ComponentOne

Gauges for UWP

ComponentOne, a division of GrapeCity
201 South Highland Avenue, Third Floor
Pittsburgh, PA 15206 USA

Website: http://www.componentone.com
Sales: sales@componentone.com
Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

Trademarks

Warranty

ComponentOne warrants that the media on which the software is delivered is free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective media to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for the defective media by sending it and a check for $2 5 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original media on which the software is delivered is set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. ComponentOne is not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

Table of Contents

Gauges for UWP 3
Getting Started 4
Help with UWP Edition 4
Gauges for UWP Key Features 5
Gauges for UWP Quick Start 6
Step 1 of 4: Setting up the Application 6
Step 2 of 4: Adding Controls 6-9
Step 3 of 4: Adding Code to the Application 9
Step 4 of 4: Running the Application 9-11
Why Use Gauge Controls? 12
Using C1Radial Gauge 13
C1RadialGauge Values 13-14
C1RadialGauge Start Angle and Sweep Angle 14-15
C1RadialGauge Decorators 15-16
C1RadialGauge Decorators Location 16-17
C1RadialGauge Decorators Value Binding 17
C1RadialGauge Pointer and Pointer Cap 18
C1RadialGauge Face and Cover 18-19
Using C1Linear Gauge 20
C1LinearGauge Values 20-21
C1LinearGauge Orientation 21
C1LinearGauge Decorators 21-22
C1LinearGauge Decorators Location 22-23
C1LinearGauge Pointer 23
C1LinearGauge Face and Cover 23-24
Using C1Knob 25
C1Knob Values 25-26
C1Knob Start Angle and Sweep Angle 26
C1Knob Interaction 26
C1Knob Decorators 26-27
C1KnobDecorators Location 27-28
Gauges for UWP Task-Based Help 29
Setting the Start Value 29-30
Setting the Minimum and Maximum Values 30-31

Adding Labels to the Gauge 31-32
Adding Tick Marks to the Gauge 32-34
Customizing Tick Marks 34-35
Customizing the Gauge Shape 35-37
Customizing the Pointer's Appearance 37-38

Gauges for UWP

Add some flare to your dashboards with modern looking and interactive gauges. **Gauges for UWP** includes several gauge controls to enhance your data visualizations and business dashboards. They provide you an attractive way to display your data on a Windows tablet.

Getting Started

Help with UWP Edition

Getting Started

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

Gauges for UWP Key Features

Gauges for UWP includes the following key features:

- **Seven Gauge Controls**

  Gauges for UWP includes 7 controls with different shapes. Select the most appropriate gauge for your data:

  - C1RadialGauge
  - C1LinearGauge
  - C1Knob
  - C1RegionKnob
  - C1RulerGauge
  - C1SpeedometerGauge
  - C1VolumeGauge
  - Interactive Gauges

  The **C1Knob** control enables the end-user to drag the pointer to a value. Also included is the unique **C1RegionKnob** with customizable regions. Interactive gauges offer an alternative to text-based editors or sliders and a very engaging experience for your users.

- **Tick Marks and Labels**

  Define marks and labels in XAML or code. Use simple properties to customize their interval, location, and appearance. Apply formatting to the gauge labels; for example, format labels in currency or percentage format using standard format strings.

- **Ranges**

  Add colored ranges to the gauge to draw attention to a certain range of values. Use simple properties to customize their start and end points, as well as location, size, and appearance. Create non-linear ranges by specifying a start and end width to show growth and add visual appeal to any gauge.

- **Pointer Customization**

  Use simple properties to customize the appearance and location of the pointer and pointer cap.

- **Scale Customization**

  Use simple properties to set the start and sweep angle of the gauge scale. **Gauges for UWP** also supports logarithmic scales.

- **Off Mode Support**

  If there is no value, you can set the off position outside the range.

- **Easily Change Colors with ClearStyle**

  Gauges for UWP supports ClearStyle technology that allows you to easily change control colors without having to change control templates. By just setting a few color properties in Visual Studio you can easily change the look of any gauge.

Gauges for UWP Quick Start

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

You will create an application that includes the C1RadialGauge, C1LinearGauge, and C1Knob controls. At run time when a user changes the value of a slider, the value of the gauge controls will also change.

Step 1 of 4: Setting up the Application

In this step you will create an application in Visual Studio, and add **StackPanel** panels to customize the layout of the controls.

To set up your project, follow these steps:

1. In Visual Studio, select **File | New | Project**.
2. Select Templates | Visual C# | Windows | Universal. From the templates list, select Blank App (Universal Windows). Enter a **Name** for your project and click **OK** to create your project.

   The MainPage.xaml page will open with the cursor between the <Grid> and </Grid> tags.
3. Navigate to the Toolbox and double-click the **StackPanel** icon to add the panel to **MainPage.xaml**.
4. Add x:Name="sp1" Width="Auto" Height="Auto" Orientation="Vertical" HorizontalAlignment="Center" VerticalAlignment="Center" to the <StackPanel> tag so that it appears similar to the following:

   Markup

   ```
   <StackPanel x:Name="sp1" Width="Auto" Height="Auto" Orientation="Vertical"
   HorizontalAlignment="Center" VerticalAlignment="Center"></StackPanel>
   ```

   Elements in the panel will now appear centered and vertically positioned.
5. In the XAML window of the project, place the cursor between the <StackPanel> and </StackPanel> tags.
6. Navigate to the Toolbox and double-click the **StackPanel** icon to add the panel to the existing **StackPanel**.
7. Add x:Name="sp2" Width="Auto" Height="Auto" Orientation="Horizontal" HorizontalAlignment="Center" VerticalAlignment="Center" to the <StackPanel> tag so that it appears similar to the following:

   Markup

   ```
   <StackPanel x:Name="sp2" Width="Auto" Height="Auto" Orientation="Horizontal"
   HorizontalAlignment="Center" VerticalAlignment="Center"></StackPanel>
   ```

   Elements in the panel will now appear centered and horizontally positioned.

You've successfully created a new UWP style project and set up your application. In the next step you'll added **Gauges** for UWP controls to the application and customize those controls.

Step 2 of 4: Adding Controls

In this step you'll set up the application by adding C1RadialGauge, C1LinearGauge, and C1Knob controls to the project.

To set add the gauge controls to your application, following these steps:

1. In the XAML window of the project, place the cursor between the <StackPanel x:Name="sp2"> and </StackPanel> tags.

2. Navigate to the Toolbox and double-click the **C1Knob** icon to add the control to the **StackPanel**. This will add

the reference and XAML namespace automatically.

3. Give your control a name by adding x:Name="c1kb1" to the <Gauge:C1Knob> tag so that it appears similar to the following:

```
Markup
<Gauge:C1Knob x:Name="c1kb1">
```

By giving it a unique identifier, you'll be able to access the control in code.

4. Resize your control a margin by adding Width="150" to the <Gauge:C1Knob> tag so that it appears similar to the following:

```
Markup
<Gauge:C1Knob x:Name="c1kb1" Width="150">
```

The control will appear smaller when the application is run.

5. Give your control a margin by adding Margin="5" to the <Gauge:C1Knob> tag so that it appears similar to the following:

```
Markup
<Gauge:C1Knob x:Name="c1kb1" Width="150" Margin="5">
```

This will add spacing between the **C1Knob** and other controls you will add to the page.

6. Set the minimum and maximum values by adding Minimum="0" Maximum="100" to the <Gauge:C1Knob> tag so that it appears similar to the following:

```
Markup
<Gauge:C1Knob x:Name="c1kb1" Width="150" Margin="5" Minimum="0" Maximum="100">
```

This determines the highest and lowest values available in the knob.

7. In the XAML window of the project, place the cursor between the </Gauge:C1Knob> and </StackPanel> tags.
8. Navigate to the Toolbox and double-click the **C1RadialGauge** icon to add the control to the **StackPanel**.
9. Customize the control by adding x:Name="c1rg1" Margin="5" Minimum="0" Maximum="100" Height="300" to the <Gauge:C1RadialGauge> tag so that it appears similar to the following:

```
Markup
<Gauge:C1RadialGauge x:Name="c1rg1" Margin="5" Minimum="0" Maximum="100"
Height="300" Value="100" StartAngle="0" SweepAngle="300"></Gauge:C1RadialGauge>
```

This will give the **C1RadialGauge** a name, resize the control, and set the minimum and maximum values of the control.

10. Add the following markup between the x <Gauge:C1RadialGauge> and </Gauge:C1RadialGauge> tags to change the appearance of the gauge:

```
Markup
<Gauge:C1GaugeRange To="40" Location="0.8" Fill="#088080" Width="0.2"
```

```
Opacity="0.6" />
<Gauge:C1GaugeRange From="75" Fill="#088080" Location="0.9" EndWidth="0.2"
Opacity="0.3" />
<Gauge:C1GaugeMark Interval="20" />
<Gauge:C1GaugeMark Interval="10" />
<Gauge:C1GaugeMark Interval="1" />
<Gauge:C1GaugeLabel Interval="20" Alignment="In" AlignmentOffset="10"
FontSize="16" />
```

This will set the appearance of the gauge range and tick marks.

