Gauges for WPF/Silverlight

2018.04.11 更新

グレープシティ株式会社

Gauges for WPF/Silverlight

目次

製品の概要

Gauges for WPF/Silverlight には、データの視覚化とビジネスダッシュボードの機能を強化する7つのコントロールが含まれます。情報をグラフィカルに表示するインタラクティブで魅力的な方法を提供します。

C1.WPF.Gauge/C1.Silverlight.Gauge アセンブリには、主要な3つのコントロールが含まれます。

- **C1RadialGauge**
  回転型のポインタを使用し、曲線目盛りに沿って値を表示します。これは、典型的なスピードメーターに似ています。
- **C1LinearGauge**
  直線型のポインタを使用し、直線目盛りに沿って値を表示します。これは、典型的な温度計に似ています。
- **C1Knob**
  C1RadialGauge を拡張します。ユーザーがポインタを回転して数値を選択できます。音楽プレイヤーのボリュームつまみを真似る場合に最適です。

C1RadialGauge と **C1LinearGauge** は、すべてのゲージに共通する基本機能を備えた一般的な抽象クラス **C1Gauge** から派生されます。

メモ: 説明内に含まれるクラスおよびメンバーに対するリファレンスへのリンクは、原則としてWPF版のリファレンスページを参照します。Silverlight版については、目次から同名のメンバーを参照してください。

主な特長

Gauges for WPF/Silverlight を使用して、機能豊富でカスタマイズされたアプリケーションを作成できます。以下に示す主要な機能をうまく活用して、**Gauges for WPF/Silverlight** を最大限に使用してください。

- **さまざまな範囲タイプ**
  非直線型または直線型の範囲を作成します。非直線型を使用して、値の変化を表したり、見やすいゲージを作成することができます。

- **見栄えのする直線型ゲージの生成**
  一般的な直線型ゲージとして、定規や温度計があります。単純なプロパティを使用して、スケール、目盛りマーク、範囲、およびポインタをカスタマイズできます。

- **本格的な円形ゲージの作成**
  一般的な円形ゲージとして、文字盤やスピードメーターがあります。単純なプロパティを使用して、開始/移動角度、目盛りマーク、範囲、およびポインタをカスタマイズできます。

- **オフモードのサポート**
  値がない場合は、範囲外にオフポジションを設定できます。

- **ノブ型ゲージの作成**
  C1Knob コントロールは、エンドユーザーがポインタを特定の値にドラッグできるように、**C1RadialGauge** を拡張しています。

- **ラベル書式のサポート**
  ラベルを書式設定します。たとえば、通貨書式(\)で値を表示するようにラベルを設定できます。

- **XAML の作業が不要な広範囲なカスタマイズ**
  C1LinearGauge および C1RadialGauge には、XAML テンプレートを変更する必要がない広範囲なカスタマイズを提供するプロパティが含まれます。もちろん、XAML を使ってさらにカスタマイズすることもできます。

- **その他のコントロール**
  Gauges for WPF/Silverlight は、使い慣れたゲージコントロールの書式設定時に使用できるいくつかの追加コントロールを提供します。これには、**C1SpeedometerGauge**、**C1VolumeGauge**、**C1RulerGauge**、および **C1RegionKnob** コントロールが含まれます。

Gauges for WPF クイックスタート

このクイックスタートは、**Gauges for WPF** を初めて使用するユーザーのために用意されています。このクイックスタートでは、Visual Studio で新しいプロジェクトを作成し、アプリケーションに **Gauges for WPF** コントロールを追加して、コントロールの外観と動作をカスタマイズします。

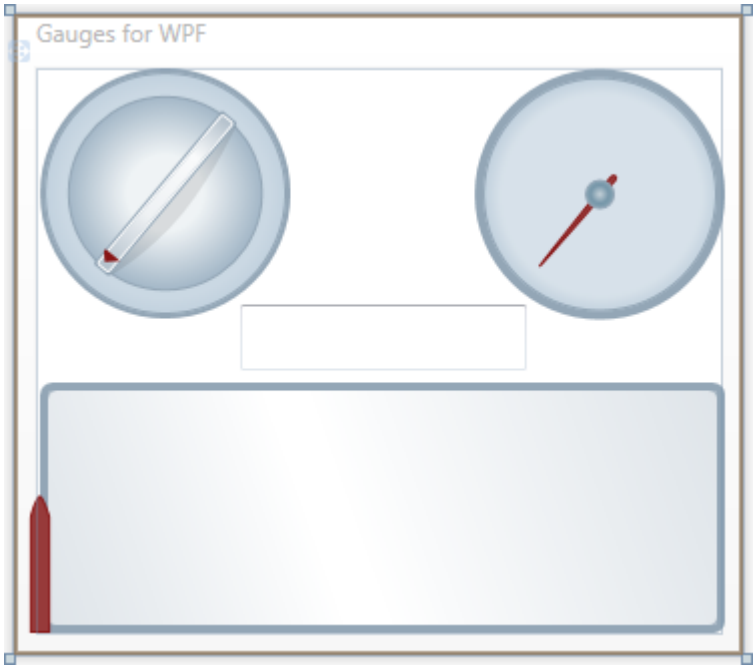
C1RadialGauge、**C1LinearGauge**、および **C1Knob** コントロールを含むアプリケーションを作成します。実行時にユーザーがノブで値を変更すると、ゲージの値も変更されます。

手順 1:アプリケーションの設定

この手順では、最初に Visual Studio で **Gauges for WPF** を使用する WPF アプリケーションを作成します。この手順では、プロジェクトに **C1RadialGauge**、**C1LinearGauge**、および **C1Knob** コントロールを追加し、ゲージコントロールの現在値を表示する標準の **TextBox** コントロールを追加して、アプリケーションを設定します。

プロジェクトをセットアップし、ゲージコントロールをアプリケーションに追加するには、次の手順に従います。

1. Visual Studio で新しい WPF プロジェクトを作成します。
2. ソリューションエクスプローラでプロジェクトを右クリックし、**[参照の追加]**を選択します。
3. **[参照の追加]**ダイアログボックスで、**C1.WPF.4.dll**、**C1.WPF.Gauge.4.dll** の各アセンブリを見つけて選択します。**[OK]**をクリックして、これらのアセンブリへの参照をアプリケーションに追加します。
4. ツールボックスに移動し、次のアイコンをダブルクリックして、Window1 にこれらのコントロールを追加します。
    - **C1RadialGauge**
    - **C1LinearGauge**
    - **C1Knob**
    - **C1TextBox**
5. ウィンドウのサイズを変更し、ウィンドウ内にコントロールを次のように配置します。



ここでは、WPF アプリケーションを作成し、**Gauges for WPF** コントロールをアプリケーションに追加し、これらのコントロールを

カスタマイズしました。これで、アプリケーションのユーザーインターフェイスが正しく設定されました。次の手順では、コードをアプリケーションに追加します。

手順 2:コードの追加

前の手順では、新しい WPF プロジェクトを作成し、アプリケーションにいくつかの **Gauges for WPF** コントロールを追加しました。この手順では、アプリケーションにコードを追加してカスタマイズします。

次の手順に従います。

1. **TextBox1** をダブルクリックしてコードビューに切り替え、**TextBox1_TextChanged** イベントハンドラを作成します。
2. 次の imports 文をページの先頭に追加します。

VisualBasic

```
using C1.WPF;
using C1.WPF.Gauge;
```

C#

```
Imports C1.WPF
Imports C1.WPF.Gauge
```

3. **TextBox1_TextChanged** イベントハンドラにコードを追加します。次のようになります。

VisualBasic

```
Private Sub TextBox1_TextChanged(ByVal sender As System.Object, ByVal e As
System.Windows.Controls.TextChangedEventArgs) Handles TextBox1.TextChanged
Me.C1LinearGauge1.Value = Me.TextBox1.Text Me.C1RadialGauge1.Value =
Me.TextBox1.Text Me.C1Knob1.Value = Me.TextBox1.Text End Sub
```

C#

```
private void textBox1_TextChanged(object sender, TextChangedEventArgs e)
{
    this.c1LinearGauge1.Value = Convert.ToDouble(this.textBox1.Text);
    this.c1RadialGauge1.Value = Convert.ToDouble(this.textBox1.Text);
    this.c1Knob1.Value = Convert.ToDouble(this.textBox1.Text);
}
```

実行時にテキストボックスに値が入力されると、ゲージコントロールの値はその値に設定されます。

4. **[表示]**→**[デザイナ]**を選択してデザインビューに戻ります。
5. **C1Knob1** をクリックして選択し、**[プロパティ]**ウィンドウに移動します。
6. **[プロパティ]**ウィンドウの**[イベント]**(稲妻)ボタンをクリックしてイベントを表示します。

7. **ValueChanged** イベントの横にあるボックスをダブルクリックします。これにより、コードビューに切り替わり、**C1Knob1_ValueChanged** イベントハンドラが作成されます。

8. **C1Knob1_ValueChanged** イベントハンドラに、ゲージとテキストボックスコントロールの値を設定するコードを入力します。次のようになります。

VisualBasic

```
Private Sub C1Knob1_ValueChanged(ByVal sender As System.Object, ByVal e As
C1.WPF.PropertyChangedEventArgs(Of System.Double)) Handles C1Knob1.ValueChanged
Me.C1LinearGauge1.Value = Me.C1Knob1.Value Me.C1RadialGauge1.Value =
Me.C1Knob1.Value Me.TextBox1.Text = Me.C1Knob1.Value.ToString End Sub
```

C#

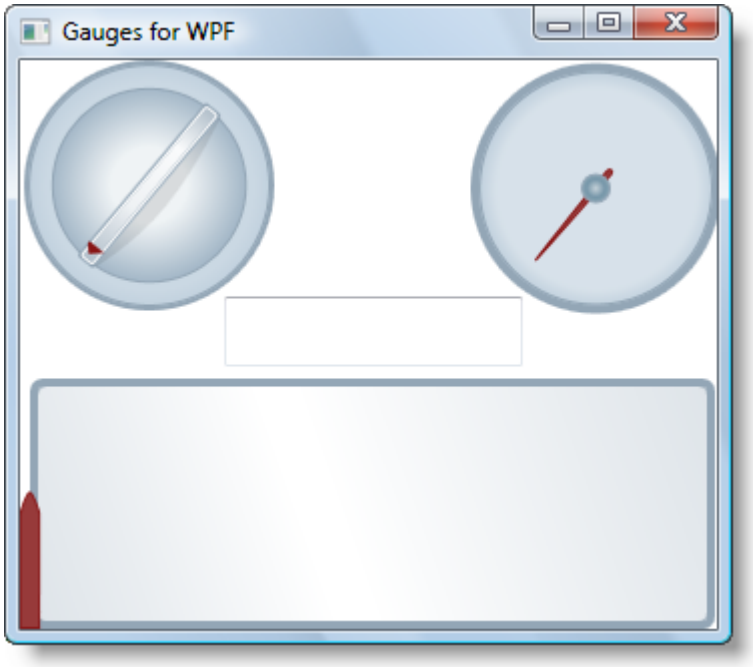
```
private void c1Knob1_ValueChanged(object sender, PropertyChangedEventArgs e)
{
    this.c1LinearGauge1.Value = this.c1Knob1.Value;
    this.c1RadialGauge1.Value = this.c1Knob1.Value;
    this.textBox1.Text = Convert.ToString(this.c1Knob1.Value);
}
```

この手順では、アプリケーションにコードを追加しました。次の手順では、アプリケーションを実行し、実行時の操作を確認します。

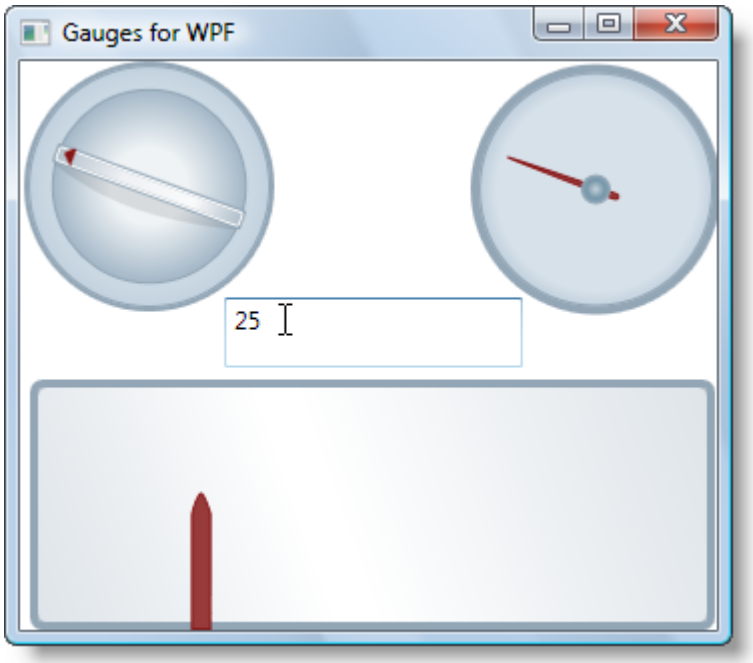
手順 3：アプリケーションの実行

これまでに WPF アプリケーションを作成し、外観と動作をカスタマイズしたので、次にアプリケーションを実行します。アプリケーションを実行し、**Gauges for WPF** の実行時の動作を確認するには、次の手順に従います。

1. **[デバッグ]**メニューから**[デバッグ開始]**を選択し、実行時にアプリケーションがどのように表示されるかを確認します。アプリケーションは次の図のように表示されます。

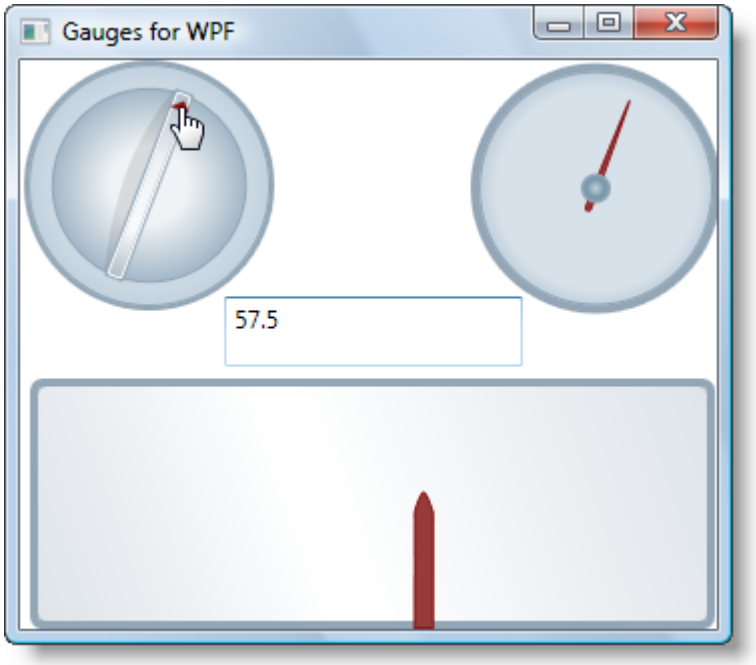


2. テキストボックスに値（**25** など）を入力します。**C1Knob**、**C1RadialGauge**、および **C1LinearGauge** コントロールの値が変化することに注目してください。



デフォルトでは、ゲージコントロールの **Minimum** プロパティは 0 に設定されており、Maximum は **100** に設定されています。したがって、**Value** が 25 に設定されると、ゲージは最初の 1/4 の位置を示します。

3. **C1Knob** コントロールをクリックし、Value を変更してみてください。他のゲージの値も同様に変更されて、現在の値がテキストボックスに表示されることがわかります。



おめでとうございます。**Gauges for WPF** のクイックスタートはこれで終了です。C1RadialGauge、C1LinearGauge、および C1Knob コントロールを使用してアプリケーションを作成し、アプリケーションの実行時の機能をいくつか確認しました。

Gauges for Silverlight クイックスタート

このクイックスタートは、**Gauges for Silverlight** を初めて使用するユーザーのために用意されています。このクイックスタートでは、Visual Studio で新しいプロジェクトを作成し、アプリケーションに **Gauges for Silverlight** コントロールを追加して、コントロールの外観と動作をカスタマイズします。

C1RadialGauge、**C1LinearGauge**、および **C1Knob** コントロールを含む単純なアプリケーションを作成します。実行時にユーザーがノブで値を変更すると、ゲージの値も変更されます。

手順 1:アプリケーションの作成

この手順では、Visual Studio で **Gauges for Silverlight** を使用する Silverlight アプリケーションを作成し、**StackPanel** パネルを追加して、アプリケーションに追加するコントロールのレイアウトをカスタマイズします。

プロジェクトを設定するには、次の手順に従います。

1. Visual Studio で、**[ファイル]**→**[新しいプロジェクト]**を選択します。
2. **[新しいプロジェクト]**ダイアログボックスで、左ペインから言語を選択し、テンプレートリストから**[Silverlight アプリケーション]**を選択します。プロジェクトの**名前**を入力し、**[OK]**をクリックします。**[新しい Silverlight アプリケーション]**ダイアログボックスが表示されます。
3. **[OK]**をクリックすると、**[新しい Silverlight アプリケーション]**ダイアログボックスが閉じ、プロジェクトが作成されます。
4. ソリューションエクスプローラでプロジェクトを右クリックし、**[参照の追加]**を選択します。
5. **[参照の追加]**ダイアログボックスで、**C1.Silverlight.dll** および **C1.Silverlight.Gauge.dll** アセンブリを見つけて選択します。**[OK]**をクリックして、これらのアセンブリへの参照をアプリケーションに追加します。
6. ツールボックスに移動し、**[StackPanel]**アイコンをダブルクリックして、**MainPage.xaml** にパネルを追加します。XAML マークアップは次のようになります。

```
XAML
<UserControl
 xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
 x:Class="C1Gauges.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
 xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
 mc:Ignorable="d" d:DesignWidth="640" d:DesignHeight="480">
  <Grid x:Name="LayoutRoot">
  <StackPanel></StackPanel>
  </Grid>
```

7. <StackPanel> タグに **x:Name="sp1" Width="Auto" Height="Auto" Orientation="Vertical" HorizontalAlignment="Center" VerticalAlignment="Center"** を追加します。次のようになります。

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

これで、パネル内の要素は、中央で縦方向に配置されて表示されます。

8. プロジェクトの XAML ウィンドウで、カーソルを タグと タグの間に置きます。
9. ツールボックスに移動し、**[StackPanel]**アイコンをダブルクリックして、既存の **StackPanel** にパネルを追加します。
10. <StackPanel> タグに **x:Name="sp2" Width="Auto" Height="Auto" Orientation="Horizontal" HorizontalAlignment="Center" VerticalAlignment="Center"** を追加します。次のようになります。

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

これで、パネル内の要素は、中央で横方向に配置されて表示されます。

これで、新しい Silverlight プロジェクトが作成され、アプリケーションが正しく設定されました。次の手順では、いくつかの **Gauges for Silverlight** コントロールをアプリケーションに追加し、これらのコントロールをカスタマイズします。

手順 2:コントロールの追加

この手順では、プロジェクトに **C1RadialGauge**、**C1LinearGauge**、および **C1Knob** コントロールを追加し、ゲージコントロールの現在値を表示する標準の **TextBox** コントロールを追加して、アプリケーションを設定します。

これらのゲージコントロールをアプリケーションに追加するには、次の手順に従います。

1. プロジェクトの XAML ウィンドウで、カーソルを <StackPanel x:Name="sp2"> タグと </StackPanel> タグの間に置きます。
2. ツールボックスに移動し、**[C1Knob]**アイコンをダブルクリックして、**StackPanel** にコントロールを追加します。XAML マークアップは次のようになります。

```
XAML
<UserControl
 xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
 x:Class="C1Gauges.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
 xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
 mc:Ignorable="d" d:DesignWidth="640" d:DesignHeight="480">
  <Grid x:Name="LayoutRoot">
   <StackPanel x:Name="sp1" Width="Auto" Height="Auto" Orientation="Vertical"

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

     HorizontalAlignment="Center" VerticalAlignment="Center">
     <c1:C1Knob></c1:C1Knob>
    </StackPanel>
   </StackPanel>
  </Grid>
</UserControl>
```

C1.Silverlight.Gauge 名前空間と <c1:C1Knob></c1:C1Knob> タグがプロジェクトに追加されていることがわかりま

す。

3. **x:Name="c1kb1"** を <c1:C1Knob> タグに追加して、コントロールに名前を付けます。次のようになります。

```
XAML
<c1:C1Knob x:Name="c1kb1">
```

それに一意の識別子を付けると、コードでそのコントロールにアクセスできるようになります。

4. <c1:C1Knob> タグに **Width="150"** を追加して、コントロールをサイズ変更します。次のようになります。

```
XAML
  <c1:C1Knob x:Name="c1kb1" Width="150">
```

アプリケーションを実行すると、このコントロールは少し小さく表示されます。

5. <c1:C1Knob> タグに **Margin="5"** を追加して、コントロールにマージンを追加します。次のようになります。

```
XAML
  <c1:C1Knob x:Name="c1kb1" Width="150" Margin="5">
```

これで、C1Knob とページに追加する他のコントロールの間にスペースが追加されます。

6. <c1:C1Knob> タグに **ValueChanged="c1kb1_ValueChanged"** を追加します。次のようになります。

```
XAML
<c1:C1Knob x:Name="c1kb1" Width="150" Margin="5"
ValueChanged="c1kb1_ValueChanged">
```

このイベントハンドラのコードは、この後の手順で追加します。

7. プロジェクトの XAML ウィンドウで、カーソルを **</c1:C1Knob>** タグと **</StackPanel>** タグの間に置きます。
8. ツールボックスに移動し、**[C1RadialGauge]**アイコンをダブルクリックして、**StackPanel** にコントロールを追加します。
9. <c1:C1RadialGauge> タグに **x:Name="c1rg1" Width="150" Margin="5"** を追加して、コントロールをカスタマイズします。次のようになります。

```
XAML
  <c1:C1RadialGauge x:Name="c1rg1" Width="150" Margin="5">

   </c1:C1RadialGauge>
```

これで、**C1RadialGauge** の名前が指定され、コントロールがサイズ変更され、このコントロールと他のコントロールの間にスペースが追加されます。

10. プロジェクトの XAML ウィンドウで、カーソルを最初と2番目の </StackPanel> タグの間に置きます。
11. ツールボックスに移動し、**[TextBox]**アイコンをダブルクリックして、**StackPanel** に標準コントロールを追加します。
12. <TextBox> タグに **x:Name="tb1" Width="300" Margin="5" TextChanged="tb1_TextChanged"** を追加して、コントロールをカスタマイズします。次のようになります。

```
XAML
```

```
<TextBox x:Name="tb1" Width="300" Margin="5" TextChanged="tb1_TextChanged">
</TextBox>
```

これで、**TextBox** の名前が指定され、コントロールがサイズ変更され、このコントロールと他のコントロールの間にスペースが追加されます。イベントハンドラのコードは、この後の手順で追加します。

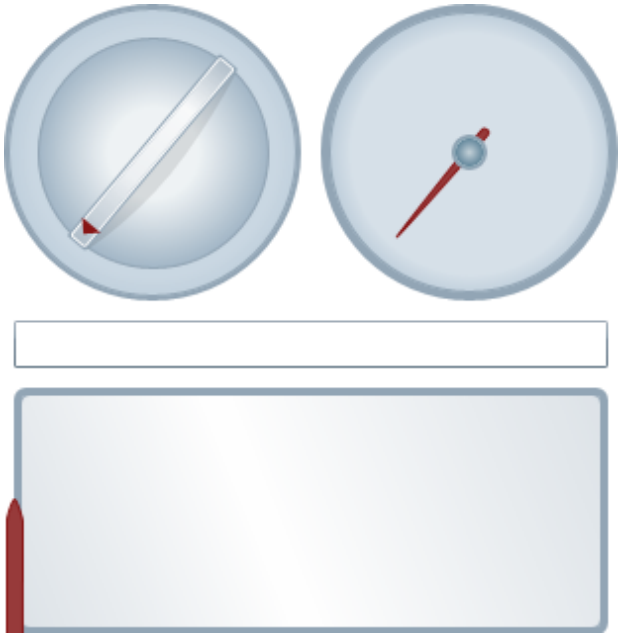
13. プロジェクトの XAML ウィンドウで、カーソルを </TextBox> タグと </StackPanel> タグの間に置きます。
14. ツールボックスに移動し、**[C1LinearGauge]**アイコンをダブルクリックして、**StackPanel** にコントロールを追加します。
15. <c1:C1LinearGauge> タグに **x:Name="c1lg1" Width="300" Height="125" Margin="5"** を追加して、コントロールをカスタマイズします。次のようになります。

XAML

```
<c1:C1LinearGauge x:Name="c1lg1" Width="300" Height="125" Margin="5">
        </c1:C1LinearGauge>
```

これで、**C1LinearGauge** コントロールの名前が指定され、コントロールがサイズ変更され、このコントロールと他のコントロールの間にスペースが追加されます。

16. アプリケーションを実行し、コントロールが次の図のように表示されていることを確認します。



いくつかの **Gauges for Silverlight** コントロールがアプリケーションに追加され、これらのコントロールがカスタマイズされました。これで、アプリケーションのユーザーインターフェイスは正しく設定されました。次の手順では、コードをアプリケーションに追加します。

手順 3:コードの追加

前の手順では、新しい Silverlight プロジェクトを作成し、アプリケーションにいくつかの **Gauges for Silverlight** コントロールを追加しました。この手順では、アプリケーションにコードを追加してカスタマイズします。

次の手順に従います。

1. **[表示]→[コード]**を選択してコードビューに切り替えます。
2. 次の imports 文をページの先頭に追加します。

VisualBasic

```
Imports C1.Silverlight
Imports C1.Silverlight.Gauge
```

C#

```
using C1.Silverlight;
using C1.Silverlight.Gauge;
```

3. **TextBox_TextChanged** イベントハンドラにコードを追加します。次のようになります。

VisualBasic

```
Private Sub tb1_TextChanged(ByVal sender As System.Object,
ByVal e As System.Windows.Controls.TextChangedEventArgs) Handles tb1.TextChanged
    Me.c1lg1.Value = Me.tb1.Text
    Me.c1rg1.Value = Me.tb1.Text
    Me.c1kb1.Value = Me.tb1.Text
End Sub
```

C#

```
private void tb1_TextChanged(object sender, TextChangedEventArgs e)
{
    this.c1lg1.Value = Convert.ToDouble(this.tb1.Text);
    this.c1rg1.Value = Convert.ToDouble(this.tb1.Text);
    this.c1kb1.Value = Convert.ToDouble(this.tb1.Text);
}
```

実行時にテキストボックスに値が入力されると、ゲージコントロールの値はその値に設定されます。

4. **C1Knob_ValueChanged** イベントハンドラに、ゲージとテキストボックスコントロールの値を設定するコードを追加します。次のようになります。

VisualBasic

```
Private Sub c1kb1_ValueChanged(ByVal sender As System.Object,
ByVal e As C1.Silverlight.PropertyChangedEventArgs(Of System.Double)) Handles
c1kb1.ValueChanged
    Me.c1lg1.Value = Me.c1kb1.Value
    Me.c1rg.Value = Me.c1kb1.Value
    Me.tb1.Text = Me.c1kb1.Value.ToString
End Sub
```

C#

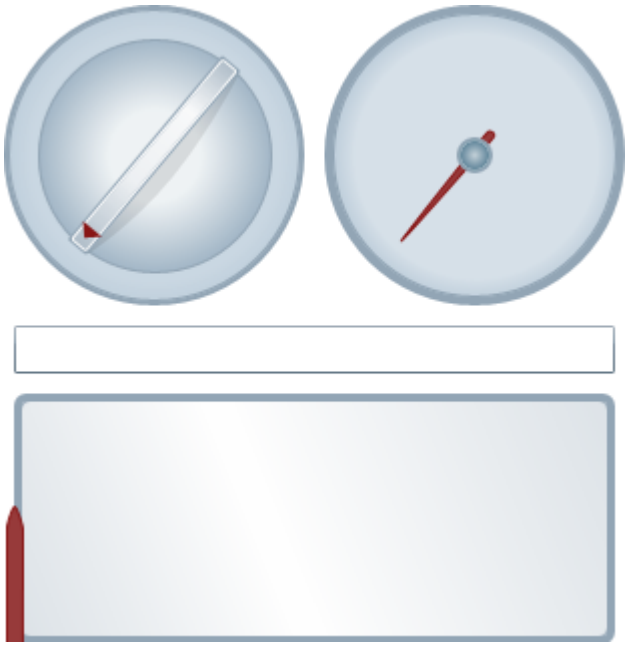
```
private void c1kb_ValueChanged(object sender, PropertyChangedEventArgs e)
{
    this.c1lg.Value = this.c1kb1.Value;
    this.c1rg1.Value = this.c1kb1.Value;
    this.tb1.Text = Convert.ToString(this.c1kb1.Value);
}
```

この手順では、アプリケーションにコードを追加しました。次の手順では、アプリケーションを実行し、実行時の操作を確認します。

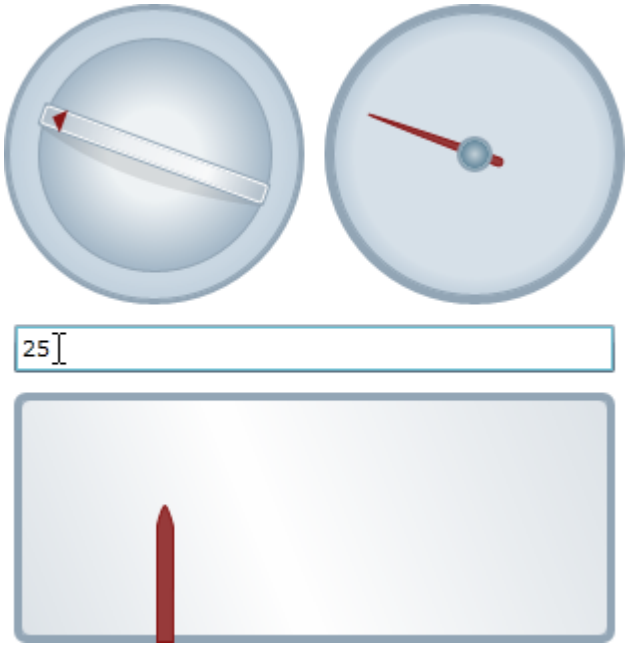
手順 4:アプリケーションの実行

これまでに Silverlight アプリケーションを作成し、外観と動作をカスタマイズしたので、次にアプリケーションを実行します。アプリケーションを実行し、**Gauges for Silverlight** の実行時の動作を確認するには、次の手順に従います。

1. **[プロジェクト]**メニューから**[ソリューションのテスト]**を選択し、実行時にアプリケーションがどのように表示されるかを確認します。アプリケーションは次の図のように表示されます。

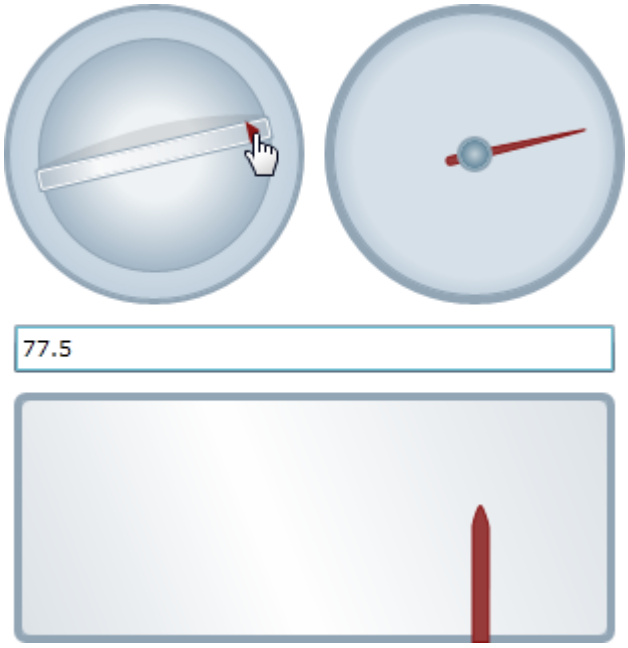


2. テキストボックスに値(**25** など)を入力します。**C1Knob**、**C1RadialGauge**、および **C1LinearGauge** コントロールの値が変化することに注目してください。



デフォルトでは、ゲージコントロールの **Minimum** プロパティは **0** に設定されており、**Maximum** は **100** に設定されています。したがって、Value が **25** に設定されると、ゲージは最初の 1/4 の位置を示します。

3. **C1Knob** コントロールをクリックし、**Value** を変更してみてください。他のゲージの値も同様に変更されて、現在の値がテキストボックスに表示されることがわかります。



おめでとうございます。**Gauges for Silverlight** のクイックスタートはこれで終了です。**C1RadialGauge**、**C1LinearGauge**、および **C1Knob** コントロールを使ってアプリケーションを作成し、アプリケーションの実行時の機能をいくつか確認しました。

Gauges for WPF/Silverlight の使い方

Gauges for WPF/Silverlight には、次のメインコントロールが含まれます。

- **C1RadialGauge**
  回転型のポインタを使用し、曲線目盛りに沿って値を表示します。これは、典型的なスピードメーターに似ています。
- **C1LinearGauge**
  直線型のポインタを使用し、直線目盛りに沿って値を表示します。これは、典型的な温度計に似ています。
- **C1Knob**
  C1RadialGauge を拡張します。ユーザーがポインタを回転して数値を選択できます。音楽プレイヤーのボリュームつまみを真似る場合に最適です。

C1RadialGauge と **C1LinearGauge** は、すべてのゲージに共通する基本機能を備えた一般的な抽象クラス **C1Gauge** から派生されます。**Gauges for WPF/Silverlight** には、次の追加コントロールが含まれます。

- **C1RegionKnob**
  このコントロールは、**C1Knob** コントロールに基づいています。アプリケーションに範囲型のゲージを簡単に追加できます。
- **C1RulerGauge**
  このコントロールは、**C1LinearGauge** コントロールに基づいています。アプリケーションにルーラー型のゲージを簡単に追加できます。
- **C1SpeedometerGauge**
  このコントロールは、**C1RadialGauge** コントロールに基づいています。アプリケーションにスピードメーター型のゲージを簡単に追加できます。
- **C1VolumeGauge**
  このコントロールは、**C1RadialGauge** コントロールに基づいています。アプリケーションにボリューム型のゲージを簡単に追加できます。

これらのコントロールは、**C1RadialGauge**、**C1LinearGauge**、および **C1Knob** コントロールから派生されます。

ゲージコントロールが便利な理由

ゲージは1つの値を示すだけです。ゲージではなく単純なラベルを使用して値を表示することもできるのに、なぜゲージコントロールを使用する必要があるのでしょうか。

ゲージには範囲を表示することもできるため、ユーザーは現在の値が大きいか、小さいか、それとも中間であるかを即座に判断することができます。このため、ゲージの方がわかりやすくて便利です。2つのラベルを追加して範囲と現在値を表示することもできますが、こうすると、わかりにくいユーザーインターフェイスになります。このため、多くのアプリケーションでは、進行状況を表示するために、ラベルではなく、単純な直線型ゲージの進捗状況インジケータを使用しています。

ゲージは、単純なラベル（またはスライダやスクロールバー）より視覚的にわかりやすく、アプリケーションの価値を高めます。

しかし、デザイナに XAML で魅力的なゲージを作成してもらい、要素をアニメーション表示して現在値を示すこともできるのに、ゲージコントロールを使用する理由があるのでしょうか。なぜ、コントロールを使用するのでしょうか。

これには、いくつかの理由があります。まず、誰もが優れたデザイナというわけではなく、また、優れたデザイナに依頼できるとも限りません。次に、アプリケーションに必要なゲージが1つだけではないこともあります。さまざまな範囲に渡る値を表示するために、複数のゲージが必要になる可能性があります。アプリケーションを記述するときには、実際の範囲（現四半期の最大売上値など）すらわかっていない場合もあります。

ゲージコントロールを使用すれば、手作業で XAML のコードを記述しなくても、データに基づいて、プログラムによって柔軟に範囲を調整できます。

C1RadialGauge の使用

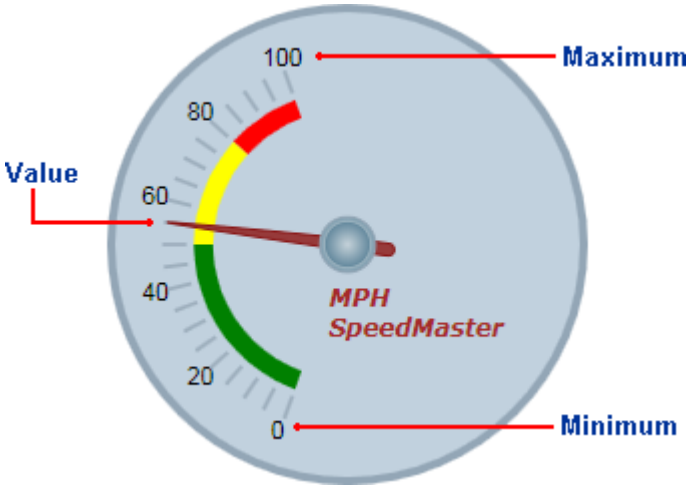
C1RadialGauge は、回転型のポインタを使用して、曲線目盛りに沿って値を表示します。**C1RadialGauge** コントロールは、回転型のポインタを使用して値を表示します。値は Value プロパティで表されます。範囲は **Minimum** プロパティと **Maximum** プロパティで定義されます。**C1RadialGauge** コントロールは、典型的なスピードメーターに似ています。

C1RadialGauge を作成して使用するには、通常、次の手順に従います。

1. **C1RadialGauge** コントロールを作成し、メインプロパティの **Minimum**、**Maximum**、**StartAngle**、および **SweepAngle** を設定します。
2. **C1GaugeMark** および **C1GaugeLabel** デコレータを追加して、スケールを表示します。各要素には、ラベル、目盛りマーク、またはその両方を表示できます。
3. オプションで、**C1GaugeRange** デコレータを追加して、スケールの一部を強調表示します。これらの範囲は、通常、小さすぎる値、適正値、または大きすぎる値を示すために使用されます。**Value** プロパティが変化すると範囲が自動的に移動するように、範囲を動的にすることもできます。
4. オプションで、XAML テンプレートを使用してゲージをカスタマイズします。
5. **Value** プロパティを設定して、初めに表示される値を指定します。

C1RadialGauge の値

C1RadialGauge コントロールの **Minimum**、**Maximum**、および Value プロパティを使用して、有効な範囲およびその範囲内で選択された値を指定できます。



Minimum および **Maximum** プロパティは、ゲージに表示される値の範囲を指定します。たとえば、温度計には -40 ～ 100 度の範囲、スピードメーターには時速0～ 140 マイルの範囲を割り当てることができます。この範囲は、**Minimum** および **Maximum** プロパティ(**double** 型)を使用して指定されます。**C1RadialGauge** コントロールのデフォルトの範囲は0～ 100 です。

Value プロパティは、ゲージの現在の値を指定します。**C1RadialGauge** コントロールでは、**Pointer** 要素がこの値を指し示すことによってこれが視覚的に示されます。**C1RadialGauge** コントロールのデフォルトの **Value** は 50 です。

C1RadialGauge の角度

範囲を定義したら、**Minimum** および **Maximum** 値に対応する角度を指定する必要があります。**StartAngle** は、**Value** プロパティが範囲の **Minimum** 値に設定されているときのポインタの位置を定義します。**SweepAngle** は、**Value** プロパティが範囲の **Maximum** 値に設定されているときの **StartAngle** からの回転角度を指定します。

すべての角度は、コントロールの上中央から時計回りに度単位で指定されます。角度には負の値も指定できますが、**SweepAngle** の絶対値が 360 度を超えることはできません。**StartAngle** のデフォルト値は -140、**SweepAngle** のデフォルト値は 280 です。

次の図は、**StartAngle** プロパティと **SweepAngle** プロパティのさまざまな値による違いを示します。

StartAngle = 0
SweepAngle = 90

StartAngle = 0
SweepAngle = -90

StartAngle = 45
SweepAngle = 270

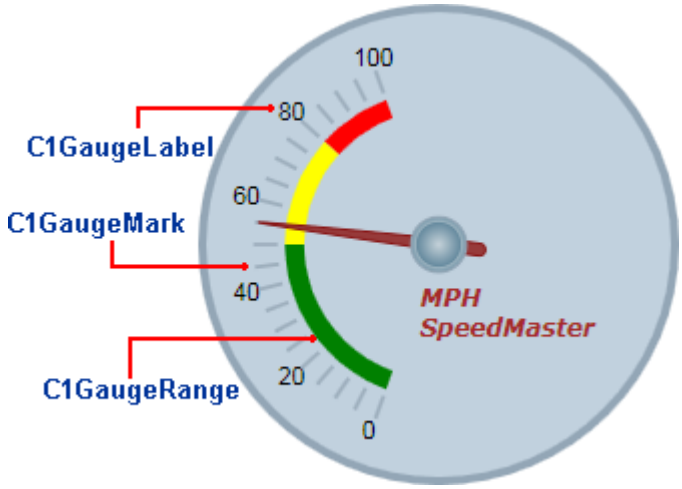
StartAngle = -160
SweepAngle = 180

StartAngle = -160
SweepAngle = -180

StartAngle = -140
SweepAngle = 280

C1RadialGauge デコレータ

C1RadialGauge コントロールには、デフォルトでは青灰色の背景とポインタのみが表示されます。ただし、多くのアプリケーションでは、現在の値がいくつで、その値がゲージの範囲内のどこに位置するかがわかるように、ラベルと目盛りマークがスケールとして表示されます。それには、**C1GaugeMark**、**C1GaugeLabel**、および **C1GaugeRange** 要素をゲージの **Decorators** コレクションに追加します。



これらのデコレータは、**From**、**To**、および **Interval** プロパティの値で決定されるスケールの特定の位置に表示されます。

上の画像では、1つの **C1GaugeMark** 要素と1つの **C1GaugeLabel** 要素が表示されています。

XAML

```
<!-- ラベルマークの追加 -->
<c1:C1GaugeLabel From="0" To="100" Interval="20" Location="1.1"/>
<!-- 目盛りマークの追加 -->
<c1:C1GaugeMark From="0" To="100" Interval="5" Location=".9"/>
```

C1GaugeLabel 要素は、スケールに沿って 0 から 100 までの値のラベルを表示します。**C1GaugeMark** 要素は、5きざみに目盛りマークを表示します。

スケールを表示するほかに、スケール範囲の一部を強調表示することができます。たとえば、赤色のマーカーを追加し、その範囲は値が小さすぎる(売上)ことや、大きすぎる(費用)ことを示すことができます。それには、いくつかの C1GaugeRange 要素をゲージの Decorators コレクションに追加します。

上の画像では、3つの **C1GaugeRange** 要素が表示されています。

XAML

```
<!-- 3つの色付き範囲の追加 -->
<c1:C1GaugeRange From="80" To="100" Location="0.7" Fill="Red" />
<c1:C1GaugeRange From="50" To="80" Location="0.7" Fill="Yellow" />
<c1:C1GaugeRange From="0" To="50" Location="0.7" Fill="Green" />
```

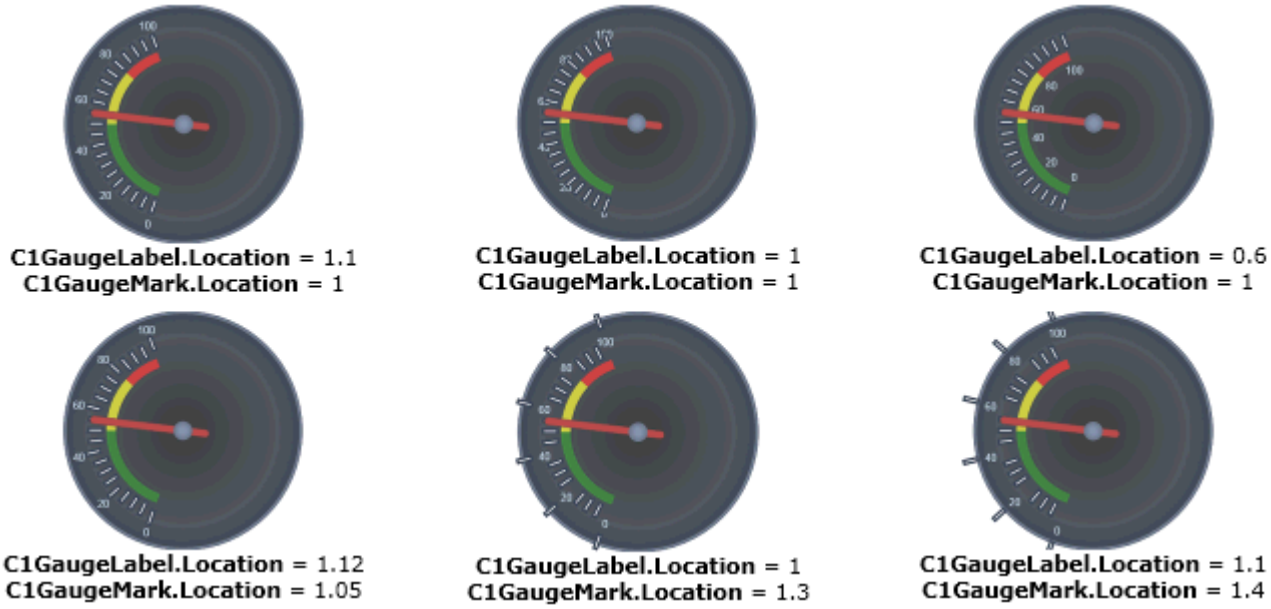
これらの **C1GaugeRange** 要素は、それぞれ赤色、黄色、および緑色の範囲を示します。各 **C1GaugeRange** 要素は、スケールに沿った1つの曲線型の帯として表示されます。帯の色は **Fill** プロパティによって決定され、位置は **From** および **To** プロパティによって決定されます。帯の太さは、**StrokeThickness** プロパティを使用して制御できます。

C1RadialGauge デコレータの位置

各デコレータ要素には、要素が表示される位置を指定する **Location** プロパティがあります。このプロパティには、0(ゲージの中心)から1(ゲージの外縁)までを指定できます。ゲージコントロールには、すべてのデコレータの配置を制御する **Radius** プロパティもあり、これも0から1までを指定できます。半径プロパティのデフォルト値は **0.8** です。この場合は、すべてのデコレータがコントロール内に表示されます。

C1RadialGauge デコレータのサンプルでは、**C1GaugeLabel** の **Location** プロパティが **1.1** に設定されています。これにより、ラベルはゲージの外側にオフセットされて表示されます。**Radius** プロパティは **0.8** に設定されているため、ラベルはコントロール内に描画されます（この場合、ラベルの実際の位置は、1.1 * 0.8 = 0.88 と計算できます）。

次の図は、サンプルゲージの **C1GaugeMark** および **C1GaugeLabel** 要素にさまざまな **Location** プロパティを適用した場合の違いを示します。



Location に1より大きな値を指定すると、ラベルやマークがゲージ本体の外側に描画されます。

C1GaugeMark 要素は、必要な数だけ指定できます。たとえば、0～ 60 分の範囲を持つ時計型のゲージを作成できます。この場合は、1つの **C1GaugeLabel** 要素と2つの **C1GaugeMark** 要素を使用します。

- 4つの「主な」時間（12、3、6、9）のラベルを表示する 15 分間隔の **C1GaugeLabel**。
- 1時間を表す5分間隔の **C1GaugeMark**。
- 1分を表す1分間隔の **C1GaugeMark**。

ラベルと目盛りマークは、**Template** プロパティを使用してカスタマイズすることもできます。

C1RadialGauge デコレータの値の連結

範囲は、静的な値に限りません。**ValueBinding** プロパティを使用して、範囲の開始位置または終了位置をゲージに表示された現在の値に連結できます。たとえば、次のコードは、速度が時速 80 マイルを超えた場合にのみ、赤色の範囲を表示します。

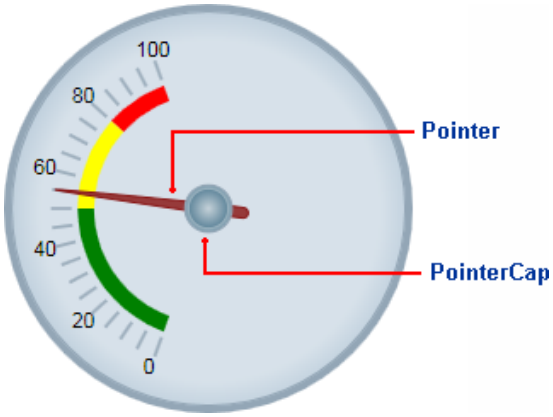
XAML

```
<!-- 3つの色付き範囲の追加 -->
<c1:C1GaugeRange From="80" ValueBinding="To" Location="0.7"Background="Red" />
<c1:C1GaugeRange From="50" To="80" Location="0.7" Fill="Yellow" />
<c1:C1GaugeRange From="0" To="50" Location="0.7"Fill="Green" />
```

次の図は、**Value** プロパティの変化に伴う表示の変化を示します。

C1RadialGauge のポインタ

C1RadialGauge コントロールには、選択されている **Value** をコントロールに示すポインタがあります。ポインタは、実際には **Pointer** 要素と **PointerCap** 要素で構成されます。



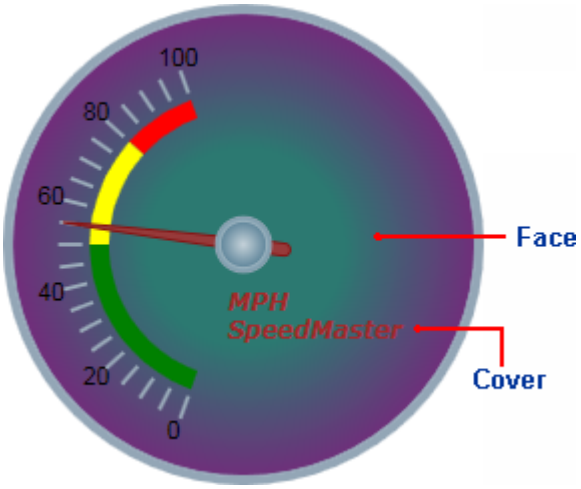
PointerOrigin プロパティは、ポインタの場所を設定します。ポインタの場所は、このプロパティを設定することでカスタマイズできます。たとえば、ポイント(0, 0)は、コントロールの左上隅を示し、ポイント(0.5, 0.5)は、コントロールの中心を示します。**Location** プロパティは、**Pointer** 要素の相対的な場所を設定します。

Pointer 要素は、デフォルトでは茶色の先細の要素として表示されますが、**PointerFill**、**PointerLength**、**PointerOffset**、**PointerStroke**、**PointerStrokeThickness**、**PointerStyle**、**PointerVisibility**、**PointerWidth** などのプロパティを設定することで、**Pointer** 要素の外観をカスタマイズできます。

PointerCap 要素は、デフォルトでは灰色の円として表示されますが、**PointerCap**、**PointerCapFill**、**PointerCapStroke**、**PointerCapStrokeThickness**、**PointerCapStyle** などのプロパティを設定することで、**PointerCap** 要素の外観をカスタマイズできます。**PointerCapSize** テンプレートを編集することもできます。詳細については、「テンプレート」を参照してください。

C1RadialGauge の面とカバー

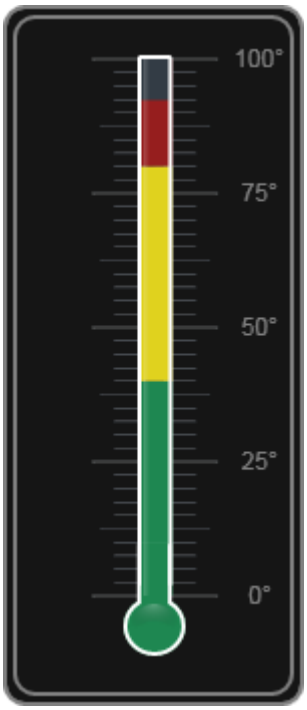
時計と同様に、**C1RadialGauge** コントロールには、**Face** および **Cover** があります。**Face** は背景の上に表示され、ポインタなどのデコレータの下に表示されます。**Cover** は、時計を覆うガラスカバーのように、他のすべての要素の上に表示されます。たとえば、次の図では、**Face** がグラデーション付きでゲージ内の要素の下に表示されています。**Cover** にはテキストが表示され、**Face** と他のすべての要素の上に表示されています。



Cover の外観をカスタマイズするには、**CoverTemplate** を使用します。**Face** の外観をカスタマイズするには、**FaceTemplate** を使用します。詳細については、「テンプレート」を参照してください。

C1LinearGauge の使用

C1LinearGauge コントロールのオブジェクトモデルは、**C1RadialGauge** のオブジェクトモデルとほとんど同じです。**C1LinearGauge** は、直線型のポインタを使用して、値を直線目盛りに沿って表示します。これは、典型的な温度計に似ています。



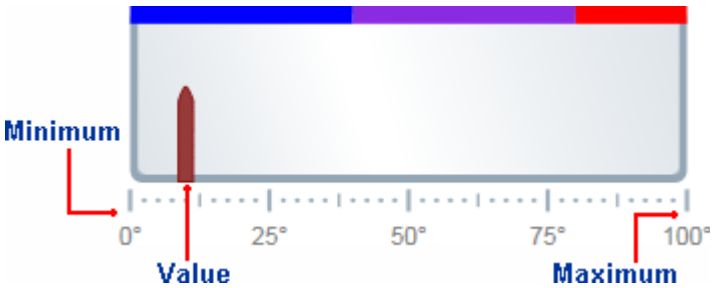
C1LinearGauge コントロールを作成して使用する手順は、**C1RadialGauge** で説明した手順と同じです。

1. **C1LinearGauge** コントロールを作成し、メインプロパティの **Minimum**、**Maximum**、および **Orientation** を設定します。
2. **C1GaugeMark** デコレータを追加して、スケールを表示します。各 **C1GaugeMark** 要素には、ラベル、目盛りマーク、またはその両方を表示できます。
3. オプションで、**C1GaugeRange** デコレータを追加して、スケールの一部を強調表示します。これらの範囲は、通常、小さすぎる値、適正値、または大きすぎる値を示すために使用されます。**Value** プロパティが変化すると範囲が自動的に移動するように、範囲を動的にすることもできます。

4. オプションで、XAML テンプレートを使用してゲージをカスタマイズします。
5. **Value** プロパティを設定して、初めに表示される値を指定します。

C1LinearGauge の値

C1LinearGauge コントロールの **Minimum**、**Maximum**、および **Value** プロパティを使用して、有効な範囲およびその範囲内で選択された値を指定できます。

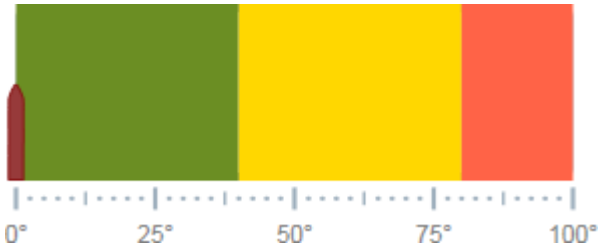


Minimum および **Maximum** プロパティは、ゲージに表示される値の範囲を指定します。たとえば、温度計には -40 ～ 100 度の範囲、スピードメーターには時速0～ 140 マイルの範囲を割り当てることができます。この範囲は、**Minimum** および **Maximum** プロパティ(**double** 型)を使用して指定されます。C1LinearGauge コントロールのデフォルトの範囲は0～ 100 です。

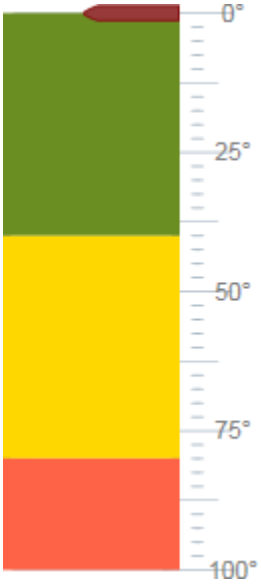
Value プロパティは、ゲージの現在の値を指定します。**C1LinearGauge** コントロールでは、**Pointer** 要素がこの値を指し示すことによってこれが視覚的に示されます。**C1LinearGauge** コントロールのデフォルトの **Value** は0です。上の図では、**Value** は 10 に設定されています。

C1LinearGauge の方向

C1LinearGauge コントロールには、円形ゲージで使用される **StartAngle** および **SweepAngle** プロパティはありません。代わりに、ゲージを縦型または横型として作成するために使用する Orientation プロパティがあります。



デフォルトでは、**Orientation** プロパティは **Horizontal** に設定され、ゲージはアプリケーション内で横向きに表示されます。



Orientation プロパティを **Vertical** に設定することで、縦型のゲージを作成できます。

デフォルトでは、縦の **C1LinearGauge** コントロールは、上から下に向けてスケールを表示します（上の例では0～ 100）。スケールの方向を逆にするには、次のプロパティを設定する必要があります。

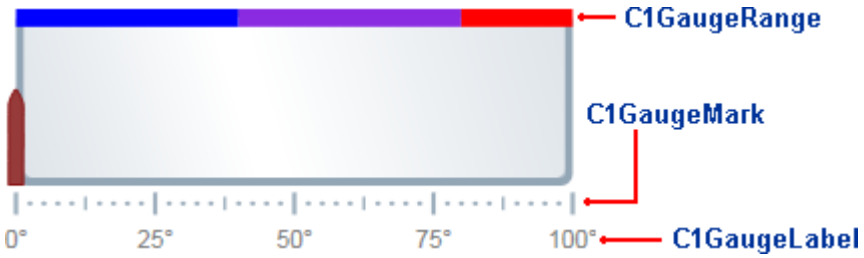
- **XAxisLocation** を **1** に設定します
- **XAxisLength** を **-1** に設定します

スケールが逆になるため、100 がゲージの上端、0が下端になります。

直線型ゲージには、さまざまな応用があります。たとえば、「C1LinearGauge の使い方」に表示されているゲージのように、縦の直線型ゲージを温度計として使用できます。

C1LinearGauge のデコレータ

デフォルトでは、**C1LinearGauge** コントロールには、単純な横の直線型ゲージのみが表示されます。ただし、多くのアプリケーションでは、現在の値がいくつで、その値がゲージの範囲内のどこに位置するかがわかるように、ラベルと目盛りマークがスケールとして表示されます。それには、**C1GaugeMark**、**C1GaugeLabel**、および **C1GaugeRange** 要素をゲージの **Decorators** コレクションに追加します。



これらのデコレータは、**From**、**To**、および **Interval** プロパティの値で決定されるスケールの特定の位置に表示されます。

上の画像では、3つの **C1GaugeMark** 要素と1つの **C1GaugeLabel** 要素が表示されています。

XAML

```
<!--目盛りマークの追加 -->
<c1:C1GaugeMark From="0" To="100" Interval="25" Location="1.1" />
<c1:C1GaugeMark From="0" To="100" Interval="12.5" Location="1.1" />
<c1:C1GaugeMark From="0" To="100" Interval="2.5" Location="1.1" />
<!--ラベルマークの追加 -->
<c1:C1GaugeLabel Location="1.3" Interval="25" Foreground="Gray" Alignment="Center"
```

```
Format="0°" />
```

この **C1GaugeLabel** 要素は、スケールに沿って、0から 100 までの値に対して 25 きざみにラベルを表示します。**C1GaugeMark** 要素は、25、12.5、および 2.5 きざみに目盛りマークを表示します。

スケールを表示するほかに、スケール範囲の一部を強調表示することができます。たとえば、赤色のマーカーを追加し、その範囲は値が小さすぎる(売上)ことや、大きすぎる(費用)ことを示すことができます。それには、いくつかの **C1GaugeRange** 要素をゲージの **Decorators** コレクションに追加します。

上の画像では、3つの **C1GaugeRange** 要素が表示されています。

XAML

```
<!-- 3つの色付き範囲の追加 -->
<c1:C1GaugeRange Fill="Blue" To="40" Width=".1" />
<c1:C1GaugeRange Fill="BlueViolet" From="40" To="80" Width=".1" />
<c1:C1GaugeRange Fill="Red" From="80" To="100" Width=".1" />
```

これらの **C1GaugeRange** 要素は、青色、青紫色、および赤色の範囲を表示します。各 **C1GaugeRange** 要素は、スケールに沿った1つの曲線型の帯として表示されます。帯の色は **Fill** プロパティによって決定され、位置は **From** および **To** プロパティによって決定されます。帯の太さは、**Width** プロパティを使用して制御できます。

C1LinearGauge デコレータの位置

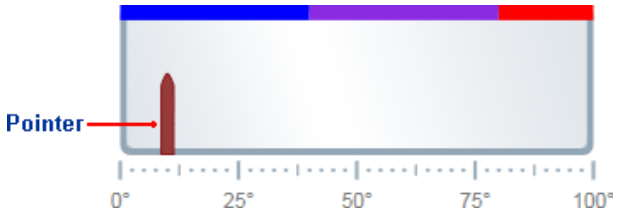
デコレータは、**C1RadialGauge** コントロールと同様に **C1LinearGauge** コントロールに対して配置されます。**C1RadialGauge** コントロールには、デコレータが表示される位置をゲージの中心からの距離で指定する **Radius** プロパティがありました。**Radius** プロパティには、0(ゲージの中心)から1(ゲージの外縁)までの値を指定できます。個々のデコレータは、**Location** プロパティで指定された量だけ **Radius** からオフセットされます。

C1LinearGauge には、Radius に似た **YAxisLocation** プロパティがあります。このプロパティには、0(ゲージの上端)から1(ゲージの下端)までの値を指定できます。個々のデコレータは、**Location** プロパティで指定された量だけ **YAxisLocation** からオフセットされます。

YAxisLocation プロパティのデフォルト値は0です。**C1GaugeMark** デコレータの **Location** のデフォルト値は1です(デフォルトでは、ゲージの下端に要素が表示されます)。**C1GaugeRange** デコレータの **Location** プロパティのデフォルト値は0です(デフォルトでは、ゲージの上端に要素が表示されます)。

C1LinearGauge のポインタ

C1LinearGauge コントロールには、選択されている **Value** をコントロールに示すポインタがあります。ポインタは、**Pointer** 要素で構成されます。

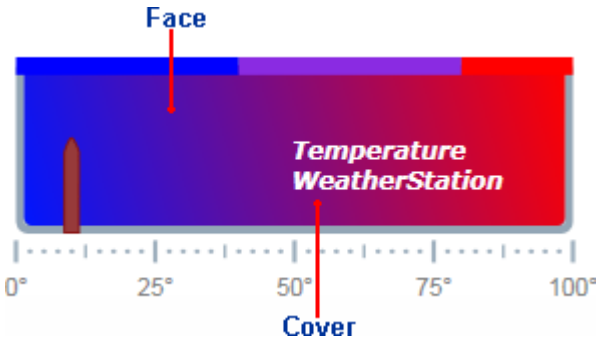


Pointer 要素は、デフォルトでは茶色の先細の要素として表示されますが、**PointerFill**、**PointerLength**、**PointerOffset**、**PointerStroke**、**PointerStrokeThickness**、**PointerStyle**、**PointerVisibility**、**PointerWidth** などのプロパティを設定することで、**Pointer** 要素の外観をカスタマイズできます。**Orientation** プロパティを **Vertical** または **Horizontal** に設定することで、**Pointer** 要素の表示方向をカスタマイズすることもできます。

C1LinearGauge の面とカバー

時計や温度計と同様に、**C1LinearGauge** コントロールには、**Face** および**Cover** があります。**Face** は背景の上に表示され、ポインタなどのデコレータの下に表示されます。**Cover** は、温度計を覆うガラスカバーのように、他のすべての要素の上に表

示されます。たとえば、次の図では、**Face** がグラデーション付きでゲージ内の要素の下に表示されています。**Cover** にはテキストが表示され、**Face** と他のすべての要素の上に表示されています。



Cover の外観をカスタマイズするには、**CoverTemplate** を使用します。**Face** の外観をカスタマイズするには、**FaceTemplate** を使用します。詳細については、「テンプレート」を参照してください。

C1Knob の使用

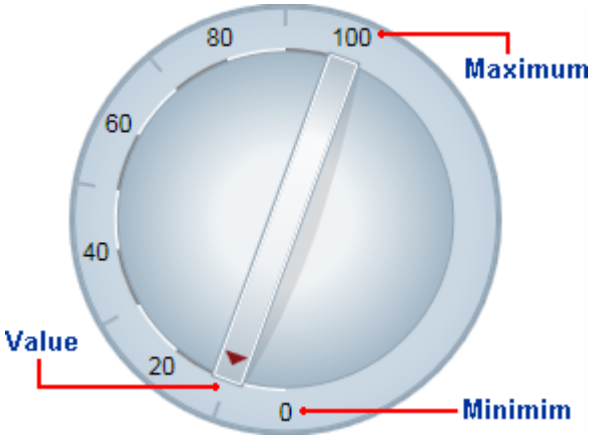
C1Knob コントロールは、**C1RadialGauge** コントロールを拡張して、ユーザーがポインタを回転して数値を選択できるようにします。たとえば、**C1Knob** は、音楽プレイヤーのボリュームつまみを真似る場合に最適です。デフォルトでは、**C1Knob** コントロールは次の図のように表示されます。

C1Knob コントロールを作成して使用する方法は、**C1RadialGauge** コントロールの作成方法に似ており、通常は同様の手順に従います。

1. **C1Knob** コントロールを作成し、メインプロパティの **Minimum**、**Maximum**、**StartAngle**、および **SweepAngle** を設定します。
2. **InteractionMode** プロパティを設定することで、ユーザーがノブを操作する方法を指定します。
3. **C1GaugeMark** および **C1GaugeLabel** デコレータを追加して、スケールを表示します。各要素には、ラベル、目盛りマーク、またはその両方を表示できます。
4. オプションで、XAML テンプレートを使用してゲージをカスタマイズします。
5. **Value** プロパティを設定して、初めに表示される値を指定します。

C1Knob の値

C1Knob コントロールの **Minimum**、**Maximum**、および **Value** プロパティを使用して、有効な範囲およびその範囲内で選択された値を指定できます。



Minimum および **Maximum** プロパティは、ノブに表示される値の範囲を指定します。この範囲は、**Minimum** および **Maximum** プロパティ(**double** 型)を使って指定されます。**C1Knob** コントロールのデフォルトの範囲は0～ 100 です。

Value プロパティは、ゲージの現在の値を指定します。**C1Knob** コントロールでは、**Pointer** 要素がこの値を指し示すことによってこれが視覚的に示されます。**C1Knob** コントロールのデフォルトの **Value** は 50 です。

C1Knob の角度

範囲を定義したら、**Minimum** および **Maximum** 値に対応する角度を指定できます。**StartAngle** は、Value プロパティが範囲の **Minimum** 値に設定されているときのポインタの位置を定義します。**SweepAngle** は、**Value** プロパティが範囲の **Maximum** 値に設定されているときの **StartAngle** からの回転角度を指定します。

すべての角度は、コントロールの上中央から時計回りに度単位で指定されます。角度には負の値も指定できますが、**SweepAngle** の絶対値が 360 度を超えることはできません。

C1Knob の操作

InteractionMode プロパティは、実行時にコントロールで使用できる操作を制御します。つまり、ユーザーがノブを動かすために、クリック、ドラッグ、またはその両方を使用できるかどうかを選択できます。

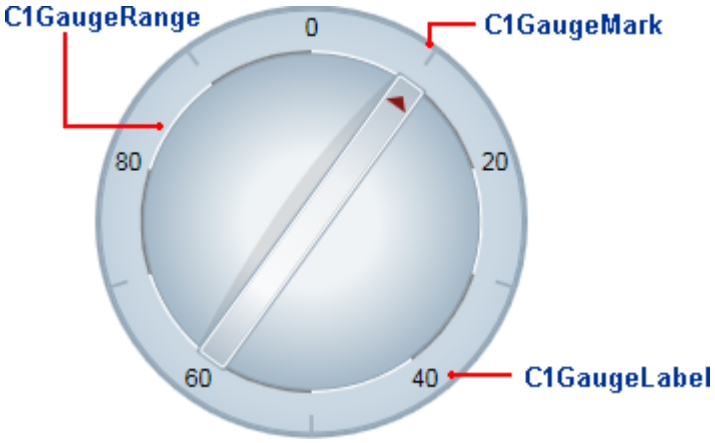
InteractionMode プロパティには、次の**KnobInteractionMode** 値の1つを設定できます。

値	説明
Drag	ユーザーがドラッグすることでポインタが移動します。
Click	ユーザーがノブ内をクリックすることでポインタが移動します。
ClickOrDrag	ユーザーがノブ内をクリックするか、ポインタをドラッグすることでポインタが移動します。

デフォルトでは、**InteractionMode** プロパティは **Click** に設定されています。

C1Knob のデコレータ

C1Knob コントロールには、デフォルトでは青灰色の背景とポインタのみが表示されます。**C1RadialGauge** や **C1LinearGauge** と同様に、**C1GaugeMark**、**C1GaugeLabel**、および**C1GaugeRange** 要素をゲージの **Decorators** コレクションに追加することで、ノブのインジケータをカスタマイズできます。



これらのデコレータは、**From**、**To**、および **Interval** プロパティの値で決定されるスケールの特定の位置に表示されます。

上の画像では、1つの **C1GaugeMark** 要素と1つの**C1GaugeLabel** 要素が表示されています。

XAML

```
<!-- 目盛りマークの追加 -->
<c1:C1GaugeMark Interval="20" Alignment="In" From="10" />
<!-- ラベルマークの追加 -->
<c1:C1GaugeLabel Interval="20" Alignment="Center" Location="0.9" To="80" />
```

C1GaugeLabel 要素は、スケールに沿って 10 から 90 までの値のラベルを表示します。**C1GaugeMark** 要素は、20 きざみに目盛りマークを表示します。

スケールを表示するほかに、いくつかの **C1GaugeRange** 要素をゲージの **Decorators** コレクションに追加することで、スケールの一部を強調表示できます。

上の画像では、10 個の **C1GaugeRange** 要素が表示されています。

XAML

```
<!-- 10 個の色付き範囲の追加 -->
<c1:C1GaugeRange From="0" To="10" Location="0.7" Fill="White" />
<c1:C1GaugeRange From="10" To="20" Location="0.7" Fill="Gray" />
<c1:C1GaugeRange From="20" To="30" Location="0.7" Fill="White" />
<c1:C1GaugeRange From="30" To="40" Location="0.7" Fill="Gray" />
<c1:C1GaugeRange From="40" To="50" Location="0.7" Fill="White" />
<c1:C1GaugeRange From="50" To="60" Location="0.7" Fill="Gray" />
<c1:C1GaugeRange From="60" To="70" Location="0.7" Fill="White" />
<c1:C1GaugeRange From="70" To="80" Location="0.7" Fill="Gray" />
<c1:C1GaugeRange From="80" To="90" Location="0.7" Fill="White" />
<c1:C1GaugeRange From="90" To="100" Location="0.7" Fill="Gray" />
```

これらの **C1GaugeRange** 要素は、白色および灰色の範囲を示します。各 **C1GaugeRange** 要素は、スケールに沿った1つの曲線型の帯として表示されます。帯の色は **Fill** プロパティによって決定され、位置は**From** および **To** プロパティによって決定されます。帯の太さは、**StrokeThickness** プロパティを使用して制御できます。

C1Knob デコレータの位置

各デコレータ要素には、要素が表示される位置を指定する**Location** プロパティがあります。このプロパティには、0（ゲージの中心）から1（ゲージの外縁）までを指定できます。ゲージコントロールには、すべてのデコレータの配置を制御する **Radius** プロパティもあり、これも0から1までを指定できます。半径プロパティのデフォルト値は **0.8** です。この場合は、すべてのデコレータがコントロール内に表示されます。

たとえば、**C1GaugeLabel** の **Location** プロパティを **1.1** に設定すると、ラベルはノブの外側にオフセットされて表示されます。**Radius** プロパティは **0.8** に設定されているため、ラベルはコントロール内に描画されます（この場合、ラベルの実際の位置は、1.1 * 0.8 = 0.88 と計算できます）。

レイアウトおよび外観

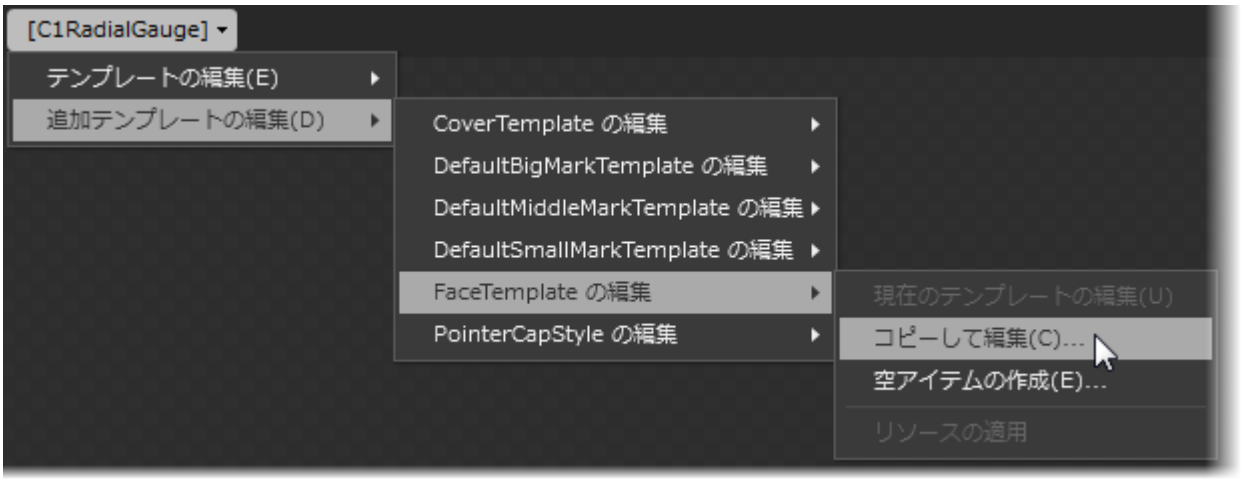
以下のトピックでは、ゲージコントロールのレイアウトと外観をカスタマイズする方法について詳しく説明します。組み込みのレイアウトオプションを使用して、グリッドやキャンバスなどのコントロールをパネル内でレイアウトできます。テーマを使用することで、グリッドの外観をカスタマイズしたり、WPF/Silverlight の XAML ベースのスタイル設定を活用することができます。また、テンプレートを使用して、グリッドを書式設定およびレイアウトしたり、グリッドの操作をカスタマイズすることもできます。

テンプレート

WPF/Silverlight コントロールを使用する主な利点の1つは、これが自由にカスタマイズできるユーザーインターフェイスを持つ「外観のない」コントロールであることです。WPF/Silverlight アプリケーションのユーザーインターフェイスであるルックアンドフィールを独自に設計するのと同様に、**Gauge for WPF/Silverlight** で管理されるデータに関して独自の UI を提供できます。Extensible Application Markup Language（XAML。「ザムル」と発音する）は、コードを記述することなく独自の UI を設計するための簡単な方法を提供する XML ベースの宣言型言語です。

テンプレートへのアクセス

テンプレートにアクセスするには、Microsoft Expression Blend で、C1Gauge コントロールを選択し、メニューから**[追加テンプレートの編集]**を選択します。**[コピーして編集]**を選択して現在のテンプレートのコピーを作成して編集するか、**[空アイテムの作成]**を選択して新しい空のテンプレートを作成します。



メニューを使用して新しいテンプレートを作成する場合、テンプレートはそのテンプレートのプロパティに自動的にリンクされます。手作業でテンプレートの XAML を作成する場合は、作成したテンプレートに適切な Template プロパティをリンクする必要があります。

付属のテンプレート

Gauges for WPF/Silverlight には、次のテンプレートが付属しています。

テンプレート	説明
PointerCapStyle	**PointerCap** 要素に連結されるスタイル。この要素は、**C1LinearGauge** コントロールでは使用できません。
CoverTemplate	C1Gauge の上に配置される要素の生成に使用される DataTemplate。DataTemplate のロード後に、**Top** プロパティから FrameworkElement にアクセスできます。
DefaultBigMarkTemplate	大きいマーク用のデフォルトの DataTemplate。
DefaultMiddleMarkTemplate	普通のマーク用のデフォルトの DataTemplate。

DefaultSmallMarkTemplate	小さいマーク用のデフォルトの DataTemplate。
FaceTemplate	C1Gauge の背景の上に配置される要素の生成に使用される DataTemplate。DataTemplate のロード後に、**Bottom** プロパティから FrameworkElement にアクセスできます。

Template プロパティを使用してテンプレートをカスタマイズできます。

XAML 要素

Gauges for WPF/Silverlight をインストールすると、いくつかの補助 XAML 要素が同時にインストールされます。これらの要素にはテンプレートやテーマが含まれており、**Gauges for WPF/Silverlight** インストールディレクトリに格納されています。これらの要素をプロジェクトに組み込んで、たとえば、デフォルトのテーマに基づく独自のテーマを作成できます。

含まれる補助 XAML 要素

Gauges for WPF には、次の補助 XAML 要素が含まれています。

要素	フォルダ	説明
generic.xaml	XAML	コントロールのさまざまなスタイルと初期スタイルのテンプレートを指定します。

ComponentOne ClearStyle 技術

ComponentOne ClearStyle は、WPF/Silverlight コントロールのスタイル設定をすばやく簡単に実行できる新技術です。ClearStyle を使用すると、面倒な XAML テンプレートやスタイルリソースを操作しなくても、コントロールのカスタムスタイルを作成できます。

現在のところ、すべての標準 WPF/Silverlight コントロールにテーマを追加するには、スタイルリソーステンプレートを作成する必要があります。Microsoft Visual Studio ではこの処理は困難であるため、Microsoft は、このタスクを簡単に実行できるように Expression Blend を導入しました。ただし、Blend に不慣れであったり、十分な学習時間を取れない開発者にとっては、この2つの環境を行き来することはかなり困難な作業です。デザイナに作業を任せることも考えられますが、デザイナと開発者が XAML ファイルを共有すると、かえって煩雑になる可能性があります。

このような場合に、ClearStyle を使用します。ClearStyle は、Visual Studio を使用して直感的な方法でスタイル設定を実行できるようにします。ほとんどの場合は、アプリケーション内のコントロールに対して単純なスタイル変更を行うだけなので、この処理は簡単に行えるべきです。たとえば、データグリッドの行の色を変更するだけであれば、1つのプロパティを設定するだけで簡単に行えるようにする必要があります。一部の色を変更するためだけに、完全で複雑なテンプレートを作成する必要はありません。

ClearStyle の仕組み

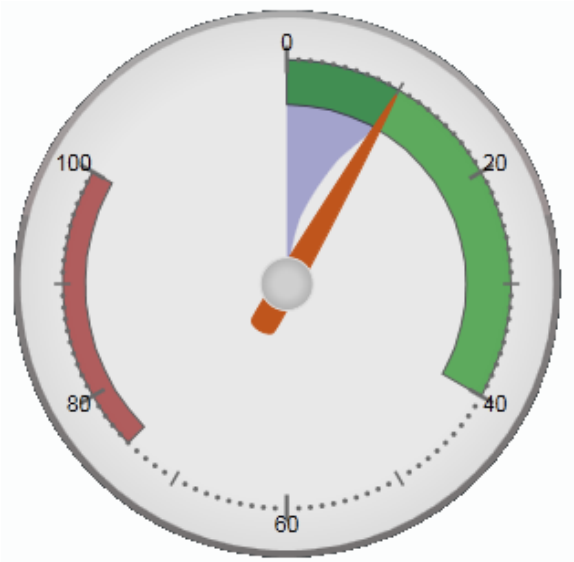
コントロールのスタイルの主な要素は、それぞれ単純な色プロパティとして表されます。これが集まって、コントロール固有のスタイルプロパティセットを形成します。たとえば、**Gauge** には **PointerFill** プロパティや **PointerStroke** プロパティがあり、**DataGrid** の行には **SelectedBrush** や **MouseOverBrush** があります。

たとえば、フォーム上に ClearStyle をサポートしていないコントロールがあるとします。その場合は、ClearStyle によって作成された XAML リソースを使用して、フォーム上の他のコントロールを調整して合わせることができます（正確な色合わせなど）。また、スタイルセットの一部を ClearStyle（カスタムスクロールバーなど）で上書きしたいとします。ClearStyle は拡張可能なのでこれも可能です。必要な場所でスタイルを上書きできます。

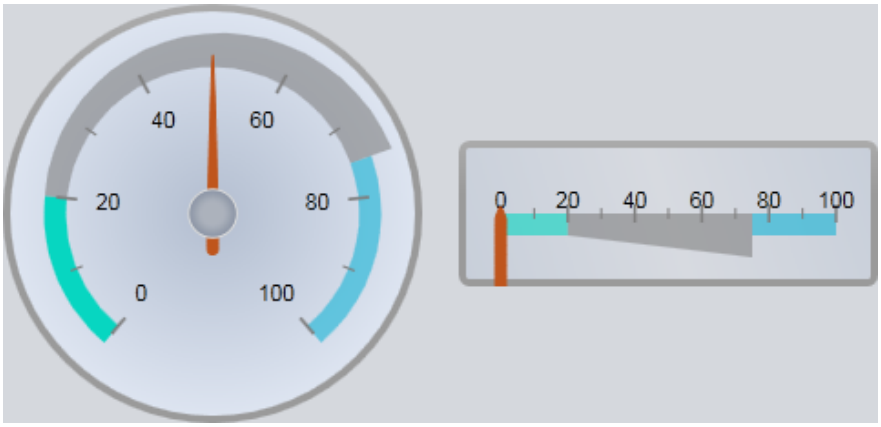
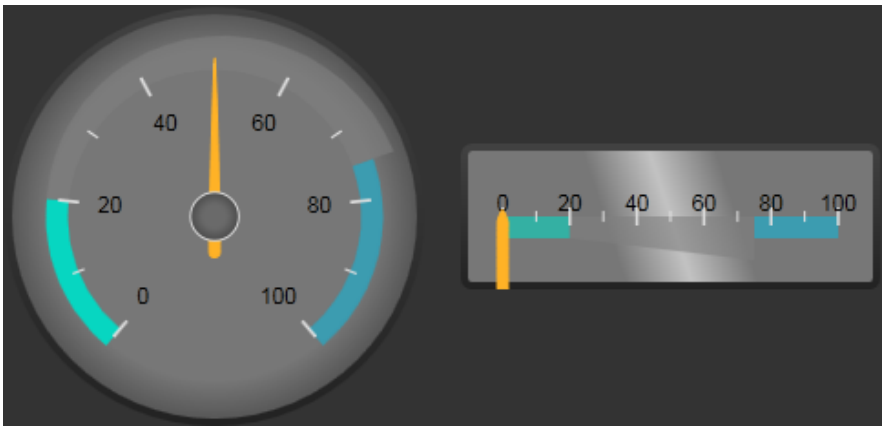
ClearStyle は、すばやく簡単にスタイルを変更することを意図したソリューションですが、ComponentOne のコントロールには引き続き従来の方法を使用して、必要なスタイルを細かく指定して作成できます。完全なカスタム設計が必要になる特別な状況で ClearStyle が邪魔になることはありません。

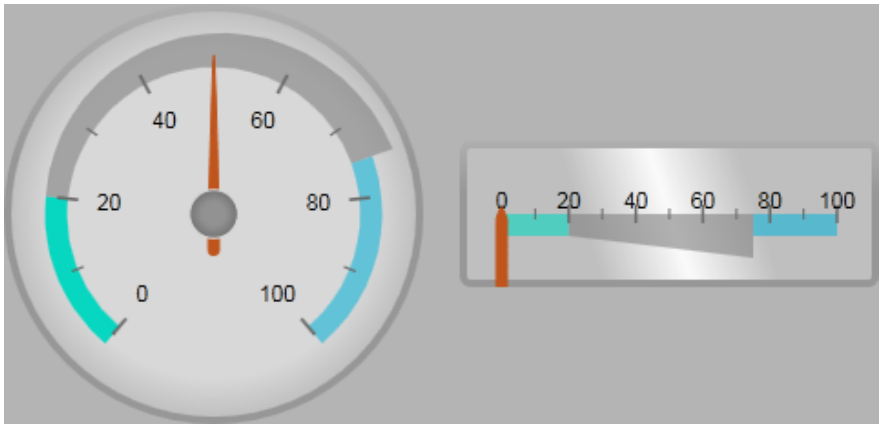
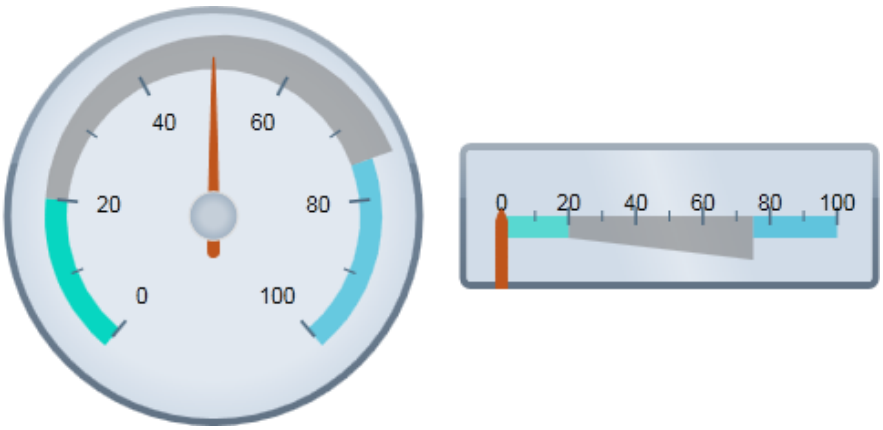
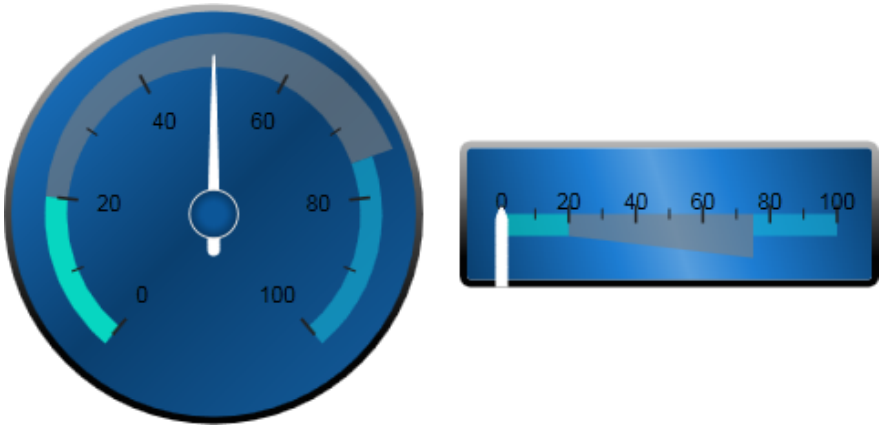
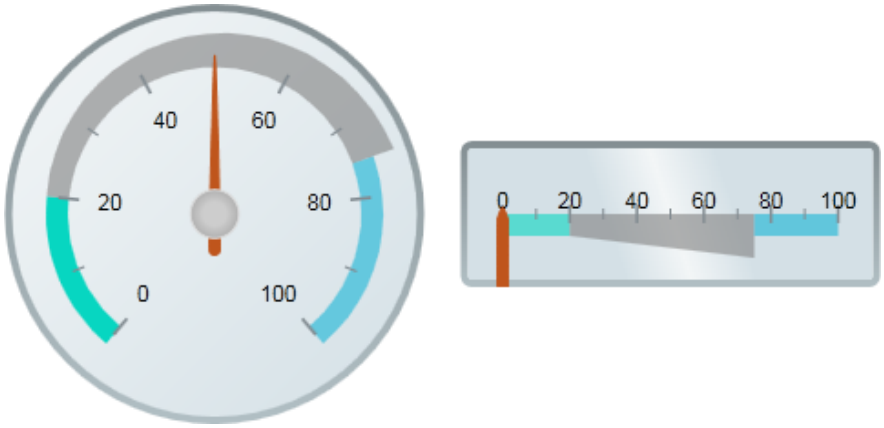
利用可能なテーマ

Gauges for WPF/Silverlight には、グリッドの外観をカスタマイズできるいくつかのテーマが組み込まれています。デフォルトのテーマは、次の図のように表示されます。



これは、このコントロールのデフォルトの外観です。この外観は、組み込みテーマの1つを使用したり、独自のカスタムテーマを作成することで変更できます。すべての組み込みテーマは、WPF Toolkit テーマに基づいています。以下に、組み込みテーマの説明と図を示します。以下の図では、選択状態のスタイルを示すために1つの行が選択されています。

テーマ名	テーマのプレビュー
C1ThemeBureauBlack	
C1ThemeExpressionDark	

C1ThemeExpressionLight	
C1Blue	
C1ThemeShinyBlue	
C1ThemeWhistlerBlue	

要素のテーマを設定するには、**ApplyTheme** メソッドを使用します。最初に、テーマアセンブリへの参照をプロジェクトに追加し、次のようにコードでテーマを設定します。

VisualBasic

```
Private Sub Window_Loaded(sender As System.Object, e As System.Windows.RoutedEventArgs)
Handles MyBase.Loaded
   Dim theme As New C1ThemeExpressionDark
   ' ApplyTheme の使用
   C1Theme.ApplyTheme(LayoutRoot, theme)
```

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
   {
      C1ThemeExpressionDark theme = new C1ThemeExpressionDark();
      //ApplyTheme の使用
      C1Theme.ApplyTheme(LayoutRoot, theme);
   }
```

アプリケーション全体にテーマを適用するには、**System.Windows.ResourceDictionary.MergedDictionaries** プロパティを使用します。最初に、テーマアセンブリへの参照をプロジェクトに追加し、次のようにコードでテーマを設定します。

VisualBasic

```
Private Sub Window_Loaded(sender As System.Object, e As System.Windows.RoutedEventArgs)
Handles MyBase.Loaded
   Dim theme As New C1ThemeExpressionDark
   ' MergedDictionaries の使用
Application.Current.Resources.MergedDictionaries.Add(C1Theme.GetCurrentThemeResources(theme))
End Sub
```

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
   {
      C1ThemeExpressionDark theme = new C1ThemeExpressionDark();
      //MergedDictionaries の使用
Application.Current.Resources.MergedDictionaries.Add(C1Theme.GetCurrentThemeResources(theme));
   }
```

この方法は、初めてテーマを適用する場合にのみ使用できることに注意してください。別の ComponentOne テーマに切り替える場合は、最初に、**Application.Current.Resources.MergedDictionaries** から前のテーマを削除します。

タスク別ヘルプ

タスク別ヘルプは、ユーザーの皆様が Visual Studio のプログラミングに精通しており、ゲージコントロールを使用する一般的な方法を理解していることを前提としています。**Gauges for WPF/Silverlight** 製品を初めて使用される場合は、まず「クイックスタート」を参照してください。

このセクションの各トピックは、**Gauges for WPF/Silverlight** 製品を使用して特定のタスクを行う方法を提供します。

また、タスク別ヘルプトピックは、新しい WPF プロジェクトが作成されており、そのプロジェクトにゲージコントロールが追加されていることを前提としています。

開始値を設定する

このトピックでは、**C1LinearGauge** コントロールの **Value** プロパティを変更します。**Value** プロパティは、現在選択されている値を決定します。デフォルトでは、**C1LinearGauge** コントロールは、最初に **Value** が**0**に設定されますが、設計時、XAML、またはコードでこの値をカスタマイズできます。このトピックでは、**C1LinearGauge** コントロールの **Value** を設定しますが、同じ手順を使用して、他のコントロールの **Value** をカスタマイズすることもできます。

設計時

実行時に **C1LinearGauge** コントロールの **Value** プロパティを設定するには、次の手順に従います。

1. **C1LinearGauge** コントロールをクリックして選択します。
2. [プロパティ]ウィンドウに移動し、**Value** プロパティの横にあるテキストボックスに値（たとえば、20）を入力します。

これで、**Value** プロパティは指定された値に設定されます。

XAML の場合

たとえば、**Value** プロパティを設定するには、**Value="20"** を <c1:C1LinearGauge> タグに追加します。次のようになります。

XAML

```
<c1:C1LinearGauge Height="89" Margin="47,57,33,43" Name="C1LinearGauge1" Width="287"
Value="20">
```

コードの場合

たとえば、Value プロパティを設定するには、プロジェクトに次のコードを追加します。

VisualBasic

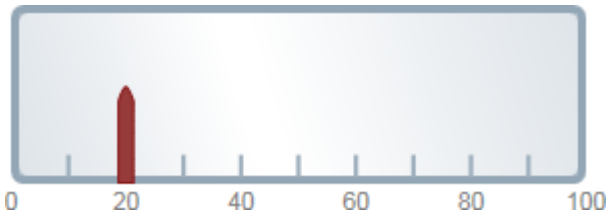
```
C1LinearGauge1.Value = 20
```

C#

```
c1LinearGauge1.Value = 20;
```

プロジェクトの実行と確認

最初に、ゲージの **Pointer** は、指定された **Value** に設定されます。



最小値および最大値を設定する

Minimum および **Maximum** プロパティを使用して、ゲージの数値範囲を設定できます。**Minimum** および **Maximum** 値は、設計時に、XAML、またはコードでカスタマイズできます。このトピックでは、**C1LinearGauge** コントロールの **Minimum** および **Maximum** プロパティを設定しますが、同じ手順を使用して、他のコントロールの **Minimum** および **Maximum** をカスタマイズすることもできます。

> **Minimum** および **Maximum** プロパティを設定する場合は、**Minimum** を **Maximum** より小さくする必要があります。また、Value プロパティは、**Minimum** から **Maximum** までの範囲内の値に設定してください（デフォルトは0です。これは、下で設定する範囲に入っています）。

設計時

実行時に **C1LinearGauge** の **Minimum** および **Maximum** を設定するには、次の手順に従います。

1. **C1LinearGauge** コントロールをクリックして選択します。
2. [プロパティ]ウィンドウに移動し、**Maximum** プロパティの横に値（たとえば、**50**）を入力します。
3. [プロパティ]ウィンドウで、**Minimum** の横に値（たとえば、**-50**）を入力します。

これで、**Minimum** と **Maximum** の値が設定されます。

XAML

```
<c1:C1LinearGauge Height="89" Margin="47,57,33,43" Name="C1LinearGauge1" Width="287"
Maximum="50" Minimum="-50">
```

コードの場合

C1LinearGauge コントロールの **Minimum** および **Maximum** を設定するには、プロジェクトに次のコードを追加します。

VisualBasic

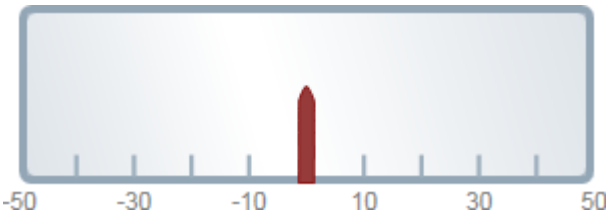
```
C1LinearGauge1.Minimum = -50
C1LinearGauge1.Maximum = 50
```

C#

```
c1LinearGauge1.Minimum = -50;
c1LinearGauge1.Maximum = 50;
```

プロジェクトの実行と確認

実行時に、選択された範囲にゲージが制限されます。



ラベルを追加する

[プロパティ]ウィンドウ、XAML、またはコードで、**C1RadialGauge** コントロールのラベルを追加およびカスタマイズできます。このトピックでは、**C1RadialGauge** コントロールの **C1GaugeLabel** プロパティを設定しますが、同じ手順を使用して、他のコントロールの **C1GaugeLabel** をカスタマイズすることもできます。

設計時

設計時に[プロパティ]ウィンドウで **C1RadialGauge** コントロールにラベルを追加するには、次の手順に従います。

1. **C1RadialGauge** コントロールをクリックして選択します。
2. **[プロパティ]**ウィンドウに移動し、**[デコレータ]**項目の横にある**省略符**ボタンをクリックします。**[デコレータ]**コレクションエディタが開きます。
3. エディタの左上にあるドロップダウンリストで **C1GaugeLabel** を選択し、**[追加]**ボタンをクリックします。**C1GaugeLabel** デコレータがコレクションに追加されて選択されます。
4. 右側の[プロパティ]ペインで、**C1GaugeLabel** 要素の **Location** を **1** に設定します。
5. **C1GaugeLabel** 要素の **Interval** を **20** に設定します。
これで、コントロールのラベルが設定されます。

XAML の場合

XAML で、**C1RadialGauge** コントロールにラベルを追加するには、<c1:C1GaugeLabel> タグを <c1:C1RadialGauge> タグに追加します。次のようになります。

XAML

```
<c1:C1RadialGauge Height="189" Margin="42,29,188,31" Name="C1RadialGauge1"
Width="189">
    <c1:C1GaugeLabel Interval="20" Location="1" />
</c1:C1RadialGauge>
```

コードの場合

ウィンドウを右クリックし、**[コードの表示]**を選択してコードエディタを開きます。コードをメインクラス **Window1_Loaded** イベントハンドラに追加します。次のようになります。

VisualBasic

```
Window1_Loaded(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs) Handles MyBase.LoadedPublic Sub New()
InitializeComponent()
    Dim c1gl As New C1.WPF.Gauge.C1GaugeLabel
```

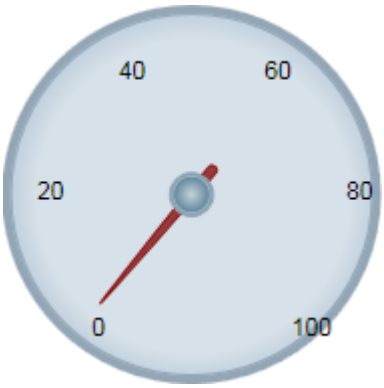
```
    c1gl.Location = 1
    c1gl.Interval = 20
    Me.C1RadialGauge1.Decorators.Add(c1gl)
End Sub
```

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)public MainPage(){
InitializeComponent();
{     C1.WPF.Gauge.C1GaugeLabel c1gl = new C1.WPF.Gauge.C1GaugeLabel();
    c1gl.Location = 1;
    c1gl.Interval = 20;
    this.c1RadialGauge1.Decorators.Add(c1gl);
}
```

プロジェクトの実行と確認

C1RadialGauge コントロールにラベルが表示されます。



目盛りマークを追加する

[プロパティ]ウィンドウ、XAML、またはコードで、**C1LinearGauge** コントロールに目盛りマークを追加できます。このトピックでは、**C1LinearGauge** コントロールの **C1GaugeMark** プロパティを設定しますが、同じ手順を使用して、他のコントロールの **C1GaugeMark** をカスタマイズすることもできます。

設計時

設計時に[プロパティ]ウィンドウで **C1LinearGauge** コントロールに目盛りマークを追加するには、次の手順に従います。

1. **C1LinearGauge** コントロールをクリックして選択します。
2. **[プロパティ]**ウィンドウに移動し、**[デコレータ]**項目の横にある**省略符**ボタンをクリックします。**[デコレータ]**コレクションエディタが開きます。
3. エディタの左上にあるドロップダウンリストで **C1GaugeMark** を選択し、**[追加]**ボタンをクリックします。**C1GaugeMark** デコレータがコレクションに追加されて選択されます。
4. 右側のプロパティペインで、**C1GaugeMark** 要素の **Location** を **1.1** に設定します。
5. **C1GaugeLabel** 要素の **Interval** を **20** に設定します。
6. エディタの左上にあるドロップダウンリストで **C1GaugeMark** を選択し、**[追加]**ボタンをクリックします。second

C1GaugeMark デコレータがコレクションに追加されて選択されます。

7. 右側のプロパティペインで、**C1GaugeMark** 要素の **Location** を **1.1** に設定します。
8. **C1GaugeLabel** 要素の **Interval** を **10** に設定します。
9. エディタの左上にあるドロップダウンリストで **C1GaugeMark** を選択し、**[追加]**ボタンをクリックします。3番目の **C1GaugeMark** デコレータがコレクションに追加されて選択されます。
10. 右側のプロパティペインで、**C1GaugeMark** 要素の **Location** を **1.1** に設定します。
11. **C1GaugeLabel** 要素の **Interval** を **5** に設定します。

XAML の場合

XAML で、**C1LinearGauge** コントロールにラベルを追加するには、3つの <c1:C1GaugeMark> タグを <c1:C1LinearGauge> タグに追加します。次のようになります。

XAML

```
<c1:C1LinearGauge Height="89" Margin="90,72,41,88" Name="C1LinearGauge1" Width="287">
    <c1:C1GaugeMark Interval="20" Location="1.1" />
    <c1:C1GaugeMark Interval="10" Location="1.1" />
    <c1:C1GaugeMark Interval="5" Location="1.1" />
</c1:C1LinearGauge>
```

コードの場合

ウィンドウを右クリックし、**[コードの表示]**を選択してコードエディタを開きます。次に、コードをメインクラス **Window1_Loaded** イベントハンドラに追加します。次のようになります。

VisualBasic

```
Window1_Loaded(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs) Handles MyBase.LoadedPublic Sub New()
InitializeComponent()
    Dim c1gm1 As New C1.WPF.Gauge.C1GaugeMark
    c1gm1.Location = 1.1
    c1gm1.Interval = 20
    Me.C1LinearGauge1.Decorators.Add(c1gm1)
    Dim c1gm2 As New C1.WPF.Gauge.C1GaugeMark
    c1gm2.Location = 1.1
    c1gm2.Interval = 10
    Me.C1LinearGauge1.Decorators.Add(c1gm2)
    Dim c1gm3 As New C1.WPF.Gauge.C1GaugeMark
    c1gm3.Location = 1.1
    c1gm3.Interval = 5
    Me.C1LinearGauge1.Decorators.Add(c1gm3)
End Sub
```

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)public MainPage(){
```

```
InitializeComponent();
{
    C1.WPF.Gauge.C1GaugeLabel c1gm1 = new C1.WPF.Gauge.C1GaugeMark();
    c1gm1.Location = 1.1;
    c1gm1.Interval = 20;
    this.c1LinearGauge1.Decorators.Add(c1gm1);
    C1.WPF.Gauge.C1GaugeLabel c1gm2 = new C1.WPF.Gauge.C1GaugeMark();
    c1gm2.Location = 1.1;
    c1gm2.Interval = 10;
    this.c1LinearGauge1.Decorators.Add(c1gm2);
    C1.WPF.Gauge.C1GaugeLabel c1gm3 = new C1.WPF.Gauge.C1GaugeMark();
    c1gm3.Location = 1.1;
    c1gm3.Interval = 5;
    this.c1LinearGauge1.Decorators.Add(c1gm3);
}
```

プロジェクトの実行と確認

C1LinearGauge コントロールに3種類のサイズの目盛りマークが表示されます。

目盛りマークをカスタマイズする

デフォルトでは、ゲージの目盛りマークは青灰色の四角形として描画されます。**C1GaugeMark** 要素の Template プロパティにカスタムテンプレートを割り当てることで、目盛りマークの外観をカスタマイズできます。以下の手順では、**C1GaugeMark** の外観を定義する新しい **DataTemplate** を作成し、そのテンプレートを **C1RadialGauge** コントロールの **C1GaugeMark** 要素の **Template** プロパティに割り当てます。

次の手順に従います。

1. XAML ビューに切り替えて、3つの <c1:C1GaugeMark> タグを <c1:C1LinearGauge> タグに追加します。次のようになります。

   XAML

   ```
   <c1:C1LinearGauge Height="89" Margin="90,72,41,88" Name="C1LinearGauge1"
   Width="287">
       <c1:C1GaugeMark From="0" To="100" Interval="10" Template="{StaticResource
   MyMarkTemplate}"/>
       <c1:C1GaugeMark Interval="5" Location="1.1" />
   </c1:C1LinearGauge>
   ```

2. 次のマークアップを **UserControl/Window** タグの直後に追加して、テンプレートを追加します。

   XAML

   ```
   <UserControlWindow.Resources>
       <!-- ゲージマークを表示するためのテンプレート -->
       <DataTemplate x:Key="MyMarkTemplate">
   ```

```
        <Rectangle Width="4" Height="18" Fill="BlueViolet" Stroke="Black"
StrokeThickness=".5"/>
    </DataTemplate>
</UserControlWindow.Resources>
                <UserControl.Resources>
    <!-- ゲージマークを表示するためのテンプレート -->
    <DataTemplate x:Key="MyMarkTemplate">
        <Rectangle Width="4" Height="18" Fill="BlueViolet" Stroke="Black"
StrokeThickness=".5"/>
    </DataTemplate>
</UserControl.Resources>
```

このテンプレートは、目盛りマークの外観を定義します。

3. 次に、最初に追加した **C1GaugeMark** 要素に **Template** プロパティを設定して、新しいテンプレートのキーを参照します。

XAML

```
<c1:C1GaugeMark From="0" To="100" Interval="10" Template="{StaticResource
MyMarkTemplate}"/>
```

プロジェクトの実行と確認

C1RadialGauge コントロールに、カスタマイズされた目盛りマークが表示されます。

マークは、**Location** プロパティで指定された位置から内側に向かって描画されています。マークの表示に使用される四角形の **Height** を増やすと、目盛りマークはゲージの中心に向かって長くなります。マークを外側に向けて延ばすには、**C1GaugeMark** 要素の **Location** プロパティを変更します。また、目盛りマークの表示に使用される要素はスケールに沿って向きが変わりますが、ラベルの表示に使用される要素は向きが変わらないことがわかります(「ラベルを追加する」を参照)。

ゲージ形状をカスタマイズする

ほとんどの円形ゲージは丸形ですが、他の形状でゲージを作成することもできます。**C1RadialGauge** の形状をカスタマイズするには、次の手順に従う必要があります。

- ゲージの形状を選択します。
- ゲージの形状を考慮し、ポインタの位置に合わせて **PointerOrigin** プロパティを設定します。
- ゲージの **Background** プロパティを **Transparent** に、**BorderThickness** プロパティを **0** に設定して、デフォルトの丸い背景を非表示にします。

- 要素を **Face** レイヤに追加して、新しいゲージの形状を表示します。

上の手順に基づいてカスタマイズされた形状の **C1RadialGauge** を作成するには、次の手順に従います。

1. XAML ビューに切り替えて、<c1:C1RadialGauge> タグを編集します。次のようになります。

```
XAML
<c1:C1RadialGauge Height="189" Margin="102,34,127,26" Name="C1RadialGauge1"
Width="189" StartAngle="-160" SweepAngle = "140">
</c1gauge>
```

   これで、**C1RadialGauge** コントロールの初期プロパティが設定されます。

2. XAML ビューで、**PointerOrigin="0.8,0.5"** を <c1:C1RadialGauge> タグに追加します。次のようになります。

```
XAML
<c1:C1RadialGauge Height="189" Margin="102,34,127,26" Name="C1RadialGauge1"
Width="189" StartAngle="-160" SweepAngle = "140" PointerOrigin="0.8,0.5">
</c1gauge>
```

   PointerOrigin プロパティは、**C1RadialGauge** コントロールの **Pointer** の開始位置を設定します。

3. XAML ビューで、**Background="Transparent"** を <c1:C1RadialGauge> タグに追加します。次のようになります。

```
XAML
<c1:C1RadialGauge Height="189" Margin="102,34,127,26" Name="C1RadialGauge1"
Width="189" StartAngle="-160" SweepAngle = "140" PointerOrigin="0.8,0.5"
Background="Transparent">
</c1gauge>
```

   これで、**C1RadialGauge** コントロールが透明に表示されます。

4. XAML ビューで、BorderThickness="0" を <c1:C1RadialGauge> タグに追加します。次のようになります。

```
XAML
<c1: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">
</c1gauge>
```

   これで、C1RadialGauge コントロールが境界線なしで表示されます。

5. XAML ビューで、マークアップを <c1:C1RadialGauge> タグの後に追加します。次のようになります。

```
XAML
<c1: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">
    <!-- ゲージに目盛りマークを追加します -->
    <c1:C1GaugeMark Interval="10" Location="1"/>
    <c1:C1GaugeMark Interval="5" Location="1" />
</c1gauge>
```

これで、ゲージに **C1GaugeMark** 要素と目盛りマークが追加されます。

6. XAML ビューで、マークアップを <c1:C1RadialGauge> タグの後に追加します。次のようになります。

XAML

```
<c1: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">
    <!-- ゲージに目盛りマークを追加します -->
    <c1:C1GaugeMark Interval="10" Location="1"/>
    <c1:C1GaugeMark Interval="5" Location="1" />
    <!-- カスタム形状のフェイスを追加します -->
    <c1: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>
    </c1:C1RadialGauge.Face>
</c1gauge>
```

これで、ゲージにカスタマイズされた **Face** が追加されます。

プロジェクトの実行と確認

C1RadialGauge コントロールに、カスタマイズされたフェイスが表示されます。

C1RadialGauge コントロールの **Face** は、さまざまにカスタマイズできます。たとえば、**ComponentOne** と共にインストールされる **GaugeSamples** サンプルの **SpeedometersPage.xaml** ページには、次のカスタマイズされたゲージがあります。

ポインタの外観をカスタマイズする

デフォルトでは、**Pointer** は先細の茶色の四角形として表示され、ポインタキャップはグラデーション付きの灰色の円として表示されます。どちらの外観もカスタマイズできます。以下の手順では、**C1RadialGauge** コントロールの **Pointer** と **PointerCap** の外観をカスタマイズします。

次の手順に従います。

1. **C1RadialGauge** コントロールをクリックして選択します。
2. XAML ビューに切り替えて、**PointerFill="SkyBlue" PointerStroke="CornflowerBlue"** を <c1:C1RadialGauge> タグに追加します。次のようになります。

   XAML
   ```
   <c1:C1RadialGauge Height="226" Margin="22,24,0,12" Name="C1RadialGauge1"
   Width="256" PointerFill="SkyBlue" PointerStroke="CornflowerBlue">
   </c1:C1RadialGauge>
   ```

   これで、**Pointer** の色がカスタマイズされます。
3. XAML ビューで、**PointerCapStroke="CornflowerBlue"** を <c1:C1RadialGauge> タグに追加します。次のようになります。

   XAML
   ```
   <c1:C1RadialGauge Height="226" Margin="22,24,0,12" Name="C1RadialGauge1"
   Width="256" PointerFill="SkyBlue" PointerCapStroke="CornflowerBlue"
   PointerStroke="CornflowerBlue">
   </c1:C1RadialGauge>
   ```

   これで、**PointerCap** の輪郭の色がカスタマイズされます。
4. XAML ビューで、次の <c1:C1RadialGauge.PointerCapFill> マークアップを <c1:C1RadialGauge> タグに追加します。次のようになります。

   XAML
   ```
   <c1:C1RadialGauge Height="226" Margin="22,24,0,12" Name="C1RadialGauge1"
   Width="256" PointerFill="SkyBlue" PointerCapStroke="CornflowerBlue"
   PointerStroke="CornflowerBlue" >
       <c1:C1RadialGauge.PointerCapFill>
           <RadialGradientBrush>
               <GradientStop Color="CornflowerBlue" Offset="0"/>
               <GradientStop Color="SkyBlue" Offset="1"/>
           </RadialGradientBrush>
       </c1:C1RadialGauge.PointerCapFill>
   </c1:C1RadialGauge>
   ```

   これで、**C1RadialGauge** コントロールの **PointerCap** に半径方向のグラデーションが追加されます。

プロジェクトの実行と確認

C1RadialGauge コントロールに、カスタマイズされた **Pointer** と **PointerCap** が表示されます。