
                </TransformGroup>
            </Rectangle.RenderTransform>
            <!-- A BevelBitmapEffect is applied to give the button a "Beveled" look. -->
            <Rectangle.BitmapEffect>
                <BevelBitmapEffect />
            </Rectangle.BitmapEffect>
        </Rectangle>

        <!-- Present Text of the button. -->
        <DockPanel Name="myContentPresenterDockPanel">
            <ContentPresenter x:Name="myContentPresenter" Margin="20"
                Content="{TemplateBinding Content}" TextBlock.Foreground="Black" />
        </DockPanel>
    </Grid>
</ControlTemplate>
</Setter.Value>

```

Notice that the **Opacity** of the rectangle with the `x:Name` property of "glassCube" is 0, so when you run the sample, you do not see the glass rectangle overlaid on top. This is because we will later add triggers to the template for when the user interacts with the button. However, you can see what the button looks like now by changing the **Opacity** value to 1 and running the application. See the following figure. Before proceeding to the next step, change the **Opacity** back to 0.



## Create Button Interactivity

In this section, you will create property triggers and event triggers to change property values and run animations in response to user actions such as moving the mouse pointer over the button and clicking.

An easy way to add interactivity (mouse-over, mouse-leave, click, and so on) is to define triggers within your template or style. To create a [Trigger](#), you define a property "condition" such as: The button `IsMouseOver` property value is equal to `true`. You then define setters (actions) that take place when the trigger condition is true.

### To create button interactivity

1. **Add template triggers:** Add the highlighted markup to your template.

```

<Setter.Value>
<ControlTemplate TargetType="{x:Type Button}">
    <Grid Width="{TemplateBinding Width}"
          Height="{TemplateBinding Height}" ClipToBounds="True">

        <!-- Outer Rectangle with rounded corners. -->
        <Rectangle x:Name="outerRectangle" HorizontalAlignment="Stretch"
                   VerticalAlignment="Stretch" Stroke="{TemplateBinding Background}"
                   RadiusX="20" RadiusY="20" StrokeThickness="5" Fill="Transparent" />

        <!-- Inner Rectangle with rounded corners. -->
        <Rectangle x:Name="innerRectangle" HorizontalAlignment="Stretch"
                   VerticalAlignment="Stretch" Stroke="Transparent"
                   StrokeThickness="20"
                   Fill="{TemplateBinding Background}" RadiusX="20" RadiusY="20"
        />

        <!-- Glass Rectangle -->
        <Rectangle x:Name="glassCube" HorizontalAlignment="Stretch"
                   VerticalAlignment="Stretch"
                   StrokeThickness="2" RadiusX="10" RadiusY="10" Opacity="0"
                   Fill="{StaticResource MyGlassBrushResource}"
                   RenderTransformOrigin="0.5,0.5">
            <Rectangle.Stroke>
                <LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">
                    <LinearGradientBrush.GradientStops>
                        <GradientStop Offset="0.0" Color="LightBlue" />
                        <GradientStop Offset="1.0" Color="Gray" />
                    </LinearGradientBrush.GradientStops>
                </LinearGradientBrush>
            </Rectangle.Stroke>
        </Rectangle>

        <!-- These transforms have no effect as they
             are declared here.
             The reason the transforms are included is to be targets
             for animation (see later). -->
        <Rectangle.RenderTransform>
            <TransformGroup>
                <ScaleTransform />
                <RotateTransform />
            </TransformGroup>
        </Rectangle.RenderTransform>

        <!-- A BevelBitmapEffect is applied to give the button a
             "Beveled" look. -->
        <Rectangle.BitmapEffect>
            <BevelBitmapEffect />
        </Rectangle.BitmapEffect>
    </Rectangle>

    <!-- Present Text of the button. -->
    <DockPanel Name="myContentPresenterDockPanel">
        <ContentPresenter x:Name="myContentPresenter" Margin="20"
                          Content="{TemplateBinding Content}" TextBlock.Foreground="Black" />
    </DockPanel>
</Grid>

<ControlTemplate.Triggers>      <!-- Set action triggers for the buttons and define
what the button does in response to those triggers. -->      </ControlTemplate.Triggers>
</ControlTemplate>
</Setter.Value>

```

2. **Add property triggers:** Add the highlighted markup to the `ControlTemplate.Triggers` block:

```

<ControlTemplate.Triggers>

    <!-- Set properties when mouse pointer is over the button. --> <Trigger Property="IsMouseOver"
Value="True">      <!-- Below are three property settings that occur when the condition is met
(user mouses over button). -->      <!-- Change the color of the outer rectangle when user
mouses over it. -->      <Setter Property ="Rectangle.Stroke" TargetName="outerRectangle"      Value="
{DynamicResource {x:Static SystemColors.HighlightBrushKey}}" />      <!-- Sets the glass opacity to 1,
therefore, the glass "appears" when user mouses over it. -->      <Setter
Property="Rectangle.Opacity" Value="1" TargetName="glassCube" />      <!-- Makes the text slightly
blurry as though you were looking at it through blurry glass. -->      <Setter
Property="ContentPresenter.BitmapEffect"          TargetName="myContentPresenter">      <Setter.Value>
<BlurBitmapEffect Radius="1" />      </Setter.Value>      </Setter>      </Trigger>

<ControlTemplate.Triggers/>

```

Press F5 to run the application and see the effect as you run the mouse pointer over the buttons.

3. **Add a focus trigger:** Next, we'll add some similar setters to handle the case when the button has focus (for example, after the user clicks it).

```

<ControlTemplate.Triggers>

    <!-- Set properties when mouse pointer is over the button. -->
<Trigger Property="IsMouseOver" Value="True">

        <!-- Below are three property settings that occur when the
            condition is met (user mouses over button). -->
        <!-- Change the color of the outer rectangle when user mouses over it. -->
        <Setter Property ="Rectangle.Stroke" TargetName="outerRectangle"
Value="{DynamicResource {x:Static SystemColors.HighlightBrushKey}}" />

        <!-- Sets the glass opacity to 1, therefore, the glass "appears" when user mouses over it.
-->
        <Setter Property="Rectangle.Opacity" Value="1"      TargetName="glassCube" />

        <!-- Makes the text slightly blurry as though you were looking at it through blurry glass.
-->
        <Setter Property="ContentPresenter.BitmapEffect"      TargetName="myContentPresenter">
            <Setter.Value>
                <BlurBitmapEffect Radius="1" />
            </Setter.Value>
        </Setter>
    </Trigger>

    <!-- Set properties when button has focus. --> <Trigger Property="IsFocused" Value="true">
<Setter Property="Rectangle.Opacity" Value="1"      TargetName="glassCube" />      <Setter
Property="Rectangle.Stroke" TargetName="outerRectangle"      Value="{DynamicResource {x:Static
SystemColors.HighlightBrushKey}}" />      <Setter Property="Rectangle.Opacity" Value="1"
TargetName="glassCube" />      </Trigger>

</ControlTemplate.Triggers>

```

Press F5 to run the application and click on one of the buttons. Notice that the button stays highlighted after you click it because it still has focus. If you click another button, the new button gains focus while the last one loses it.

4. **Add animations for MouseEnter and MouseLeave:** Next we add some animations to the triggers. Add the following markup anywhere inside of the `ControlTemplate.Triggers` block.

```

<!-- Animations that start when mouse enters and leaves button. -->
<EventTrigger RoutedEvent="Mouse.MouseEnter">
    <EventTrigger.Actions>
        <BeginStoryboard Name="mouseEnterBeginStoryboard">
            <Storyboard>
                <!-- This animation makes the glass rectangle shrink in the X direction. -->
                <DoubleAnimation Storyboard.TargetName="glassCube"
                    Storyboard.TargetProperty=
                    "(Rectangle.RenderTransform).(TransformGroup.Children)[0].(ScaleTransform.ScaleX)"
                    By="-0.1" Duration="0:0:0.5" />
                <!-- This animation makes the glass rectangle shrink in the Y direction. -->
                <DoubleAnimation
                    Storyboard.TargetName="glassCube"
                    Storyboard.TargetProperty=
                    "(Rectangle.RenderTransform).(TransformGroup.Children)[0].(ScaleTransform.ScaleY)"
                    By="-0.1" Duration="0:0:0.5" />
            </Storyboard>
        </BeginStoryboard>
    </EventTrigger.Actions>
</EventTrigger>
<EventTrigger RoutedEvent="Mouse.MouseLeave">
    <EventTrigger.Actions>
        <!-- Stopping the storyboard sets all animated properties back to default. -->
        <StopStoryboard BeginStoryboardName="mouseEnterBeginStoryboard" />
    </EventTrigger.Actions>
</EventTrigger>

```

The glass rectangle shrinks when the mouse pointer moves over the button and returns back to normal size when the pointer leaves.

There are two animations that are triggered when the pointer goes over the button ([MouseEnter](#) event is raised). These animations shrink the glass rectangle along the X and Y axis. Notice the properties on the [DoubleAnimation](#) elements — [Duration](#) and [By](#). The [Duration](#) specifies that the animation occurs over half a second, and [By](#) specifies that the glass shrinks by 10%.

The second event trigger ([MouseLeave](#)) simply stops the first one. When you stop a [Storyboard](#), all the animated properties return to their default values. Therefore, when the user moves the pointer off the button, the button goes back to the way it was before the mouse pointer moved over the button. For more information about animations, see [Animation Overview](#).

- 5. Add an animation for when the button is clicked:** The final step is to add a trigger for when the user clicks the button. Add the following markup anywhere inside of the `ControlTemplate.Triggers` block:

```

<!-- Animation fires when button is clicked, causing glass to spin. -->
<EventTrigger RoutedEvent="Button.Click">
    <EventTrigger.Actions>
        <BeginStoryboard>
            <Storyboard>
                <DoubleAnimation Storyboard.TargetName="glassCube"
                    Storyboard.TargetProperty=
                    "(Rectangle.RenderTransform).(TransformGroup.Children)[1].(RotateTransform.Angle)"
                    By="360" Duration="0:0:0.5" />
            </Storyboard>
        </BeginStoryboard>
    </EventTrigger.Actions>
</EventTrigger>

```

Press F5 to run the application, and click one of the buttons. When you click a button, the glass rectangle spins around.

## Summary

In this walkthrough, you performed the following exercises:

- Targeted a [Style](#) to an object type ([Button](#)).
- Controlled basic properties of the buttons in the entire application using the [Style](#).
- Created resources like gradients to use for property values of the [Style](#) setters.
- Customized the look of buttons in the entire application by applying a template to the buttons.
- Customized behavior for the buttons in response to user actions (such as [MouseEnter](#), [MouseLeave](#), and [Click](#)) that included animation effects.

## See also

- [Create a Button by Using Microsoft Expression Blend](#)
- [Styling and Templating](#)
- [Animation Overview](#)
- [Painting with Solid Colors and Gradients Overview](#)
- [Bitmap Effects Overview](#)