11. In the XAML window of the project, place the cursor between the first and second </StackPanel> tags.
12. Navigate to the Toolbox and double-click the **Slider** icon to add the standard control to the **StackPanel**.
13. Customize the control by adding x:Name="s1" Height="400" Minimum="0" Maximum="100" ValueChanged="s1_ValueChanged_1" Orientation="Vertical" to the <Slider> tag so that it appears similar to the following:

Markup

```
<Slider x:Name="s1" Height="400" Minimum="0" Maximum="100"
ValueChanged="s1_ValueChanged_1" Orientation="Vertical"/>
```

This will give the **Slider** a name, resize the control, and set the minimum and maximum values. You will add code for the event handler in a later step.

14. In the XAML window of the project, place the cursor between the </Slider> and </StackPanel> tags.
15. Navigate to the Toolbox and double-click the **C1LinearGauge** icon to add the control to the **StackPanel**.
16. Customize the control by adding x:Name="c1lg1" Minimum="0" Maximum="100" Width="120" Height="500" to the <Gauge:C1LinearGauge> tag so that it appears similar to the following:

Markup

```
<Gauge:C1LinearGauge x:Name="c1lg1" Minimum="0" Maximum="100"  Width="120"
Height="500" Orientation="Vertical" XAxisLocation="0.05" XAxisLength="0.9"
YAxisLocation="0.2"></Gauge:C1LinearGauge>
```

This will give the **C1LinearGauge** control a name, resize the control, and set the minimum and maximum values.

17. Add the following markup between the x <Gauge:C1LinearGauge> and </Gauge:C1LinearGauge> tags to change the appearance of the gauge:

Markup

```
<Gauge:C1GaugeMark Interval="20" />
<Gauge:C1GaugeMark Interval="10" />
<Gauge:C1GaugeMark Interval="2" />
<Gauge:C1GaugeLabel Interval="20" Format="n0" Alignment="Out"
AlignmentOffset="30" FontSize="16"/>
<Gauge:C1GaugeRange To="40" Location="0" Fill="#088080" Width="0.2"
Opacity="0.2"/>
<Gauge:C1GaugeRange From="40" To="80" Location="0" Fill="#088080" Width="0.2"
```

```
Opacity="0.4"/>
<Gauge:C1GaugeRange From="80" To="100" Location="0" Fill="#088080" Width="0.2"
Opacity="0.6"/>
```

This will set the appearance of the gauge range and tick marks.

You've successfully set up your application's user interface. You've added **Gauges for UWP** controls to the application and customized those controls. In the next step you'll add code to your application.

Step 3 of 4: Adding Code to the Application

In the previous step you created a new UWP style project and added **Gauges for UWP** controls to the application. In this step you'll add code to your application to customize it.
Complete the following steps:

1. Select **View | Code** to switch to Code view.
2. Add the following imports statements to the top of the page:

Visual Basic

```
Imports C1.Xaml
Imports C1.Xaml.Gauge
```

C#

```
using C1.Xaml;
using C1.Xaml.Gauge;
```

3. Add the code for the s1_ValueChanged_1 event handler to set the gauge and slider control values. It will look like the following:

Visual Basic

```
Private Sub s1_ValueChanged_1(sender As Object, e As
RangeBaseValueChangedEventArgs)
    Me.c1lg1.Value = Me.s1.Value
    Me.c1rg1.Value = Me.s1.Value
    Me.c1kb1.Value = Me.s1.Value
End Sub
```

C#

```
private void s1_ValueChanged_1(object sender, RangeBaseValueChangedEventArgs e)
{
    this.c1lg1.Value = this.s1.Value;
    this.c1rg1.Value = this.s1.Value;
    this.c1kb1.Value = this.s1.Value;
}
```

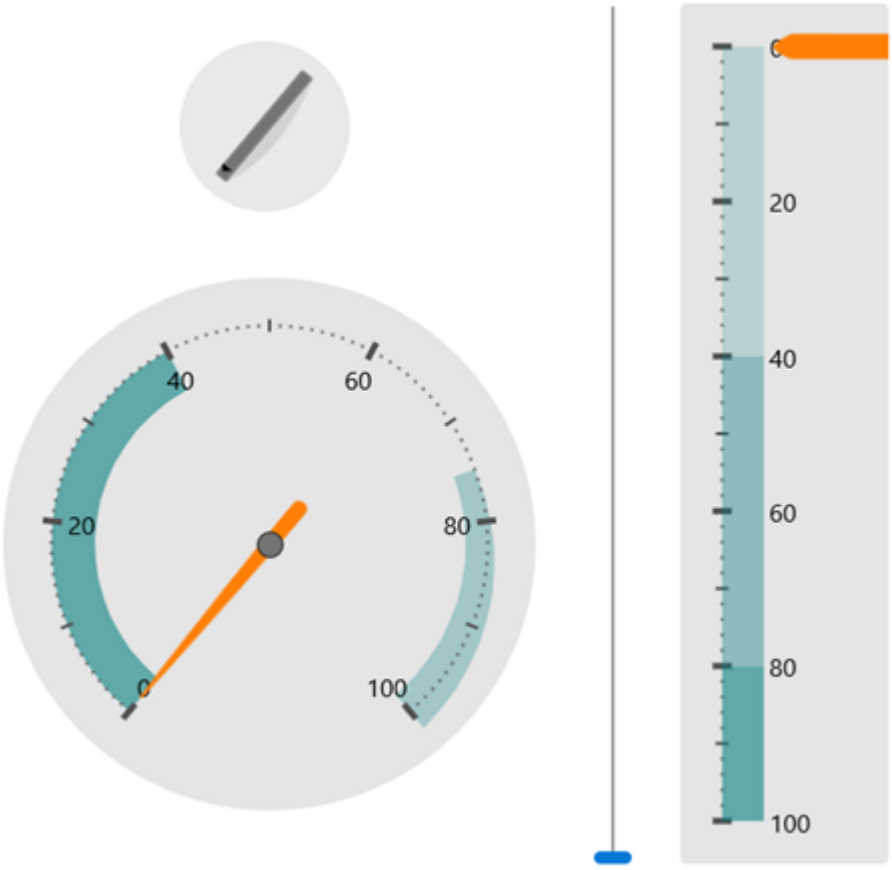
In this step you completed adding code to your application. In the next step you'll run the application and observe run-time interactions.

Step 4 of 4: Running the Application

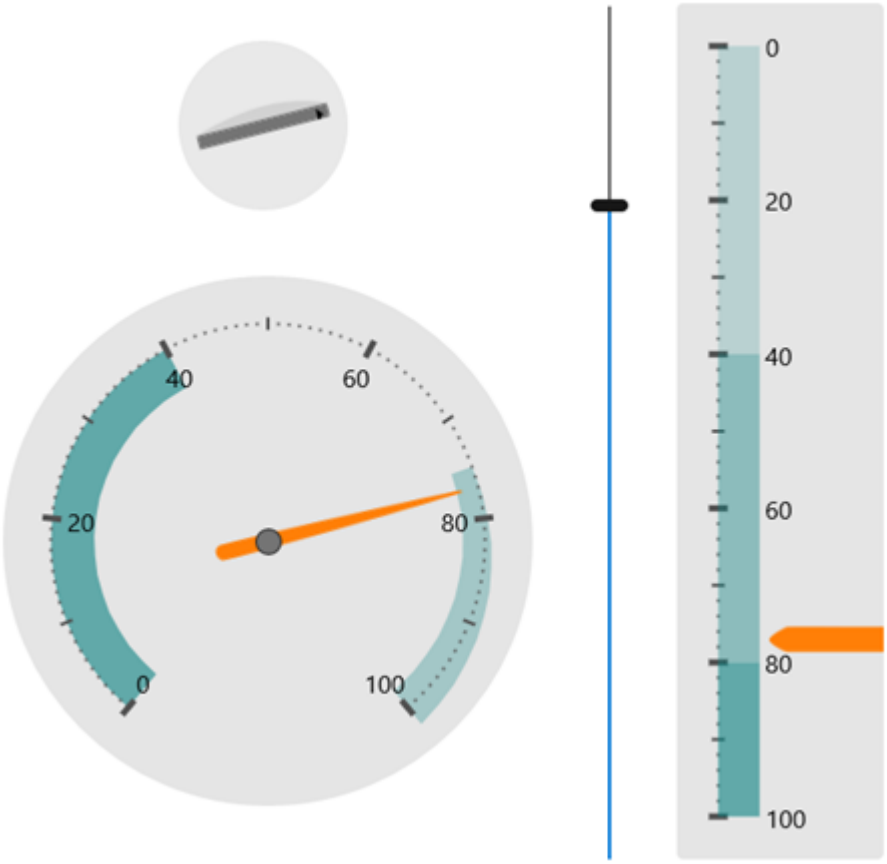
Now that you've created a UWP style application and customized the application's appearance and behavior, the only thing left to do is run your application. To run your application and observe **Gauges for UWP**'s run-time behavior, complete the following steps:

1. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time.

   The application will appear similar to the following:



2. Click and drag the slider's thumb button. Notice that the values of the C1Knob, C1RadialGauge, and C1LinearGauge controls change:



Congratulations!

You've completed the **Gauges for UWP** quick start and created an application using the C1RadialGauge, C1LinearGauge, and C1Knob controls and viewed some of the run-time capabilities of your application.

Why Use Gauge Controls?

You might be asking why you'd need to use gauge controls – after all, gauges just display a single value and you could display that value using a simple label instead of a gauge.

Gauges are better because they also display a range, allowing users to determine instantly whether the current value is low, high, or intermediate. You could use two additional labels to display the range as well as the current value, but that would make your user interface more confusing. That is why many applications use progress indicators that are simple linear gauges, instead of showing progress simply as a label.

Gauges are also more visually attractive than simple labels (or sliders or scrollbars), and that adds value to your applications.

But why use a gauge control instead of simply asking a designer to create a visually attractive gauge in XAML and then animating an element to show the current value? Why use a control?

There are a couple of reasons for that. First, you may not be a great designer and may not have access to one. Second, you probably don't need a single gauge in your application. You may need several, showing values that span different ranges. Maybe you don't even know the actual range when you are writing the application (what's the maximum value of sales this quarter?).

Gauge controls provide the flexibility to adjust the ranges programmatically, based on data, rather than hardwiring them in XAML.

Using C1Radial Gauge

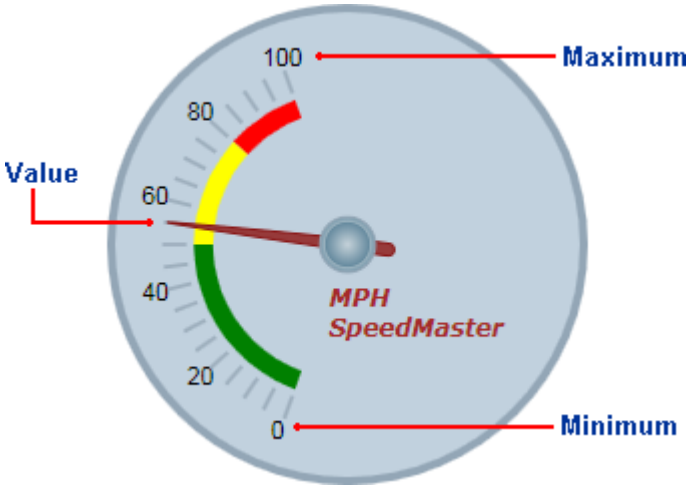
C1RadialGauge uses a rotating pointer to show a value along a curved scale. The C1RadialGauge control displays a value using a rotating pointer. The value is represented by a Value property and the range is defined by the Minimum and Maximum properties. The C1RadialGauge control appears similar to a typical speedometer:

Creating and using a **C1RadialGauge** typically involves the following steps:

1. Creating the **C1RadialGauge** control and setting its main properties: **C1Gauge.Minimum**, **C1Gauge.Maximum**, C1RadialGauge.StartAngle, and C1RadialGauge.SweepAngle.
2. Adding C1GaugeMark and C1GaugeLabel decorators to show the scale. Each element may show a set of labels, tick marks, or both.
3. Optionally adding C1GaugeRange decorators to highlight parts of the scale. Ranges are typically used to indicate ranges that are too low, acceptable, or too high. Ranges can also be dynamic, moving automatically when the C1Gauge.Value property changes.
4. Optionally customizing gauge elements with XAML templates.
5. Setting the **C1Gauge.Value** property to display the value you want to show.

C1RadialGauge Values

You can use the C1RadialGauge control's C1Gauge.Minimum, C1Gauge.Maximum, and C1Gauge.Value properties to specify the available range and the selected value in that range:



The **C1Gauge.Minimum** and **C1Gauge.Maximum** properties specify the range of values the gauge is designed to show. For example, a thermometer may have a scale ranging from -40 to 100 degrees, and a speedometer may have range of 0 to 140 miles per hour. The range is specified through the **C1Gauge.Minimum** and **C1Gauge.Maximum** properties (of type **double**). The default range for a **C1RadialGauge** control is from 0 to 100.

The **C1Gauge.Value** property indicates the current value of the gauge. In the **C1RadialGauge** control, this is indicated visually by the value the **C1GaugePointer** element is pointing to. The default **C1Gauge.Value** for a **C1RadialGauge** control is 50.

C1RadialGauge Start Angle and Sweep Angle

Once the range has been defined, you need to specify the angles that match the C1Gauge.Minimum and C1Gauge.Maximum values. The C1RadialGauge.StartAngle defines the position of the pointer when the C1Gauge.Value property is set to the **C1Gauge.Minimum** value in the range. The C1RadialGauge.SweepAngle specifies the rotation added to the **C1RadialGauge.StartAngle** when the **C1Gauge.Value** property is set to the **C1Gauge.Maximum** value in the range.

All angles are specified in degrees, measured clockwise from the top of the control. The angles may be negative, but the absolute value of the **C1RadialGauge.SweepAngle** may not exceed 360 degrees. The default values for **C1RadialGauge.StartAngle** is -140 and for **C1RadialGauge.SweepAngle** is 280.

The images below show the effect of the **C1RadialGauge.StartAngle** and **C1RadialGauge.SweepAngle** properties:

StartAngle = 0
SweepAngle = 90

StartAngle = 0
SweepAngle = -90

StartAngle = 45
SweepAngle = 270

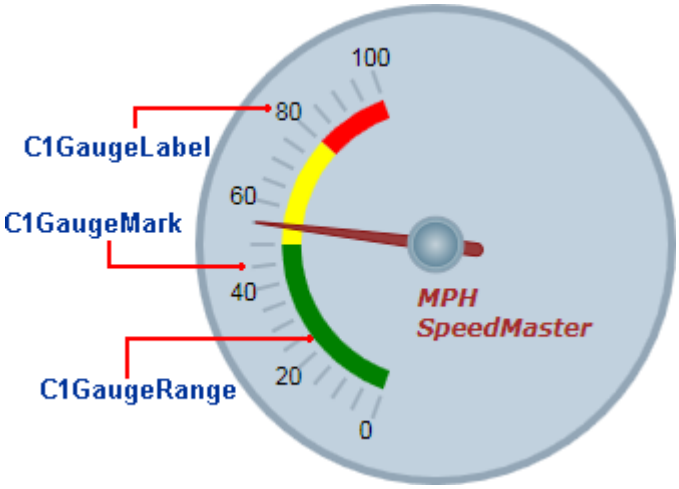
StartAngle = -160
SweepAngle = 180

StartAngle = -160
SweepAngle = -180

StartAngle = -140
SweepAngle = 280

C1RadialGauge Decorators

By default, the C1RadialGauge control displays only a blue-gray background and a pointer. In most applications, you'll also want to display a scale composed of labels and tick marks that allow users to see what the current value is and where it lies within the gauge's range. This is done by adding C1GaugeMark, C1GaugeLabel, and C1GaugeRange elements to the gauge's C1Gauge.Decorators collection:



The decorators are displayed at specific positions on the scale, determined by the value of the C1GaugeDecorator.From, C1GaugeDecorator.To, and C1GaugeMark.Interval properties.

In the image above, you'll see one **C1GaugeMark** and one **C1GaugeLabel** element:

Markup

```
<!-- Add label marks -->
<Gauge:C1GaugeLabel From="0" To="100" Interval="20" Location="1.1"/>

<!-- Add tick marks -->
```

```
<Gauge:C1GaugeMark From="0" To="100" Interval="5" Location=".9"/>
```

The **C1GaugeLabel** element shows labels for the values 0 to 100 along the scale. The **C1GaugeMark** element shows tick marks spaced 5 units apart.

In addition to showing the scale, you may want to highlight parts of the scale range. For example, you may want to add a red marker to indicate that values in that range are too low (sales) or too high (expenses). This can be done easily by adding one or more **C1GaugeRange** elements to the gauge's **C1Gauge.Decorators** collection.

In the image above, you'll see three **C1GaugeRange** elements:

Markup

```
<!-- Add three colored ranges -->
<Gauge:C1GaugeRange From="80" To="100" Location="0.7" Fill="Red" />
<Gauge:C1GaugeRange From="50" To="80" Location="0.7" Fill="Yellow" />
<Gauge:C1GaugeRange From="0" To="50" Location="0.7" Fill="Green" />
```

The **C1GaugeRange** elements show red, yellow, and green range areas. Each **C1GaugeRange** element displays a curved swath along the scale. The color of the swath is determined by the C1GaugeRange.Fill property, and the position is determined by the **C1GaugeDecorator.From** and **C1GaugeDecorator.To** properties. You can control the thickness of the ranges using the C1GaugeRange.StrokeThickness property.

C1RadialGauge Decorators Location

Each decorator element has a C1GaugeDecorator.Location property that determines where the elements are displayed. These properties range from zero (the center of the gauge) to one (the outer edge of the gauge). The gauge control also has a C1RadialGauge.Radius property that ranges from zero to one and affects the positioning of all decorators. The default value for the radius property is 0.8, which causes all decorators to be displayed entirely within the control.

In the C1RadialGauge Decorators example, the **C1GaugeDecorator.Location** property for the C1GaugeLabel was set to 1.1. This caused the labels to appear offset towards the outer edge of the gauge. The labels are still drawn within the control because the **C1RadialGauge.Radius** property is set to 0.8 (the actual position of the labels in this case can be calculated as 1.1 * 0.8 = 0.88).

The diagrams below show the effect of the **C1GaugeDecorator.Location** properties applied to the C1GaugeMark and **C1GaugeLabel** elements on our sample gauge:


C1GaugeLabel.Location = 1.1
C1GaugeMark.Location = 1


C1GaugeLabel.Location = 1
C1GaugeMark.Location = 1


C1GaugeLabel.Location = 0.6
C1GaugeMark.Location = 1


C1GaugeLabel.Location = 1.12
C1GaugeMark.Location = 1.05


C1GaugeLabel.Location = 1
C1GaugeMark.Location = 1.3


C1GaugeLabel.Location = 1.1
C1GaugeMark.Location = 1.4

Notice how specifying values greater than 1 for the **C1GaugeDecorator.Location** may cause the labels or marks to be drawn outside the body of the gauge.

You may specify as many **C1GaugeMark** elements as you want. For example, you could create a clock gauge with a range from 0 to 60 minutes. In that case, you could use one **C1GaugeLabel** and two **C1GaugeMark** elements:

- One **C1GaugeLabel** with an interval of 15, to show labels for the four "main" hours (12, 3, 6, 9);
- One **C1GaugeMark** with an interval of five, to show every full hour;
- One **C1GaugeMark** with an interval of one, to show each minute.

You may also customize the labels and tick marks using the Template properties.

C1RadialGauge Decorators Value Binding

The ranges are not restricted to static values. You can use the C1GaugeRange.ValueBinding property to bind the range's starting or ending positions to the current value being displayed by the gauge. For example, the code below would cause the red range to appear only when the speed exceeded 80 miles per hour:

Markup

```
<!-- Add three colored ranges -->
<Gauge:C1GaugeRange From="80" ValueBinding="To" Location="0.7" Background="Red" />
<Gauge:C1GaugeRange From="50" To="80" Location="0.7" Fill="Yellow" />
<Gauge:C1GaugeRange From="0" To="50" Location="0.7" Fill="Green" />
```

The diagrams below show the effect of this change as the **Value** property changes:

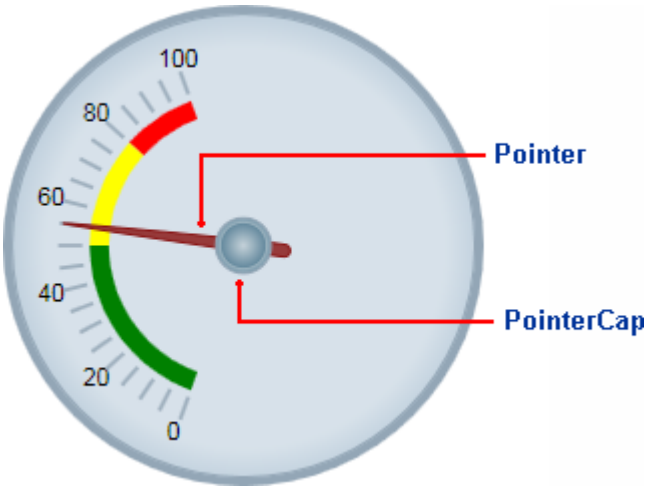

Value = 55


Value = 90


Value = 100

C1RadialGauge Pointer and Pointer Cap

The C1RadialGauge control includes a pointer which indicates the selected C1Gauge.Value of the control. The pointer consists of the actual **C1GaugePointer** element and the C1RadialGauge.PointerCap element:



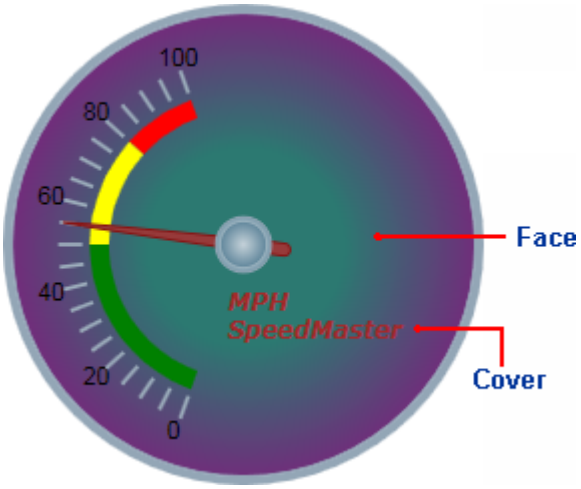
The C1RadialGauge.PointerOrigin property sets the location of the pointer; you can customize the location of the pointer by setting this property – for example the point (0, 0) is the top-left corner of the control and the point (0.5, 0.5) is the center of the control. The C1RadialGaugePointer.Location property sets the relative location of the **C1GaugePointer** element.

The **C1GaugePointer** element appears by default as a brown tapering element, but you can customize the appearance of the **C1GaugePointer** element by setting several properties, including the C1Gauge.PointerFill,C1Gauge.PointerLength , C1Gauge.PointerOffset, , C1Gauge.PointerStrokeThickness, C1Gauge.PointerStyle, C1Gauge.PointerVisibility, and C1Gauge.PointerWidth properties.

The **C1RadialGauge.PointerCap** element appears by default as a gray circle, but you can customize the appearance of the **C1RadialGauge.PointerCap** element by setting several properties, including the **C1RadialGauge.PointerCap**, C1RadialGauge.PointerCapFill, C1RadialGauge.PointerCapStroke, C1RadialGauge.PointerCapStrokeThickness, and C1RadialGauge.PointerCapStyle properties. You can also edit the C1RadialGauge.PointerCapSize template.

C1RadialGauge Face and Cover

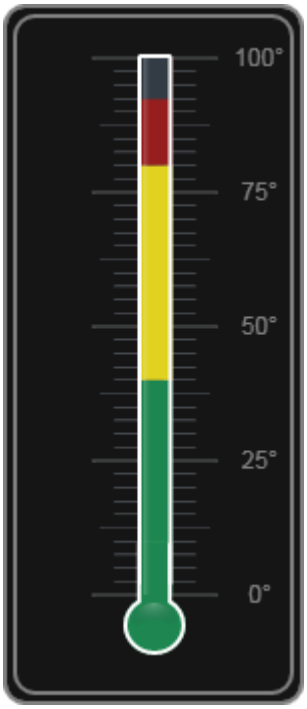
Like a watch, the C1RadialGauge control includes a C1Gauge.Face and a C1Gauge.Cover. The **C1Gauge.Face** appears above the background but behind the pointer and other decorators, and the **C1Gauge.Cover** appears, like a glass over on a watch, above all other elements. For example, in the image below the **C1Gauge.Face** includes a gradient that appears behind the elements in the gauge and the **C1Gauge.Cover** includes text that appears above the **C1Gauge.Face** and all other elements:



You can customize the appearance of the **C1Gauge.Cover** by using the C1Gauge.CoverTemplate and you can customize the appearance of the **C1Gauge.Face** by using the C1Gauge.FaceTemplate.

Using C1Linear Gauge

The C1LinearGauge control has an object model that is almost identical to the one of the **C1RadialGauge**. C1LinearGauge uses a linear pointer to show a value along a linear scale. This is similar to a typical thermometer:

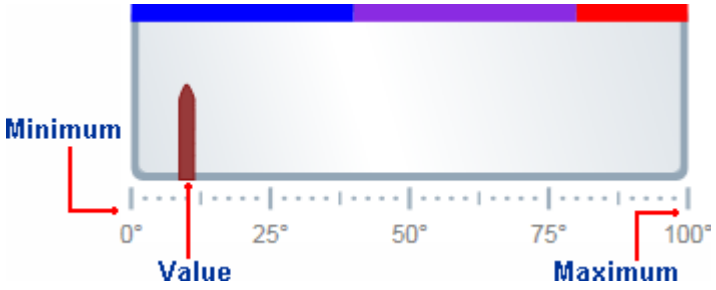


The steps involved in creating and using a **C1LinearGauge** control are the same as the ones we described before for the C1RadialGauge:

1. Create the **C1LinearGauge** control and set its main properties: C1Gauge.Minimum, C1Gauge.Maximum, and C1LinearGauge.Orientation.
2. Add **C1GaugeMark** decorators to show the scale. Each **C1GaugeMark** element may show a set of labels, tick marks, or both.
3. Optionally add C1GaugeRange decorators highlight parts of the scale. Ranges are typically used to indicate ranges that are too low, acceptable, or too high. Ranges can also be dynamic, moving automatically when the C1Gauge.Value property changes.
4. Optionally customize gauge elements with XAML templates.
5. Set the C1Gauge.Value property to display the value you want to show.

C1LinearGauge Values

You can use the C1LinearGauge control's C1Gauge.Minimum, C1Gauge.Maximum, and C1Gauge.Value properties to specify the available range and the selected value in that range:



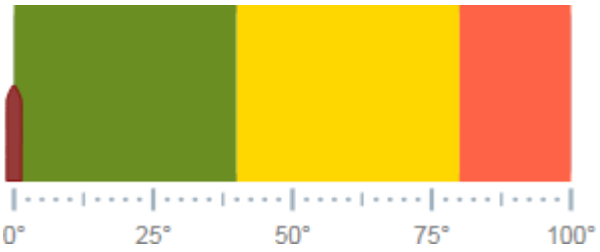
The **C1Gauge.Minimum** and **C1Gauge.Maximum** properties specify the range of values the gauge is designed to show. For example, a thermometer may have a scale ranging from -40 to 100 degrees, and a speedometer may have range of 0 to 140 miles per hour. The range is specified through the **C1Gauge.Minimum** and **C1Gauge.Maximum** properties (of type **double**). The default range for a **C1LinearGauge** control is from 0 to 100.

The **C1Gauge.Value** property indicates the current value of the gauge. In the **C1LinearGauge** control, this is indicated visually by the value the **C1GaugePointer** element is pointing to. The default **C1Gauge.Value** for a **C1LinearGauge** control is 0; in the above image, the **C1Gauge.Value** was set to 10.

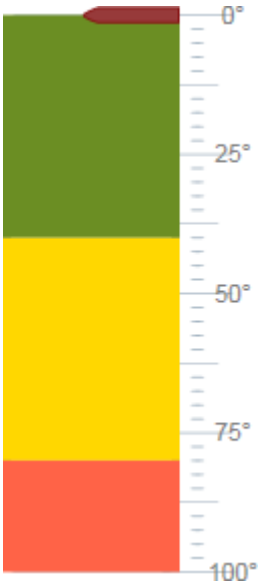
C1LinearGauge Orientation

C1LinearGauge controls do not have the C1RadialGauge.StartAngle and C1RadialGauge.SweepAngle properties used with radial gauges. Instead, they have an C1LinearGauge.Orientation property that you can use to create vertical or horizontal gauges.

By default, the **C1LinearGauge.Orientation** property is set to **Horizontal** and the gauge appears displayed horizontally in the application:



You can set the **C1LinearGauge.Orientation** property to **Vertical** to create a vertical gauge:



By default, the vertical **C1LinearGauge** control will display a scale from top to bottom – for example 0 to 100 in the example above. To reverse the direction of the scale you would need to set the following properties:

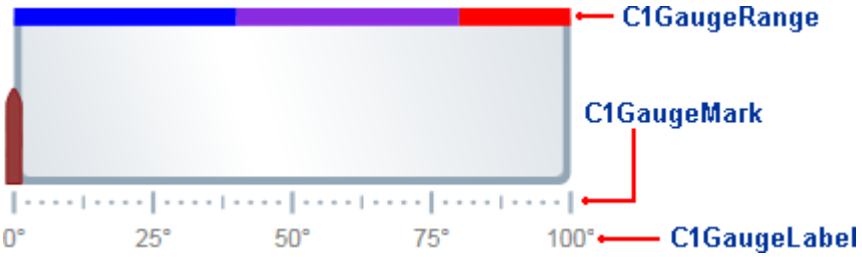
- Set **XAxisLocation** to **1**
- Set **XAxisLength** to **-1**

The scale will be reversed so 100 is at the top and 0 is at the bottom of the gauge.

A linear gauge has many applications. For example, a vertical linear gauge could be used as a thermometer such as the one displayed in the Using C1LinearGauge topic.

C1LinearGauge Decorators

By default, the C1LinearGauge control displays only a plain horizontal linear gauge. In most applications, you'll also want to display a scale composed of labels and tick marks that allow users to see what the current value is and where it lies within the gauge's range. This is done by adding C1GaugeMark, C1GaugeLabel, and C1GaugeRange elements to the gauge's C1Gauge.Decorators collection:



The decorators are displayed at specific positions on the scale, determined by the value of the C1GaugeDecorator.From, C1GaugeDecorator.To, and C1GaugeMark.Interval properties.

In the image above, you'll see three **C1GaugeMark** elements and one **C1GaugeLabel** element:

Markup

```
<!-- Add tick marks -->
<Gauge:C1GaugeMark From="0" To="100" Interval="25" Location="1.1" />
<Gauge:C1GaugeMark From="0" To="100" Interval="12.5" Location="1.1" />
<Gauge:C1GaugeMark From="0" To="100" Interval="2.5" Location="1.1" />
<!-- Add label marks -->
<Gauge:C1GaugeLabel Location="1.3" Interval="25" Foreground="Gray" Alignment="Center"
Format="0°" />
```

The **C1GaugeLabel** element shows labels every 25 units for the values 0 to 100 along the scale. The **C1GaugeMark** element shows tick marks spaced 25, 12.5, and 2.5 units apart.

In addition to showing the scale, you may want to highlight parts of the scale range. For example, you may want to add a red marker to indicate that values in that range are too low (sales) or too high (expenses). This can be done easily by adding one or more **C1GaugeRange** elements to the gauge's **C1Gauge.Decorators** collection.

In the image above, you'll see three **C1GaugeRange** elements:

Markup

```
<!-- Add three colored ranges -->
<Gauge:C1GaugeRange Fill="Blue" To="40" Width=".1" />
<Gauge:C1GaugeRange Fill="BlueViolet" From="40" To="80" Width=".1" />
<Gauge:C1GaugeRange Fill="Red" From="80" To="100" Width=".1" />
```

The **C1GaugeRange** elements show blue, blue violet, and red range areas. Each **C1GaugeRange** element displays a curved swath along the scale. The color of the swath is determined by the C1GaugeRange.Fill property, and the position is determined by the **C1GaugeDecorator.From** and **C1GaugeDecorator.To** properties. You can control the thickness of the ranges using the C1GaugeRange.Width property.

C1LinearGauge Decorators Location

Decorators are positioned with the C1LinearGauge control compared to the C1RadialGauge control. Recall that the C1RadialGauge control has a C1RadialGauge.Radius property that determines how far from the center of the gauge

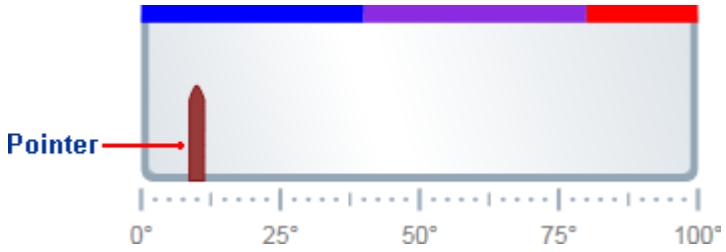
that decorators are displayed. The C1RadialGauge.Radius property ranges from zero (center of the gauge) to one (outer edge of the gauge). Individual decorators are offset from the C1RadialGauge.Radius by an amount specified by the **Location** property.

The **C1LinearGauge** has a C1LinearGauge.YAxisLocation property that is analogous to C1RadialGauge.Radius. This property ranges from zero (top of the gauge) to one (bottom of the gauge). Individual decorators are offset from the **C1LinearGauge.YAxisLocation** by an amount specified by their C1GaugeDecorator.Location property.

The default value for the **C1LinearGauge.YAxisLocation** property is zero. The default value for the **C1GaugeDecorator.Location** of the C1GaugeMark decorators is one (causing these elements to appear at the bottom of the gauge by default). The default value for the C1GaugeDecorator.Location property of the C1GaugeRange decorator is zero (causing it to appear at the top of the gauge by default).

C1LinearGauge Pointer

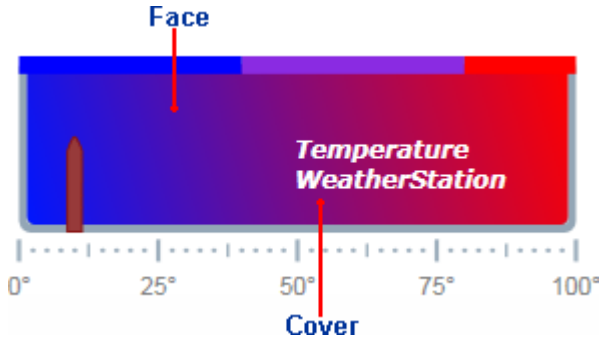
The **C1GaugeDecorator.Fr**control includes a pointer which indicates the selected C1Gauge.Value of the control. The pointer consists of the **C1GaugePointer** element:



The **C1GaugePointer** element appears by default as a brown tapering element, but you can customize the appearance of the **C1GaugePointer** element by setting several properties, including the C1Gauge.PointerFill, C1Gauge.PointerLength, C1Gauge.PointerOffset, C1Gauge.PointerStroke, C1Gauge.PointerStrokeThickness, C1Gauge.PointerStyle, C1Gauge.PointerVisibility, and C1Gauge.PointerWidth properties. You can also customize how the **C1GaugePointer** element appears by setting the C1LinearGaugePointer.Orientation property to **Vertical** or **Horizontal**.

C1LinearGauge Face and Cover

Like a watch or thermometer, the C1LinearGauge control includes a C1Gauge.Face and a C1Gauge.Cover. The **C1Gauge.Face** appears above the background but behind the pointer and other decorators, and the **C1Gauge.Cover** appears, like a glass over a thermometer, above all other elements. For example, in the image below the **C1Gauge.Face** includes a gradient that appears behind the elements in the gauge and the **C1Gauge.Cover** includes text that appears above the **C1Gauge.Face** and all other elements:



You can customize the appearance of the **C1Gauge.Cover** by using the C1Gauge.CoverTemplate and you can

customize the appearance of the **C1Gauge.Face** by using the C1Gauge.FaceTemplate.

Using C1Knob

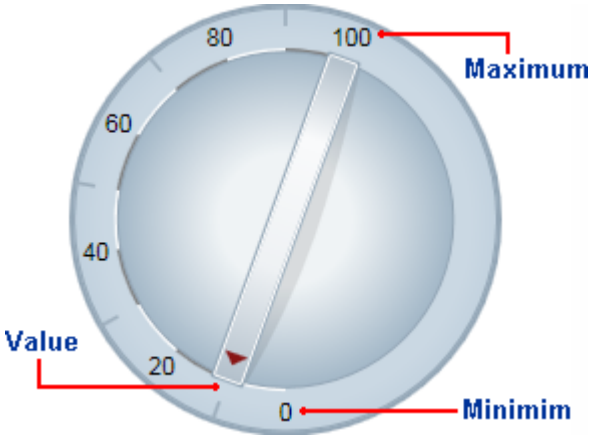
The C1Knob control extends a C1RadialGauge control to let the user select a numerical value by rotating the pointer. For example, **C1Knob** is perfect if you want to simulate the volume knob of a music player. By default, the **C1Knob** control appears similar to the following image:

Creating and using a **C1Knob** control is similar to creating a C1RadialGauge control and typically involves the similar steps:

1. Creating the **C1Knob** control and setting its main properties: **C1Gauge.Minimum**, **C1Gauge.Maximum**, **C1RadialGauge.StartAngle**, and **C1RadialGauge.SweepAngle**.
2. Setting how users interact with the knob, by setting the C1Knob.InteractionMode property.
3. Adding **C1GaugeMark** and **C1GaugeLabel** decorators to show the scale. Each element may show a set of labels, tick marks, or both.
4. Optionally customizing gauge elements with XAML templates.
5. Setting the **C1Gauge.Value** property to display the value you want to initially show.

C1Knob Values

You can use the C1Knob control's **C1Gauge.Minimum**, **C1Gauge.Maximum**, and **C1Gauge.Value** properties to specify the available range and the selected value in that range:



The **C1Gauge.Minimum** and **C1Gauge.Maximum** properties specify the range of values the knob is designed to show. The range is specified through the **C1Gauge.Minimum** and **C1Gauge.Maximum** properties (of type **double**). The default range for a **C1Knob** control is from 0 to 100.

The **C1Gauge.Value** property indicates the current value of the gauge. In the **C1Knob** control, this is indicated visually by the value the **C1GaugePointer** element is pointing to. The default **C1Gauge.Value** for a **C1Knob** control is 50.

C1Knob Start Angle and Sweep Angle

Once the range has been defined, you can specify the angles that match the **C1Gauge.Minimum** and **C1Gauge.Maximum** values. The **C1RadialGauge.StartAngle** defines the position of the pointer when the **C1Gauge.Value** property is set to the **C1Gauge.Minimum** value in the range. The **C1RadialGauge.SweepAngle** specifies the rotation added to the **C1RadialGauge.StartAngle** when the **C1Gauge.Value** property is set to the **C1Gauge.Maximum** value in the range.

All angles are specified in degrees, measured clockwise from the top of the control. The angles may be negative, but the absolute value of the **C1RadialGauge.SweepAngle** may not exceed 360 degrees.

C1Knob Interaction

The **C1Knob.InteractionMode** property controls what interactions are possible with the control at run time – you can choose whether users can move the knob by clicking, dragging, or both.

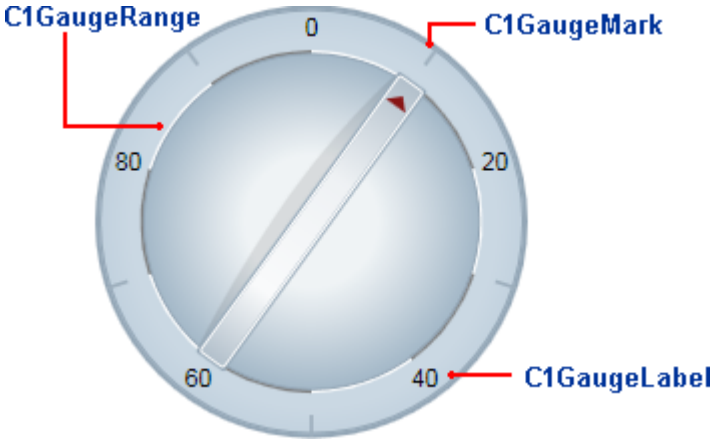
You can set the **C1Knob.InteractionMode** property to one of the following values:

Value	Description
Drag	Moves the pointer when the user drags it.
Click	Moves the pointer when the user clicks inside the knob.
ClickOrDrag	Moves the pointer when the user clicks inside the knob or drags the pointer.

By default the **C1Knob.InteractionMode** property is set to **Click**.

C1Knob Decorators

By default, the C1Knob control displays a simple blue-gray background and a pointer. Like with C1RadialGauge and C1LinearGauge, you can customize the indicators of the grid by adding the C1GaugeMark, C1GaugeLabel, and C1GaugeRange elements to the gauge's **C1Gauge.Decorators** collection:



The decorators are displayed at specific positions on the scale, determined by the value of the **C1GaugeDecorator.From, C1GaugeDecorator.To**, and **C1GaugeMark.Interval** properties.

In the image above, you'll see one **C1GaugeMark** and one **C1GaugeLabel** element:

Markup

```
<!-- Add tick marks -->
<Gauge:C1GaugeMark Interval="20" Alignment="In" From="10" />

<!-- Add label marks -->
<Gauge:C1GaugeLabel Interval="20" Alignment="Center" Location="0.9" To="80" />
```

The **C1GaugeLabel** element shows labels for the values 10 to 90 along the scale. The **C1GaugeMark** element shows tick marks spaced 20 units apart.

In addition to showing the scale, you may want to highlight parts of the scale range by adding one or more **C1GaugeRange** elements to the gauge's **C1Gauge.Decorators** collection.

In the image above, you'll see ten **C1GaugeRange** elements:

Markup

```
<!-- Add ten colored ranges -->
<Gauge:C1GaugeRange From="0" To="10" Location="0.7" Fill="White" />
<Gauge:C1GaugeRange From="10" To="20" Location="0.7" Fill="Gray" />
<Gauge:C1GaugeRange From="20" To="30" Location="0.7" Fill="White" />
<Gauge:C1GaugeRange From="30" To="40" Location="0.7" Fill="Gray" />
<Gauge:C1GaugeRange From="40" To="50" Location="0.7" Fill="White" />
<Gauge:C1GaugeRange From="50" To="60" Location="0.7" Fill="Gray" />
<Gauge:C1GaugeRange From="60" To="70" Location="0.7" Fill="White" />
<Gauge:C1GaugeRange From="70" To="80" Location="0.7" Fill="Gray" />
<Gauge:C1GaugeRange From="80" To="90" Location="0.7" Fill="White" />
<Gauge:C1GaugeRange From="90" To="100" Location="0.7" Fill="Gray" />
```

The **C1GaugeRange** elements show white and gray range areas. Each **C1GaugeRange** element displays a curved swath along the scale. The color of the swath is determined by the **C1GaugeRange.Fill** property, and the position is determined by the **C1GaugeDecorator.From** and **C1GaugeDecorator.To** properties You can control the thickness of the ranges using the **C1GaugeRange.StrokeThickness** property.

C1KnobDecorators Location

Each decorator element has a **C1GaugeDecorator.Location** property that determines where the elements are displayed. These properties range from zero (the center of the gauge) to one (the outer edge of the gauge). The gauge control also has a C1RadialGauge.Radius property that ranges from zero to one and affects the positioning of all decorators. The default value for the radius property is 0.8, which causes all decorators to be displayed entirely within the control.

For example, if you set the **C1GaugeDecorator.Location** property for the **C1GaugeLabel** was to 1.1 the labels will appear offset towards the outer edge of the knob. The labels are still drawn within the control because the C1RadialGauge.Radius property is set to 0.8 (the actual position of the labels in this case can be calculated as 1.1 * 0.8 = 0.88).

Gauges for UWP Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET and know how to use the gauge controls in general. If you are unfamiliar with the **Gauges for UWP** product, please see the Gauges for UWP Quick Start first.

Each topic in this section provides a solution for specific tasks using the **Gauges for UWP** product.

Each task-based help topic also assumes that you have created a new UWP project and added a gauge control to the project.

Setting the Start Value

In this topic you'll change the **C1LinearGauge** control's **C1Gauge.Value** property. The **C1Gauge.Value** property determines the currently selected number. By default the **C1LinearGauge** control starts with its **C1Gauge.Value** set to 0 but you can customize this number at design time, in XAML, and in code. Note that although this topic sets the **C1Gauge.Value** of the **C1LinearGauge** control, the same steps can be used to customize the **C1Gauge.Value** of other controls.

At Design Time

To set the **C1LinearGauge** control's **C1Gauge.Value** property at run time, complete the following steps:

1. Click the **C1LinearGauge** control once to select it.
2. Navigate to the Properties window, and enter a number, for example "20", in the text box next to the **C1Gauge.Value** property.

   This will set the **C1Gauge.Value** property to the number you chose.

In XAML

For example, to set the **C1Gauge.Value** property add Value="20" to the <Gauge:C1LinearGauge> tag so that it appears similar to the following:

Markup

```
<Gauge:C1LinearGauge Height="89" Margin="47,57,33,43" Name="C1LinearGauge1"
Width="287" Value="20">
```

In Code

For example, to set the **C1Gauge.Value** property, add the following code to your project:

Visual Basic

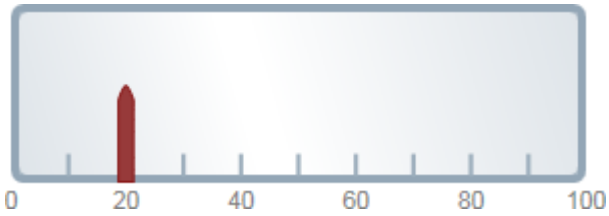
```
C1LinearGauge1.Value = 20
```

C#

```
C1LinearGauge1.Value = 20;
```

Run your project and observe:

Initially the gauge's **C1GaugePointer** will be set to the **C1Gauge.Value** you selected:



Setting the Minimum and Maximum Values

You can use the C1Gauge.Minimum and C1Gauge.Maximum properties to set a numeric range that the gauge would be limited to. You can customize the C1Gauge.Minimum and C1Gauge.Maximum values at design time, in XAML, and in code. Although this topic sets the C1Gauge.Minimum and C1Gauge.Maximum properties of the C1LinearGauge control, the same steps can be used to customize the C1Gauge.Minimum and C1Gauge.Maximum of other controls.

> When setting the C1Gauge.Minimum and C1Gauge.Maximum properties, the C1Gauge.Minimum should be smaller than the C1Gauge.Maximum. Also be sure to set the C1Gauge.Value property to a number within the C1Gauge.Minimum and C1Gauge.Maximum range (here the default is 0, which falls within the range set below).

At Design Time

To set the C1Gauge.Minimum and C1Gauge.Maximum for the C1LinearGauge at run time, complete the following steps:

1. Click the C1LinearGauge control once to select it.

2. Navigate to the Properties window, and enter a number, for example 50, next to the C1Gauge.Maximum property.

3. In the Properties window, enter a number, for example -50, next to the C1Gauge.Minimum property.

This will set C1Gauge.Minimum and C1Gauge.Maximum values.

In XAML

To set the C1LinearGauge control's C1Gauge.Minimum and C1Gauge.Maximum in XAML add Maximum="50" Minimum="-50" to the <Gauge:C1LinearGauge> tag so that it appears similar to the following:

Markup

```
<Gauge:C1LinearGauge Height="89" Margin="47,57,33,43" Name="C1LinearGauge1"
Width="287" Maximum="50" Minimum="-50">
```

In Code

To set the **C1LinearGauge** control's **C1Gauge.Minimum** and **C1Gauge.Maximum** add the following code to your project:

Visual Basic

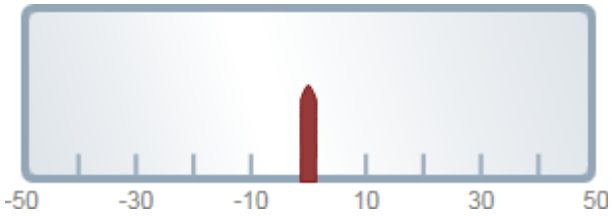
```
C1LinearGauge1.Minimum = -50
C1LinearGauge1.Maximum = 50
```

C#

```
C1LinearGauge1.Minimum = -50;
C1LinearGauge1.Maximum = 50;
```

Run your project and observe:

The gauge will be limited to the selected range at run time:



Adding Labels to the Gauge

You can add and customize the C1RadialGauge control's labeling in the Properties window, XAML, or through code. Although this topic sets the **C1GaugeLabel** properties of the **C1RadialGauge** control, the same steps can be used to customize the C1GaugeLabel of other controls.

At Design Time

To add labeling to the **C1RadialGauge** control in the Properties window at design time, complete the following steps:

1. Click the **C1RadialGauge** control once to select it.
2. Navigate to the Properties window, and click the ellipsis button next to the Decorators item. The Decorators collection editor will open.
3. Choose **C1GaugeLabel** in the drop-down list in the top-left of the editor and click the Add button. A **C1GaugeLabel** decorator will be added to the collection and will be selected.
4. In the right-side properties pane, set the **C1GaugeLabel** element's **C1GaugeDecorator.Location** to 1. Set the **C1GaugeLabel** element's **C1GaugeLabel.Interval** to 20.

This will set the control's label.

In XAML

To add labeling to the **C1RadialGauge** control in XAML add the <Gauge:C1GaugeLabel> tag to the <Gauge:C1RadialGauge> tag so that it appears similar to the following:

Markup

```
<Gauge:C1RadialGauge Height="189" Margin="42,29,188,31" Name="C1RadialGauge1"
Width="189">
     <Gauge:C1GaugeLabel Interval="20" Location="1" />
 </Gauge:C1RadialGauge>
```

In Code

Right-click the window and select View Code to open the Code Editor. Add code to the main class, so it appears similar to the following:

Visual Basic

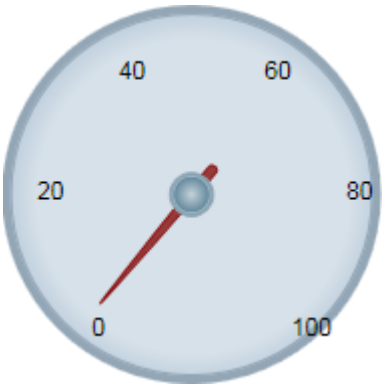
```
Public Sub New()
 InitializeComponent()
     Dim c1gl As New C1.Xaml.Gauge.C1GaugeLabel
     c1gl.Location = 1
     c1gl.Interval = 20
     Me.C1RadialGauge1.Decorators.Add(c1gl)
 End Sub
```

C#

```
public MainPage(){
 InitializeComponent();
     C1.Xaml.Gauge.C1GaugeLabel c1gl = new C1.Xaml.Gauge.C1GaugeLabel();
     c1gl.Location = 1;
     c1gl.Interval = 20;
     this.C1RadialGauge1.Decorators.Add(c1gl);
 }
```

Run your project and observe:

The **C1RadialGauge** control will appear with labeling:



Adding Tick Marks to the Gauge

You can add tick marks to the C1LinearGauge control through the Properties window, XAML, or through code. Although this topic sets the **C1GaugeMark** properties of the **C1LinearGauge** control, the same steps can be used to customize the C1GaugeMark of other controls.

At Design Time

To add tick marks to the **C1LinearGauge** control in the Properties window at design time, complete the following steps:

1. Click the **C1LinearGauge** control once to select it.
2. Navigate to the Properties window, and click the ellipsis button next to the Decorators item. The Decorators collection editor will open.
3. Choose **C1GaugeMark** in the drop-down list in the top-left of the editor and click the Add button. A **C1GaugeMark** decorator will be added to the collection and will be selected.
4. In the right-side properties pane, set the **C1GaugeMark** element's **C1GaugeDecorator.Location** to 1.1.
5. Set the **C1GaugeLabel** element's **C1GaugeLabel.Interval** to 20.
6. Choose **C1GaugeMark** in the drop-down list in the top-left of the editor and click the Add button. A second **C1GaugeMark** decorator will be added to the collection and will be selected.
7. In the right-side properties pane, set the **C1GaugeMark** element's **C1GaugeDecorator.Location** to 1.1.
8. Set the **C1GaugeLabel** element's **C1GaugeLabel.Interval** to 10.
9. Choose **C1GaugeMark** in the drop-down list in the top-left of the editor and click the Add button. A third **C1GaugeMark** decorator will be added to the collection and will be selected.
10. In the right-side properties pane, set the **C1GaugeMark** element's **C1GaugeDecorator.Location** to 1.1.
11. Set the **C1GaugeLabel** element's **C1GaugeLabel.Interval** to 5.

In XAML

To add labeling to the **C1LinearGauge** control in XAML add three <Gauge:C1GaugeMark> tags to the

<Gauge:C1LinearGauge> tag so that it appears similar to the following:

Markup

```
<Gauge:C1LinearGauge Height="89" Margin="90,72,41,88" Name="C1LinearGauge1"
Width="287">
    <Gauge:C1GaugeMark Interval="20" Location="1.1" />
    <Gauge:C1GaugeMark Interval="10" Location="1.1" />
    <Gauge:C1GaugeMark Interval="5" Location="1.1" />
</Gauge:C1LinearGauge>
```

In Code

Right-click the window and click View Code to switch to the Code Editor. And add code to the main class, so it appears similar to the following:

Visual Basic

```
Public Sub New()
InitializeComponent()
    Dim c1gm1 As New C1.Xaml.Gauge.C1GaugeMark
    c1gm1.Location = 1.1
    c1gm1.Interval = 20
    Me.C1LinearGauge1.Decorators.Add(c1gm1)
    Dim c1gm2 As New C1.Xaml.Gauge.C1GaugeMark
    c1gm2.Location = 1.1
    c1gm2.Interval = 10
    Me.C1LinearGauge1.Decorators.Add(c1gm2)
    Dim c1gm3 As New C1.Xaml.Gauge.C1GaugeMark
    c1gm3.Location = 1.1
    c1gm3.Interval = 5
    Me.C1LinearGauge1.Decorators.Add(c1gm3)
End Sub
```

C#

```
public MainPage(){
InitializeComponent();
    C1.Xaml.Gauge.C1GaugeLabel c1gm1 = new C1.Xaml.Gauge.C1GaugeMark();
    c1gm1.Location = 1.1;
    c1gm1.Interval = 20;
    this.C1LinearGauge1.Decorators.Add(c1gm1);
    C1.Xaml.Gauge.C1GaugeLabel c1gm2 = new C1.Xaml.Gauge.C1GaugeMark();
    c1gm2.Location = 1.1;
    c1gm2.Interval = 10;
    this.C1LinearGauge1.Decorators.Add(c1gm2);
    C1.Xaml.Gauge.C1GaugeLabel c1gm3 = new C1.Xaml.Gauge.C1GaugeMark();
    c1gm3.Location = 1.1;
    c1gm3.Interval = 5;
    this.C1LinearGauge1.Decorators.Add(c1gm3);
}
```

Run your project and observe:

The **C1LinearGauge** control will appear with tick marks of three sizes:

Customizing Tick Marks

By default, gauge marks are drawn as blue-gray rectangles. You can customize their appearance by assigning a custom template to the C1GaugeMark.Template property of the C1GaugeMark element. In the following steps, you'll create a new DataTemplate which defines the **C1GaugeMark** appearance and then you'll assign that template to the **C1GaugeMark** element's **C1GaugeMark.Template** property in the C1RadialGauge control.

Complete the following steps:

1. Switch to XAML view and add three <Gauge:C1GaugeMark> tags to the <Gauge:C1RadialGauge> tag so that it appears similar to the following:

Markup

```
<Gauge:C1RadialGauge Height="89" Margin="90,72,41,88" Name="C1RadialGauge1"
Width="287">
    <Gauge:C1GaugeMark From="0" To="100" Interval="10" />
    <Gauge:C1GaugeMark Interval="5" Location="1.1" />
</Gauge:C1RadialGauge>
```

2. Add the following markup just under the UserControl tag to add a template:

Markup

```
<UserControl.Resources>
    <!-- Template used to render the gauge marks -->
    <DataTemplate x:Key="MyMarkTemplate">
        <Rectangle Width="4" Height="18" Fill="BlueViolet" Stroke="Black"
StrokeThickness=".5"/>
    </DataTemplate>
</UserControl.Resources>
```

This template defines the appearance of the tick marks.

3. Next, set the **C1GaugeMark.Template** property on the first **C1GaugeMark** element you added to reference the new template's key:

Markup

```
<Gauge:C1GaugeMark From="0" To="100" Interval="10" Template="{StaticResource
MyMarkTemplate}"/>
```

Run your project and observe:

The **C1RadialGauge** control appears with custom tick marks:

Notice that the marks are drawn from the position specified by the C1GaugeDecorator.Location property and grow inward. If you increase the Height of the rectangles used to show the marks, the tick marks will extend farther toward the center of the gauge. To make them extend out you would change the C1GaugeDecorator.Location property on the C1GaugeMark element. Also, notice how elements used to show tick marks are rotated along the scale; elements used to show labels are not (see Adding Labels to the Gauge).

Customizing the Gauge Shape

Most radial gauges are circular, but you can create gauges with other shapes as well. To customize the shape of a C1RadialGauge, you would need to:

- Choose a shape for the gauge.
- Set the C1RadialGauge.PointerOrigin property to match the position of the pointer taking into account the gauge shape.
- Hide the default round background by setting the gauge's Background property to Transparent and the BorderThickness to 0.
- Add elements to the **C1Gauge.Face** layer to show the new gauge shape.

Complete the following steps to follow the steps above to create a **C1RadialGauge** with a customized shape:

1. Switch to XAML view and modify the <Gauge:C1RadialGauge> tag so that it appears similar to the following:

   Markup

   ```
   <Gauge:C1RadialGauge Height="189" Margin="102,34,127,26" Name="C1RadialGauge1"
   Width="189" StartAngle="-160" SweepAngle = "140">
   </Gauge:C1RadialGauge>
   ```

   This will set the **C1RadialGauge** control's initial properties.

2. In XAML view add PointerOrigin="0.8,0.5" to the <Gauge:C1RadialGauge> tag so that it appears similar to the following:

   Markup

   ```
   <Gauge:C1RadialGauge Height="189" Margin="102,34,127,26" Name="C1RadialGauge1"
   Width="189" StartAngle="-160" SweepAngle = "140" PointerOrigin="0.8,0.5">
   </Gauge:C1RadialGauge>
   ```

   The **C1RadialGauge.PointerOrigin** property will set where the **C1RadialGauge** control's **C1GaugePointer** originates.

3. In XAML view add Background="Transparent" to the <Gauge:C1RadialGauge> tag so that it appears similar to the following:

```
Markup
<Gauge:C1RadialGauge Height="189" Margin="102,34,127,26" Name="C1RadialGauge1"
Width="189" StartAngle="-160" SweepAngle = "140" PointerOrigin="0.8,0.5"
Background="Transparent">
</Gauge:C1RadialGauge>
```

   The **C1RadialGauge** control will now appear transparent.

4. In XAML view add BorderThickness="0" to the <Gauge:C1RadialGauge> tag so that it appears similar to the following:

```
Markup
<Gauge:C1RadialGauge Height="189" Margin="102,34,127,26" Name="C1RadialGauge1"
Width="189" StartAngle="-160" SweepAngle = "140" PointerOrigin="0.8,0.5"
Background="Transparent" BorderThickness="0">
</Gauge:C1RadialGauge>
```

   The **C1RadialGauge** control will now appear without a border.

5. In XAML view add markup after the <Gauge:C1RadialGauge> tag so that it appears similar to the following:

```
Markup
<Gauge:C1RadialGauge Height="189" Margin="102,34,127,26" Name="C1RadialGauge1"
Width="189" StartAngle="-160" SweepAngle = "140" PointerOrigin="0.8,0.5"
Background="Transparent" BorderThickness="0">
    <!-- Add tick marks to the gauge -->
    <Gauge:C1GaugeMark Interval="10" Location="1"/>
    <Gauge:C1GaugeMark Interval="5" Location="1" />
</Gauge:C1RadialGauge>
```

   This will add **C1GaugeMark** elements and tick marks to the gauge.

6. In XAML view add markup after the <Gauge:C1RadialGauge> tag so that it appears similar to the following:

```
Markup
<Gauge:C1RadialGauge Height="189" Margin="102,34,127,26" Name="C1RadialGauge1"
Width="189" StartAngle="-160" SweepAngle = "140" PointerOrigin="0.8,0.5"
Background="Transparent" BorderThickness="0">
    <!-- Add tick marks to the gauge -->
    <Gauge:C1GaugeMark Interval="10" Location="1"/>
    <Gauge:C1GaugeMark Interval="5" Location="1" />
    <!-- Add a face with custom shape -->
    <Gauge:C1RadialGauge.Face>
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="4*" />
                <ColumnDefinition Width="10*" />
                <ColumnDefinition Width="1*" />
```

```
            </Grid.ColumnDefinitions>
            <Border Grid.Column="1" Background="Black" BorderBrush="LightGray"
BorderThickness="4" CornerRadius="140,60,60,140"/>
        </Grid>
    </Gauge:C1RadialGauge.Face>
</Gauge:C1RadialGauge>
```

This will add a customized **C1Gauge.Face** to the gauge.

Run your project and observe:

The **C1RadialGauge** control appears with a customized face:

You can customize the **C1Gauge.Face** of the **C1RadialGauge** control even further. For example, take a look at the following customized gauges included in the SpeedometersPage.xaml page of the GaugeSamples sample installed with **UWP Edition**:

Customizing the Pointer's Appearance

By default, the C1GaugePointer appears as a tapered brown rectangle and the pointer cap appears as a gray circle with a gradient. You can customize the appearance of both. In the following steps, you'll customize the appearance of the C1RadialGauge control's **C1GaugePointer** and C1RadialGauge.PointerCap.

Complete the following steps:

1. Click once on the **C1RadialGauge** control to select it.
2. Switch to XAML view and add PointerFill="SkyBlue" PointerStroke="CornflowerBlue" to the <Gauge:C1RadialGauge> tag so that it appears similar to the following:

Markup
`<Gauge:C1RadialGauge Height="226" Margin="22,24,0,12" Name="C1RadialGauge1"`

```
Width="256" PointerFill="SkyBlue" PointerStroke="CornflowerBlue">
</Gauge:C1RadialGauge>
```

This will set customize the color of the **C1GaugePointer**.

3. In XAML view add PointerCapStroke="CornflowerBlue" to the <Gauge:C1RadialGauge> tag so that it appears similar to the following:

Markup

```
<Gauge:C1RadialGauge Height="226" Margin="22,24,0,12" Name="C1RadialGauge1"
Width="256" PointerFill="SkyBlue" PointerCapStroke="CornflowerBlue"
PointerStroke="CornflowerBlue">
</Gauge:C1RadialGauge>
```

This will customize the color that the C1RadialGauge.PointerCap is outlined in.

4. In XAML view add the following <Gauge:C1RadialGauge.PointerCapFill> markup to the <Gauge:C1RadialGauge> tag so that it appears similar to the following:

Markup

```
<Gauge:C1RadialGauge Height="226" Margin="22,24,0,12" Name="C1RadialGauge1"
Width="256" PointerFill="SkyBlue" PointerCapStroke="CornflowerBlue"
PointerStroke="CornflowerBlue" >
    <Gauge:C1RadialGauge.PointerCapFill>
        <LinearGradientBrush>
            <GradientStop Color="CornflowerBlue" Offset="0"/>
            <GradientStop Color="SkyBlue" Offset="1"/>
        </LinearGradientBrush>
    </Gauge:C1RadialGauge.PointerCapFill>
</Gauge:C1RadialGauge>
```

This will add a linear gradient to the **C1RadialGauge** control's **C1RadialGauge.PointerCap**.

Run your project and observe:

The **C1RadialGauge** control appears with a customized **C1GaugePointer** and **C1RadialGauge.PointerCap**: