ComponentOne

RangeSlider for WPF

ComponentOne, a division of GrapeCity

201 South Highland Avenue, Third Floor

Pittsburgh, PA 15206 • USA

Internet: info@ComponentOne.com

Web site: http://www.componentone.com

Sales

E-mail: sales@componentone.com

Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

This manual was produced using ComponentOne Doc-To-Help™.

Table of Contents

ComponentOne RangeSlider for WPF Overview

Add smooth numeric data selection to your WPF applications. **ComponentOne RangeSlider™ for WPF** extends the basic slider control and provides two thumb elements instead of one, allowing users to select ranges instead of single values.

For a list of the latest features added to **ComponentOne Studio for WPF**, visit What's New in Studio for WPF.

Getting Started

Get started with the following topics:

Help with ComponentOne Studio for WPF

Getting Started

For information on installing **ComponentOne Studio for WPF**, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with Studio for WPF.

What's New

For a list of the latest features added to **ComponentOne Studio for WPF**, visit What's New in Studio for WPF.

Key Features

ComponentOne RangeSlider for WPF allows you to create customized, rich applications. Make the most of **RangeSlider for WPF** by taking advantage of the following key features:

- **Horizontal or Vertical Orientation**

  Change the orientation with one simple property. Create vertical or horizontal range sliders. See Orientation (page 10) and Changing the Orientation (page 17) for details.

- **Set Min and Max Values**

  Control the minimum and maximum values of the range slider. See Minimum and Maximum (page 9) for details.

- **Customizable Thumbs**

  Customize the thumbs of **C1RangeSlider** to create custom zooming controls. See Thumb Values and Range (page 10) and Setting the Thumb Values for details.

- **Easily Change Colors with ClearStyle**

  RangeSlider supports ComponentOne ClearStyle™ technology which allows you to easily change control brushes without having to override templates. By just setting a few brush properties in Visual Studio you can quickly style each part of the control. See ComponentOne ClearStyle Technology (page 12) and Changing the Background Color (page 16) for details.

RangeSlider for WPF Quick Start

The following quick start guide is intended to get you up and running with **RangeSlider for WPF**. In this quick start you'll start in Visual Studio and create a new project, add the **RangeSlider for WPF** control to your application, and customize the appearance and behavior of the control.

You will create a simple form using a C1RangeSlider and a standard **Rectangle** control. The C1RangeSlider control will control a gradient that is applied to the **Rectangle**, so that at run time moving the slider thumbs will change the gradient and you can explore the possibilities of using **RangeSlider for WPF**.

Step 1 of 4: Setting up the Application

In this step you'll begin in Visual Studio to create a WPF application using **RangeSlider for WPF**. When you add a C1RangeSlider control to your application, you'll have a complete, functional input editor in the form of a slider. You can then further customize the control to your application.

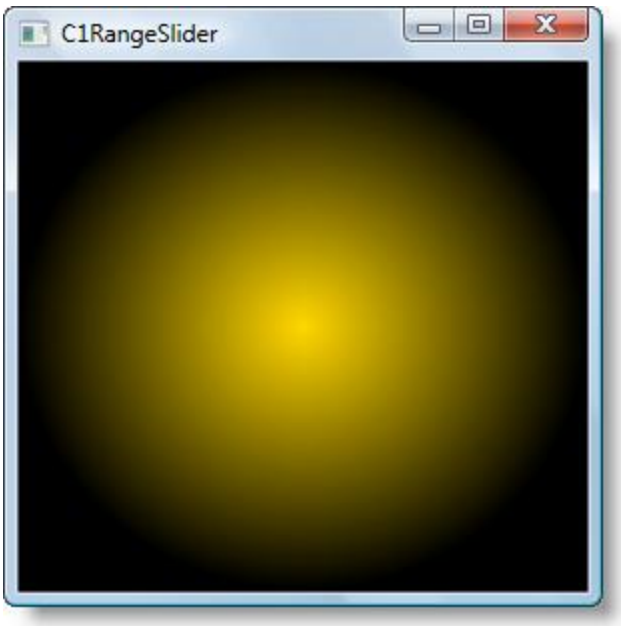
To set up your project and add a C1RangeSlider control to your application, complete the following steps:

1. Create a new WPF project in Visual Studio.
2. Click once within the **Grid** that has been added to the **Window** in your application.
3. Navigate to the Toolbox and double-click the **Rectangle** icon to add the standard control to the **Grid**.
4. In the Design pane, resize **Rectangle1** to fill the entire **Grid**.
5. Switch to XAML view and add a `Fill` to the `<Rectangle>` tag so it appears similar to the following:

```
<Rectangle Name="rectangle1" Stroke="Black">
    <Rectangle.Fill>
        <RadialGradientBrush x:Name="colors">
            <GradientStop x:Name="goldcol" Color="Gold" Offset="0" />
            <GradientStop x:Name="blackcol" Color="Black" Offset="1" />
        </RadialGradientBrush>
    </Rectangle.Fill>
</Rectangle>
```

   This will add a black and gold radial gradient fill to the rectangle.

6. Run your application now and observe that it looks similar to the following:



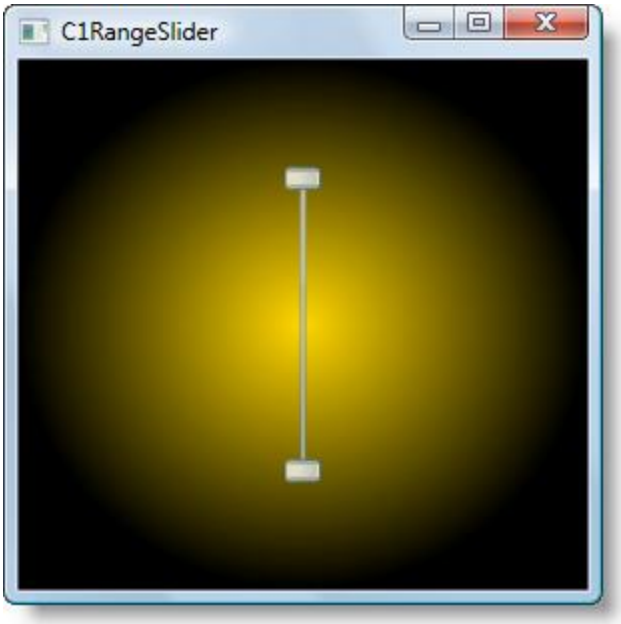
You've successfully created a WPF application and customized the **Rectangle** control. In the next step you'll add and customize the C1RangeSlider control.

Step 2 of 4: Adding a C1RangeSlider Control

In the previous step you created a new WPF project and added a **Rectangle** control with a gradient to the application. In this step you'll continue by adding a C1RangeSlider control that will control the gradient fill in the **Rectangle**.

Complete the following steps:

1. In Design view, click once on the **Rectangle** control.
2. Navigate to the Toolbox and double-click the C1RangeSlider icon to add the control to the application on top of the **Rectangle**.
3. Click once on the C1RangeSlider control and drag it to the center of the Window.
4. Navigate to the Properties window and set the **Margin** property to "50". This will set each edge the same distance away from the window border.
5. In the Properties window set the Orientation property to **Vertical**. By default Orientation is **Horizontal** and the control appears across the window.
6. In the Properties window set the UpperValue property to "1". The upper thumb will now begin at **1**.
7. In the Properties window set the Maximum property to "1". By default this property is set to **100** and the control's range is from **0** to **100**.
8. In the Properties window set the ValueChange property to "0.1". When you click on the slider track at run time, now the slider thumb will move by 0.1.
9. In the Properties window set the **Opacity** property to "0.8". By default this property is set to **1** and the control appears completely opaque. Changing this to a lower number will make the control appear slightly transparent.
10. Run your application now and observe that it looks similar to the following:



You've successfully set up your application's user interface, but right now the slider will do nothing if you move it. In the next step you'll add code to your application.

Step 3 of 4: Adding Code to the Application

In the previous steps you set up the application's user interface and added controls to your application. In this step you'll add code to your application to finalize it.

Complete the following steps:

1. Double-click the **Window1** to switch to Code view and create the **Window1_Loaded** event handler.
2. In Code view, add the following import statement to the top of the page:
   - Visual Basic
     ```
     Imports C1.WPF
     ```
   - C#
     ```
     using C1.WPF;
     ```
3. Add code to the **Window_Loaded** event handler so that it appears like the following:
   - Visual Basic
     ```
     Private Sub Window1_Loaded(ByVal sender As System.Object, ByVal e As
     System.Windows.RoutedEventArgs) Handles MyBase.Loaded
         UpdateGradient()
     End Sub
     ```
   - C#
     ```
     private void Window_Loaded(object sender, RoutedEventArgs e)
     {
         UpdateGradient();
     }
     ```
4. Add the following code just after the **Window1_Loaded** event handler to update the gradient values:
   - Visual Basic
     ```
     Private Sub UpdateGradient()
         If IsLoaded Then
     ```

```
        Me.goldcol.Offset = Me.C1RangeSlider1.LowerValue
        Me.blackcol.Offset = Me.C1RangeSlider1.UpperValue
    End If
End Sub
```

- C#

```
UpdateGradient()
{
    if (IsLoaded)
    {
        this.goldcol.Offset = this.c1RangeSlider1.LowerValue;
        this.blackcol.Offset = this.c1RangeSlider1.UpperValue;
    }
}
```

5. Return to Design view.
6. Click once on the C1RangeSlider control to select it and then navigate to the Properties window.
7. Click the lightning bolt **Events** icon at the top of the Properties window to view events.
8. Double-click the **LowerValueChanged** event to switch to Code view and create the **C1RangeSlider1_LowerValueChanged** event handler. Return to Design view and repeat this step with the **UpperValueChanged** event to create the **C1RangeSlider1_UpperValueChanged** event handler.
9. Add code to the **C1RangeSlider1_LowerValueChanged** event handler so that it appears like the following:

    - Visual Basic

```
Private Sub C1RangeSlider1_LowerValueChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
C1RangeSlider1.LowerValueChanged
    UpdateGradient()
End Sub
```

    - C#

```
private void c1RangeSlider1_LowerValueChanged(object sender, EventArgs
e)
{
    UpdateGradient();
}
```

10. Add code to the **C1RangeSlider1_UpperValueChanged** event handler so that it appears like the following:

    - Visual Basic

```
Private Sub C1RangeSlider1_UpperValueChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
C1RangeSlider1.UpperValueChanged
    UpdateGradient()
End Sub
```

    - C#

```
c1RangeSlider1_UpperValueChanged(object sender, EventArgs e)
{
    UpdateGradient();
}
```

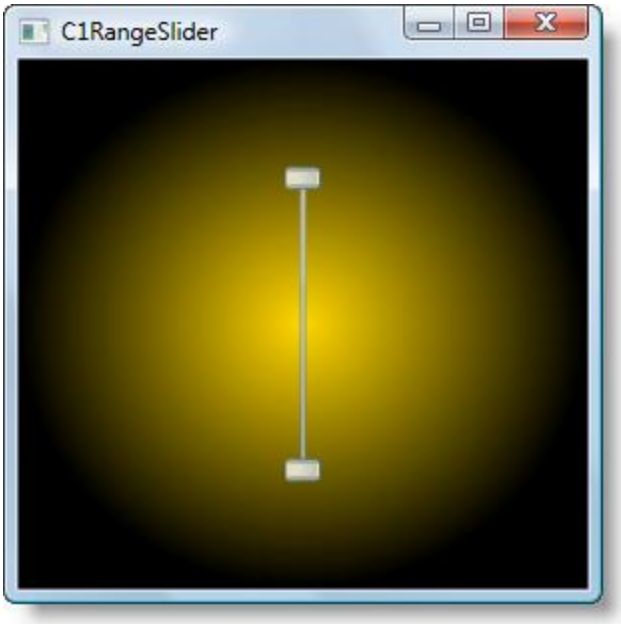
In this step you completed adding code to your application. In the next step you'll run the application and observe run-time interactions.

Step 4 of 4: Running the Application

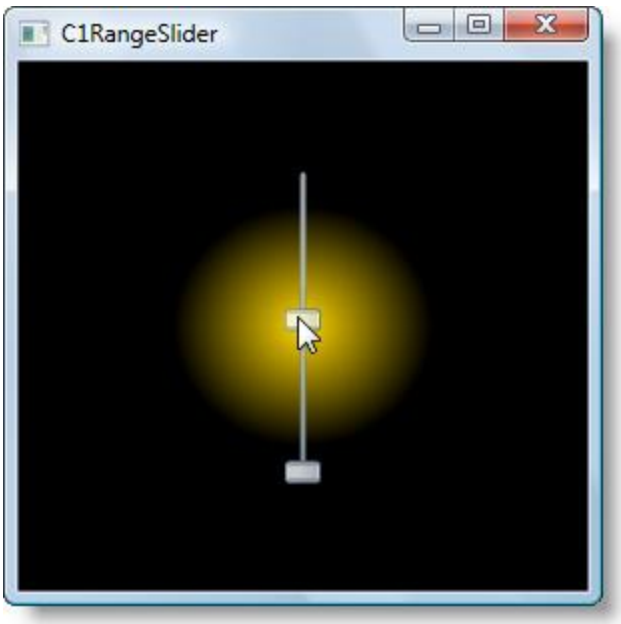
Now that you've created a WPF application and customized the application's appearance and behavior, the only thing left to do is run your application. To run your application and observe **RangeSlider for WPF**'s run-time behavior, complete the following steps:

1. From the **Project** menu, select **Test Solution** to view how your application will appear at run time.

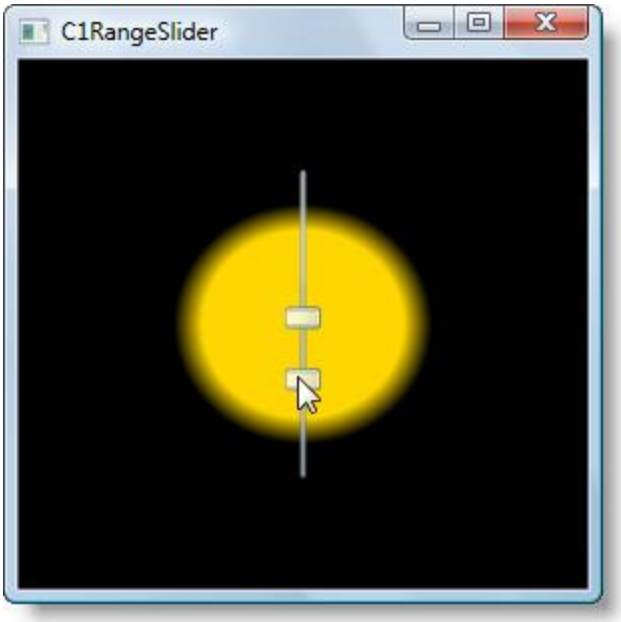
   The application will appear similar to the following:



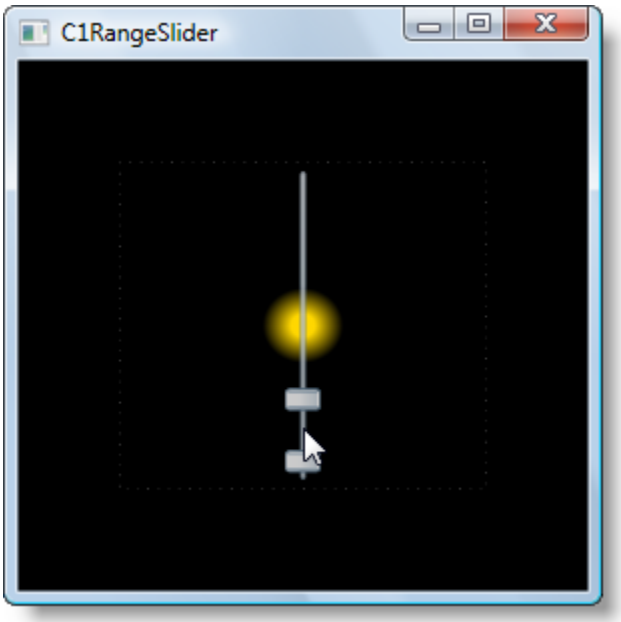
2. Move the top slider thumb down. Notice that the gradient's appearance changes:



3. Move the bottom thumb up, notice that the gradient effect appears less diffused:



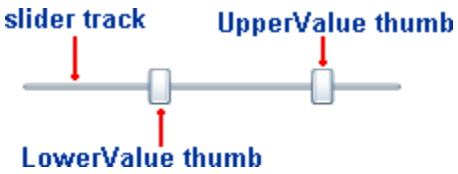
4. Click once between the slider thumbs and drag the cursor down the slider track – notice that both thumbs move together:



Congratulations! You've completed the **RangeSlider for WPF** quick start and created a **RangeSlider for WPF** application, customized the appearance and behavior of the controls, and viewed some of the run-time capabilities of your application.

Working with RangeSlider for WPF

ComponentOne RangeSlider for WPF includes the C1RangeSlider control, a simple input control that moves beyond the typical slider and includes two thumbs for selecting a range of values. When you add the C1RangeSlider control to a XAML window, it exists as a completely functional slider control which you can further customize. The control's interface looks similar to the following image:



Basic Properties

ComponentOne RangeSlider for WPF includes several properties that allow you to set the functionality of the control. Some of the more important properties are listed below. Note that you can see Appearance Properties (page 11) for more information about properties that control appearance.

The following properties let you customize the C1RangeSlider control:

Property	Description
Delay	Gets or sets the time, in milliseconds, the **RepeatButtons** (at the left of the LowerValue thumb and at the right of the UpperValue thumb) wait when they are pressed before they start repeating the click action.
Interval	Gets or sets the time, in milliseconds, between repetitions of the click action, as soon as repeating starts (for the **RepeatButtons** at the left of the LowerValue thumb and at the right of the UpperValue thumb).
LowerValue	Gets or sets the current lower magnitude of the range control.
Maximum	Gets or sets the maximum possible value of the range element.
Minimum	Gets or sets the minimum possible value of the range element.
Orientation	The orientation of the C1RangeSlider (horizontal or vertical).
UpperValue	Gets or sets the current upper magnitude of the range control.
ValueChange	Gets or sets a value to be added to or subtracted from the UpperValue/LowerValue of a **RangeBase** control.

Minimum and Maximum

The Minimum and Maximum properties set the possible range of values allowable in the C1RangeSlider control. The thumb with the smaller number, the LowerValue thumb will not be able to bet set to a value lower than the Minimum and the UpperValue thumb will not be able to bet set to a value lower than the Maximum.

By default, the Minimum property is set to **0** and the Maximum property is set to **100**.

Thumb Values and Range

The C1RangeSlider control includes two thumbs for selecting a range of values. The UpperValue and the LowerValue thumbs move along the slider track. By default, the UpperValue property is set to **100** and the LowerValue property is set to **0**.

The value range is determined by the difference between the UpperValue and the LowerValue:

The ValueChange property determines by what value the UpperValue and the LowerValue thumbs move along the slider track when the track is clicked, note that if the tack is clicked between the UpperValue and LowerValue thumbs (in the range) the thumbs will not move.

The UpperValue property cannot be less than the Minimum property and the LowerValue cannot be less than the Maximum property.

Orientation

C1RangeSlider includes the ability to orient the control either horizontally or vertically using the Orientation property. By default the control initially appears with a horizontal orientation when added to the application. You can easily change the orientation from the Properties window, in XAML, and in code using the Orientation property. For more information, see Changing the Orientation (page 17).

C1RangeSlider includes the following orientations:

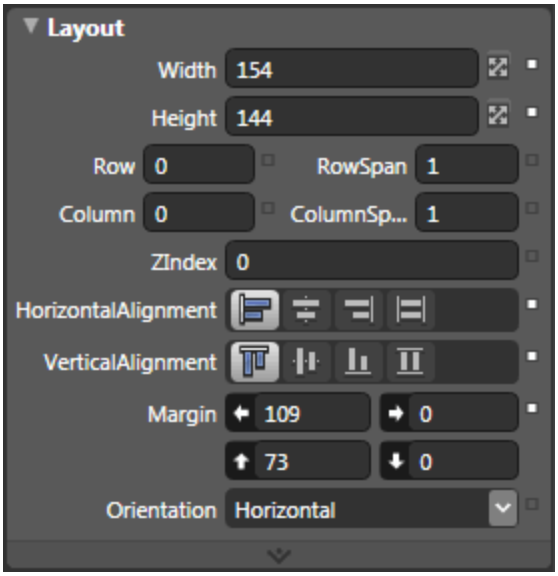
Orientation	Preview
Horizontal (default)	
Vertical	

Layout and Appearance

The following topics detail how to customize the C1RangeSlider control's layout and appearance. You can use built-in layout options to lay your controls out in panels such as Grids or Canvases. Themes allow you to customize the appearance of the grid and take advantage of WPF's XAML-based styling. You can also use templates to format and layout the control and to customize the control's actions.

Layout in a Panel

You can easily lay out the C1RangeSlider and other controls in your WPF application, using the attached layout properties. For example, you can lay out your control in a **Grid** panel with its **Row**, **ColumnSpan**, and **RowSpan** properties and in a **Canvas** panel with its **Left** and **Top** properties. For example, the C1RangeSlider control includes the following **Layout** properties when located within a **Grid** panel:



You can change the sizing, alignment, and location of the C1RangeSlider control within the **Grid** panel.

Appearance Properties

ComponentOne RangeSlider for WPF includes several properties that allow you to customize the appearance of the control. You can change the color, border, and height of the control. The following topics describe some of these appearance properties.

Color Properties

The following properties let you customize the colors used in the control itself:

Property	Description
Background	Gets or sets a brush that describes the background of a control. This is a dependency property.
Foreground	Gets or sets a brush that describes the foreground color. This is a dependency property.

Alignment Properties

The following properties let you customize the control's alignment:

Property	Description

HorizontalAlignment	Gets or sets the horizontal alignment characteristics applied to this element when it is composed within a parent element, such as a panel or items control. This is a dependency property.
VerticalAlignment	Gets or sets the vertical alignment characteristics applied to this element when it is composed within a parent element such as a panel or items control. This is a dependency property.

Border Properties

The following properties let you customize the control's border:

Property	Description
BorderBrush	Gets or sets a brush that describes the border background of a control. This is a dependency property.
BorderThickness	Gets or sets the border thickness of a control. This is a dependency property.

Size Properties

The following properties let you customize the size of the **C1RangeSlider** control:

Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.
MinHeight	Gets or sets the minimum height constraint of the element. This is a dependency property.
MinWidth	Gets or sets the minimum width constraint of the element. This is a dependency property.
Width	Gets or sets the width of the element. This is a dependency property.

ComponentOne ClearStyle Technology

ComponentOne ClearStyle™ technology is a new, quick and easy approach to providing Silverlight and WPF control styling. ClearStyle allows you to create a custom style for a control without having to deal with the hassle of XAML templates and style resources.

Currently, to add a theme to all standard WPF controls, you must create a style resource template. In Microsoft Visual Studio this process can be difficult; this is why Microsoft introduced Expression Blend to make the task a bit easier. Having to jump between two environments can be a bit challenging to developers who are not familiar with Blend or do not have the time to learn it. You could hire a designer, but that can complicate things when your designer and your developers are sharing XAML files.

That's where ClearStyle comes in. With ClearStyle the styling capabilities are brought to you in Visual Studio in the most intuitive manner possible. In most situations you just want to make simple styling changes to the controls

in your application so this process should be simple. For example, if you just want to change the row color of your data grid this should be as simple as setting one property. You shouldn't have to create a full and complicated-looking template just to simply change a few colors.

How ClearStyle Works

Each key piece of the control's style is surfaced as a simple color property. This leads to a unique set of style properties for each control. For example, a **Gauge** has **PointerFill** and **PointerStroke** properties, whereas a **DataGrid** has **SelectedBrush** and **MouseOverBrush** for rows.

Let's say you have a control on your form that does not support ClearStyle. You can take the XAML resource created by ClearStyle and use it to help mold other controls on your form to match (such as grabbing exact colors). Or let's say you'd like to override part of a style set with ClearStyle (such as your own custom scrollbar). This is also possible because ClearStyle can be extended and you can override the style where desired.

ClearStyle is intended to be a solution to quick and easy style modification but you're still free to do it the old fashioned way with ComponentOne's controls to get the exact style needed. ClearStyle does not interfere with those less common situations where a full custom design is required.

ClearStyle Properties

The following table lists all of the ClearStyle-supported properties in the C1RangeSlider control as well as a description of the property:

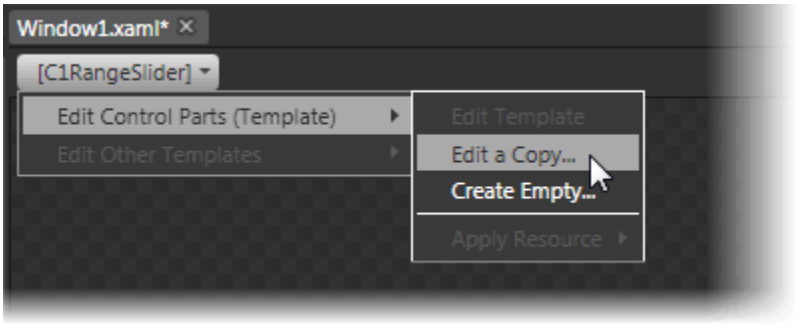
Property	Description
Background	Gets or sets a brush that describes the background of a control. The default **Background** color is LightBlue.
FocusBrush	A brush used to define the appearance of the control, when the control is in focus.
MouseOverBrush	A brush used to define the appearance of the control, when the control is in moused over.
PressedBrush	A brush used to define the appearance of the control, when the control is selected.

Templates

One of the main advantages to using a WPF control is that controls are "lookless" with a fully customizable user interface. Just as you design your own user interface (UI), or look and feel, for WPF applications, you can provide your own UI for data managed by **ComponentOne RangeSlider for WPF**. Extensible Application Markup Language (XAML; pronounced "Zammel"), an XML-based declarative language, offers a simple approach to designing your UI without having to write code.

Accessing Templates

You can access templates in Microsoft Expression Blend by selecting the C1RangeSlider control and, in the menu, selecting **Edit Control Parts (Templates)**. Select **Edit a Copy** to create an editable copy of the current template or **Create Empty**, to create a new blank template.



Note: If you create a new template through the menu, the template will automatically be linked to that template's property. If you manually create a template in XAML you will have to link the appropriate template property to the template you've created.

Note that you can use the Template property to customize the template.

XAML Elements

Several auxiliary XAML elements are installed with **ComponentOne RangeSlider for WPF**. These elements include templates and themes and are located in the **RangeSlider for WPF** installation directory. You can incorporate these elements into your project, for example, to create your own theme based on the default theme.

Included Auxiliary XAML Elements

The following auxiliary XAML element is included with **RangeSlider for WPF**:

Element	Folder	Description
generic.xaml	XAML	Specifies the templates for different styles and the initial style of the control.

RangeSlider for WPF Samples

Please be advised that this ComponentOne software tool is accompanied by various sample projects and/or demos, which may make use of other ComponentOne development tools included with the ComponentOne Studios. Samples can be accessed from the **ComponentOne Studio for WPF ControlExplorer**. To view samples, on your desktop, click the **Start** button and then click **All Programs | ComponentOne | Studio for WPF | Samples | WPF ControlExplorer**.

C# Samples

The following C# sample is included:

Sample	Description
ControlExplorer	The **RangeSlider** page in the **ControlExplorer** sample demonstrates how to customize the C1RangeSlider control.

RangeSlider for WPF Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET and know how to use the C1RangeSlider control in general. If you are unfamiliar with the **ComponentOne RangeSlider for WPF** product, please see the RangeSlider for WPF Quick Start (page 3) first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne RangeSlider for WPF** product.

Each task-based help topic also assumes that you have created a new WPF project.

Setting the Thumb Values

The UpperValue and LowerValue properties get or set the value of the two C1RangeSlider thumbs. By default the C1RangeSlider control starts with the UpperValue property set to "100" and LowerValue property set to "0" set but you can customize this at design time, in XAML, and in code.

At Design Time

To set the UpperValue and LowerValue properties at run time, complete the following steps:

1. Click the C1RangeSlider control once to select it.
2. Navigate to the Properties window, and enter a number, for example "10", in the text box next to the LowerValue property.
3. In the Properties window, enter a number, for example "90", in the text box next to the UpperValue property.

   This will set the UpperValue and LowerValue properties to the values you chose.

In XAML

For example, to set the UpperValue and LowerValue properties add `UpperValue="90" LowerValue="10"` to the `<c1:C1RangeSlider>` tag so that it appears similar to the following:

```
<c1:C1RangeSlider Height="18" HorizontalAlignment="Left"
Margin="10,10,0,0" Name="C1RangeSlider1" VerticalAlignment="Top"
Width="26" UpperValue="90" LowerValue="10" />
```

In Code

For example, to set the UpperValue and LowerValue properties add the following code to your project:

- Visual Basic
```
Me.C1RangeSlider1.LowerValue = 10
Me.C1RangeSlider1.UpperValue = 90
```
- C#
```
this.c1RangeSlider1.LowerValue = 10;
this.c1RangeSlider1.UpperValue = 90;
```

Setting the Value Change

The UpperValue and LowerValue thumbs move along the track when the C1RangeSlider track is clicked. The ValueChange property determines by how much the thumbs move. By default, the ValueChange property is set to "10" and a slider thumb will move by 10 units when the track next to it is clicked. You can customize this value at design time, in XAML, and in code.

At Design Time

To set the ValueChange property at run time, complete the following steps:

1. Click the C1RangeSlider control once to select it.
2. Navigate to the Properties window, and enter a number, for example "5", in the text box next to the ValueChange property.

   This will set the ValueChange property to the value you chose.

In XAML

For example, to set the ValueChange property add `ValueChange="5"` to the `<c1:C1RangeSlider>` tag so that it appears similar to the following:

```
<c1:C1RangeSlider Height="18" HorizontalAlignment="Left"
Margin="10,10,0,0" Name="C1RangeSlider1" VerticalAlignment="Top"
Width="26" ValueChange="5" />
```

In Code

For example, to set the ValueChange property add the following code to your project:

- Visual Basic
```
Me.C1RangeSlider1.ValueChange = 5
```
- C#
```
this.c1RangeSlider1.ValueChange = 5;
```

Run the application and observe:

When you click the track of the C1RangeSlider control, the closest thumb will now move by 5 units.

Changing the Background Color

The **Background** property gets or sets the value of the C1RangeSlider control's background color. By default the C1RangeSlider control starts with the **Background** property unset but you can customize this at design time, in XAML, and in code.

At Design Time

To set the **Background** property at run time, complete the following steps:

1. Click the C1RangeSlider control once to select it.
2. Navigate to the Properties window, and locate the **Background** property.
3. Click the drop-down arrow next to the **Background** property and choose a color, for example **Red**.

This will set the **Background** property to the color you chose

In XAML

For example, to set the **Background** property to **Red** add `Background="Red"` to the `<c1:C1RangeSlider>` tag so that it appears similar to the following:

```
<c1:C1RangeSlider Height="18" HorizontalAlignment="Left"
Margin="10,10,0,0" Name="C1RangeSlider1" VerticalAlignment="Top"
Width="26" Background="Red" />
```

In Code

For example, to set the **Background** property to **Red**, add the following code to your project:

- Visual Basic

```
Me.C1RangeSlider1.Background = System.Windows.Media.Brushes.Red
```

- C#

```
this.c1RangeSlider1.Background = System.Windows.Media.Brushes.Red;
```

Run the application and observe:

The background of the C1RangeSlider control will appear red:

Changing the Orientation

By default the Orientation property is set to **Horizontal** and the slider appears horizontally across the page. If you choose, you can change the Orientation so that content control appears vertically placed instead.

At Design Time

To set the Orientation property to **Vertical** at run time, complete the following steps:

1. Click the C1RangeSlider control once to select it.
2. Navigate to the Properties window, and locate the Orientation property.
3. Click the drop-down arrow next to the Orientation property and choose **Vertical**.

   This will change the Orientation property so that the control appears vertically.

In XAML

To set the Orientation property to **Vertical** add `Orientation="Vertical"` to the `<c1:C1RangeSlider>` tag so that it appears similar to the following:

```
<c1:C1RangeSlider HorizontalAlignment="Left" Margin="10,10,0,0"
Name="C1RangeSlider1" VerticalAlignment="Top" Width="26"
Orientation="Vertical" />
```

In Code

For example, to set the Orientation property to **Vertical**, add the following code to your project:

- Visual Basic

```
Me.C1RangeSlider1.Orientation = Orientation.Vertical
```

- C#

```
this.c1RangeSlider1.Orientation = Orientation.Vertical;
```

Run the application and observe:

The background of the C1RangeSlider control will appear vertical: