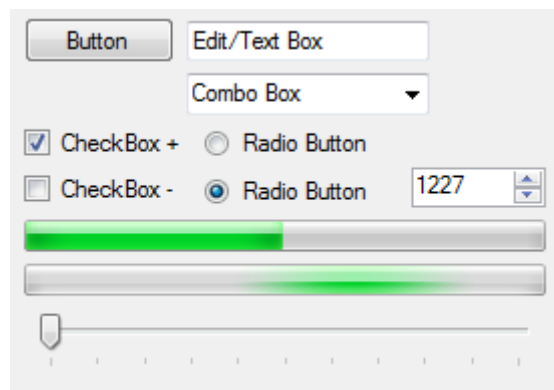


# 6. WPF - vytváříme kontroly

---

**Co je to kontrola?** Kontrola je samostatný interaktivní prvek s grafickým rozhraním.

.NET Framework obsahuje spoustu předvytvořených kontrol (*Button*, *TextBox*, *ComboBox*, *CheckBox*, *RadioButton*, *ProgressBar* ...) což usnadňuje práci programátora, který nemusí neustále implementovat prvky, které používá opakovaně v různých aplikacích.

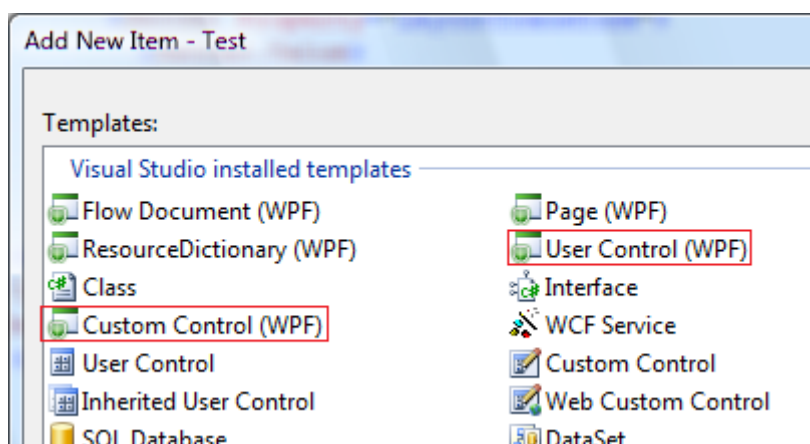


## Začínáme

Možná jste si všimli, kolik vlastností, metod a událostí mají kontroly ve WPF. Abychom je u naší kontroly nemuseli sami implementovat, máme k dispozici dvě třídy - *UserControl* nebo *CustomControl*, ze kterých vycházíme.

## UserControl vs CustomControl

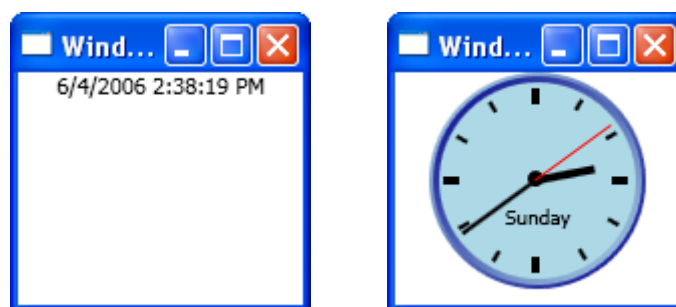
Jaký je mezi nimi rozdíl?



Při odvození z **CustomControls** máme více prostoru při vývoji, jsou zděděny pouze nejdůležitější vlastnosti, metody a události potřebné pro použití kontroly v uživatelském rozhraní. U odvození z *CustomControl* se striktně odděluje logická část od grafické. Počítá se s tím, že vývojář kontroly se zaměří zejména na logiku a až programátor používající tuto kontrolu si ji sám naskinuje. Protože vývoj kontroly odvozené ze třídy *CustomControl* je docela náročné, máme k dispozici ještě *UserControl*.

Vývoj kontroly odvozené od třídy **UserControl** je o mnoho jednodušší, vzhled je na vývojáři kontroly (není zde podpora skinovatelnosti) a programování kontroly se více podobá vývoji samotné aplikace (kontrolu skládáme z jiných komponent).

Tedy vývoj z *CustomControls* je komplikovanější při vývoji, ale flexibilnější při použití.



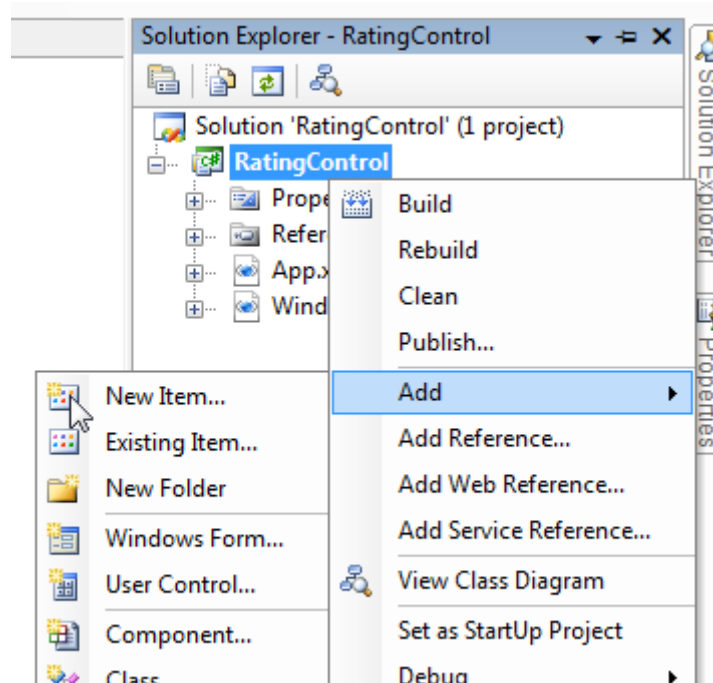
Zde vidíme vytvořenou kontrolu hodin. U prvního obrázku se jedná o *CustomControl* - je zde pouze jednoduché grafické rozhraní. Na druhém obrázku se jedná o *UserControl*. Rozdíl? *UserControl* má vytvořené grafické rozhraní vývojářem kontroly a programátor při použití nemůže tento vzhled změnit. U *CustomControl* je grafické rozhraní velice jednoduché, protože zde jde hlavně o logickou část, ale narozdíl od *UserControly* si můžeme celý vzhled přepsat a může vypadat třeba jako na druhém obrázku. Takže ... u *CustomControl* si můžeme zvolit vlastní vzhled, u *UserControl* si musíme vystačit se vzhledem vytvořeným vývojářem kontroly.

## Vytváříme kontrolu

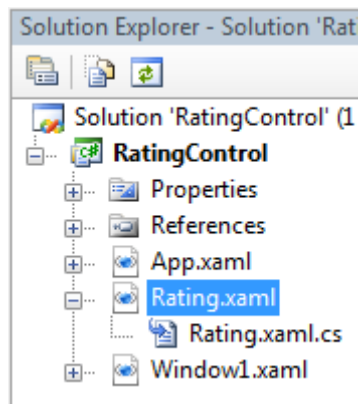
Ukážeme si vývoj kontroly založené na *UserControl* od samotné implementace až po použití v aplikaci.

Nejdříve vytvoříme nový WPF projekt pojmenovaný "RatingControl". Do tohoto projektu vložíme novou *UserControl*.

Pravým tlačítkem myši klikněte na projekt *RatingControl* v *Solution Exploreru* > Add > New Item > *User Control (WPF)* pojmenovanou "Rating"



Do projektu byl přidán soubor *Rating.xaml* a připojený soubor *Rating.xaml.cs*.



*Rating.xaml*

```
<UserControl x:Class="RatingControl.Rating"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Grid>

  </Grid>
</UserControl>
```

### Rating.xaml.cs

```
using System;
using System.Windows;
using System.Windows.Input;
using System.Windows.Controls;
using System.Windows.Shapes;

namespace RatingControl
{
    public partial class Rating : System.Windows.Controls.UserControl
    {
        public Rating()
        {
            InitializeComponent();
        }
    }
}
```

Vložíme kontrolu do aplikace, abychom viděli prováděné změny.

### Window1.xaml

```
<Window x:Class="RatingControl.Window1"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:rc="clr-namespace:RatingControl"
        Title="RatingControl" Height="300" Width="300" Padding="40"
        >
    <StackPanel Margin="40">
        <rc:Rating />
    </StackPanel>
</Window>
```

1. Namapovali jsme namespace *RatingControl* na element *rc*, přes tento element můžeme přistupovat k prvkům v tomto namespace.

```
xmlns:rc="clr-namespace:RatingControl"
```

\* Pokud bychom mapovali kontrolu implementovanou v jiném projektu našeho solution (jiné assembly):

```
xmlns:rc="clr-namespace:RatingControl;assembly=[NázevAssembly]"
```

2. Vložili jsme naši kontrolu *Rating* mezi elementy *StackPanel*.

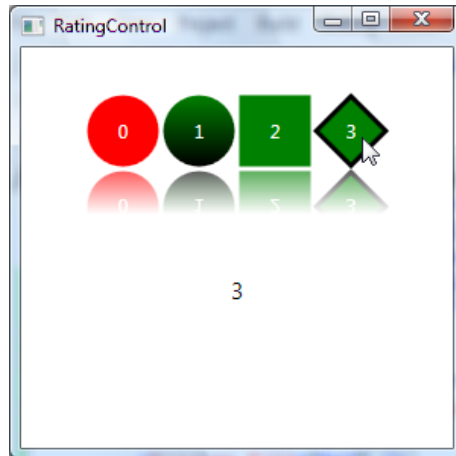
## Implementace kontroly

Vytvoříme kontrolu pro hodnocení ★★★★★ používanou často na webových stránkách.

### Co všechno má umět?

Dáme možnost programátorovi používající kontrolu zadat tvary zobrazené v kontrole a směr vykreslení. Hodnocení bude přístupné přes vlastnost *CurrentRating*. Nakonec jako bonus přidáme efekt - odlesk.

Naše kontrola bude tedy ve finále vypadat trochu nezvykle...



*\* U tohoto příkladu byl kladen důraz spíše na ukázkou nových postupů než na použitelnost kontroly v reálné aplikaci*

Nejdříve vytvoříme vlastnost *Items*, do této vlastnosti programátor vloží tvary, které se v kontrole mají zobrazit. Na obrázku výše jsou v kontrole zobrazeny dva kruhy a dva čtverce, takže do vlastnosti *Items* musely být tyto kontroly vloženy (už i takto nastýlované). Pokud bychom chtěli například jen jeden obdelník a elipsu, tak použití kontroly v aplikaci by mohlo vypadat takto:

```
<Rating>
  <Rating.Items>
    <Rectangle />
    <Ellipse />
  </Rating.Items>
</Rating>
```

### Rating.xaml

```
<UserControl x:Class="RatingControl.Rating"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="200" Height="100">
  <!-- Resources -->
  <UserControl.Resources>
    <Style TargetType="Shape" x:Key="ShapeStyler">
      <Setter Property="Stroke" Value="Transparent" />
      <Setter Property="StrokeThickness" Value="3" />
    <Style.Triggers>
```

```

        <Trigger Property="IsMouseOver" Value="True">
            <Setter Property="Stroke" Value="Black" />
        </Trigger>
    </Style.Triggers>
</Style>
<Style TargetType="TextBlock">
    <Setter Property="Foreground" Value="White" />
    <Setter Property="HorizontalAlignment" Value="Center" />
    <Setter Property="VerticalAlignment" Value="Center" />
    <Setter Property="IsHitTestVisible" Value="True" />
</Style>
</UserControl.Resources>
<Grid Name="MainPanel" />
</UserControl>

```

### Rating.xaml.cs

```

using System;
using System.Windows;
using System.Windows.Input;
using System.Windows.Controls;
using System.Windows.Shapes;
using System.Collections.Generic;

namespace RatingControl
{
    public partial class Rating : System.Windows.Controls.UserControl
    {
        private List<Shape> items = new List<Shape>();
        public List<Shape> Items
        {
            get { return items; }
        }

        public Rating()
        {
            InitializeComponent();
        }

        public override void EndInit()
        {
            base.EndInit();

            MainPanel.ColumnDefinitions.Clear();
            MainPanel.Children.Clear();

            for (int i = 0; i < items.Count; i++)
            {
                // Rating
                TextBlock rating = new TextBlock();
                rating.Text = i.ToString();
                rating.IsHitTestVisible = false;

                // Shape
                items[i].Style = (Style)FindResource("ShapeStyler");

                // Grid
                MainPanel.ColumnDefinitions.Add(new ColumnDefinition());
                Grid.SetColumn(items[i], i);
                Grid.SetColumn(rating, i);
            }
        }
    }
}

```

```

        // Insert elements into application
        MainPanel.Children.Add(items[i]);
        MainPanel.Children.Add(rating);
    }
}
}

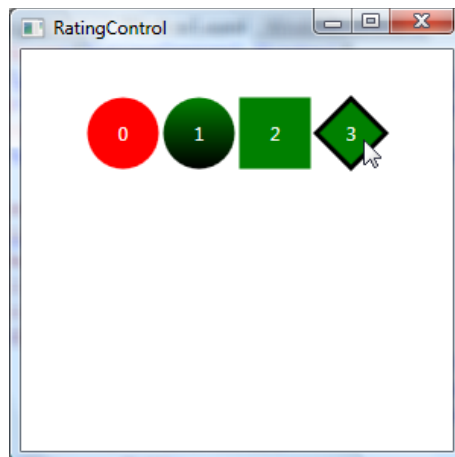
```

### Použití - Window1.xaml

```

<StackPanel Margin="30">
  <rc:Rating Width="200" Height="50">
    <rc:Rating.Items>
      <Ellipse Fill="Red" />
      <Ellipse>
        <Ellipse.Fill>
          <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
            <GradientStop Offset="0" Color="Green" />
            <GradientStop Offset="1" Color="Black" />
          </LinearGradientBrush>
        </Ellipse.Fill>
      </Ellipse>
      <Rectangle Fill="Green" />
      <Rectangle Fill="Green">
        <Rectangle.LayoutTransform>
          <RotateTransform Angle="45" />
        </Rectangle.LayoutTransform>
      </Rectangle>
    </rc:Rating.Items>
  </rc:Rating>
</StackPanel>

```



1. V proměně *items* máme uložené tvary zadané programátorem do vlastnosti *Items*. Jedná se o proměnou typu *List<Shapes>* (pole typu *Shapes*).
2. Přepsali jsme metodu *EndInit()*, která se volá vždy, když je kontrola inicializována. V této metodě voláme původní metodu, poté zpracováváme tvary vložené do *Items* a ty nakonec vkládáme do *Gridu* „MainPanel“.

Rating.xaml.cs

3. V *Resources* elementu *UserControl* jsme vytvořili styl pro prvky typu *Shape* a *TextBox*.

4. Nakonec jsme do *Rating.Items* vložili tvary pro danou instanci kontroly.

- červená elipsa
- elipsa s barevným přechodem (lineární gradient zelená → černá)
- zelený obdelník
- zelený obdelník - rotace 45°

Máme vytvořené rozhraní kontroly a nyní naimplementujeme hodnocení. Uživatel klikne na některý z tvarů (což představuje hodnocení) a to bude uloženo ve vlastnosti *CurrentRating*.

*Rating.xaml.cs*

```
public partial class Rating : System.Windows.Controls.UserControl
{
    private List<Shape> items = new List<Shape>();
    public List<Shape> Items
    {
        get { return items; }
    }

    private int currentRating = -1;
    public int CurrentRating
    {
        get { return currentRating; }
        set { currentRating = value; }
    }

    public Rating()
    {
        InitializeComponent();
    }

    public override void EndInit()
    {
        base.EndInit();

        MainPanel.ColumnDefinitions.Clear();
        MainPanel.Children.Clear();

        for (int i = 0; i < items.Count; i++)
        {
            // Rating
            TextBlock rating = new TextBlock();
            rating.Text = i.ToString();
            rating.IsHitTestVisible = false;

            // Shape
            items[i].Style = (Style)FindResource("ShapeStyler");
            items[i].Tag = i;
            items[i].MouseLeftButtonDown +=
                new MouseButtonEventHandler(ShapeClick);

            // Grid
            MainPanel.ColumnDefinitions.Add(new ColumnDefinition());
        }
    }
}
```

```

        Grid.SetColumn(items[i], i);
        Grid.SetColumn(rating, i);

        // Insert elements into application
        MainPanel.Children.Add(items[i]);
        MainPanel.Children.Add(rating);
    }
}

```

```

void ShapeClick(object sender, MouseButtonEventArgs e)
{
    Shape shape = (Shape)sender;
    currentRating = (int)shape.Tag;
}
}

```

1. Nadeklovali jsme proměnou *currentRating*, kde je uloženo hodnocení. K této proměně přistupujeme přes vlastnost *CurrentRating*.
2. Po kliknutí na tvar se zavolá metoda *ShapeClick()*. V této metodě uložíme do *currentRating* pořadí (=hodnocení) tvaru. O kolikátý prvek v pořadí se jedná poznáme z vlastnosti *Tag*, do které jsme si pořadí uložili.

Nyní máme základní funkce kontroly naimplementované, hodnotu hlasování máme přístupnou přes vlastnost *CurrentRating*. Nicméně hodnotu této vlastnosti můžeme číst, ale nemůžeme ji použít jako zdroj při bindování. Proč? Nejedná se o *DependencyProperty*. Je dobrým zvykem tyto vlastnosti jako *DependencyProperty* naimplimentovat ... a jak na to si ukážeme nyní.

## Změna na Dependency Property

*Rating.xaml.cs*

```

public partial class Rating : System.Windows.Controls.UserControl
{
    private List<Shape> items = new List<Shape>();
    public List<Shape> Items
    {
        get { return items; }
    }
}

```

```

private static readonly DependencyProperty CurrentRatingProperty =
    DependencyProperty.Register
        ("CurrentRating", typeof(int), typeof(Rating));

public int CurrentRating
{
    get { return (int)GetValue(CurrentRatingProperty); }
    set { SetValue(CurrentRatingProperty, value); }
}

```

```

public Rating()
{
    InitializeComponent();
    CurrentRating = -1;
}

```

```

public override void EndInit()
{
    base.EndInit();
    MainPanel.ColumnDefinitions.Clear();
    MainPanel.Children.Clear();

    for (int i = 0; i < items.Count; i++)
    {
        TextBlock rating = new TextBlock(); // Rating
        rating.Text = i.ToString();
        rating.IsHitTestVisible = false;

        // Shape
        items[i].Style = (Style)FindResource("ShapeStyler");
        items[i].Tag = i;
        items[i].MouseDown +=
            new MouseButtonEventHandler(ShapeClick);

        // Grid
        MainPanel.ColumnDefinitions.Add(new ColumnDefinition());
        Grid.SetColumn(items[i], i);
        Grid.SetColumn(rating, i);

        // Insert elements into application
        MainPanel.Children.Add(items[i]);
        MainPanel.Children.Add(rating);
    }
}

void ShapeClick(object sender, MouseButtonEventArgs e)
{
    Shape shape = (Shape)sender;
    CurrentRating = (int)shape.Tag;
}
}

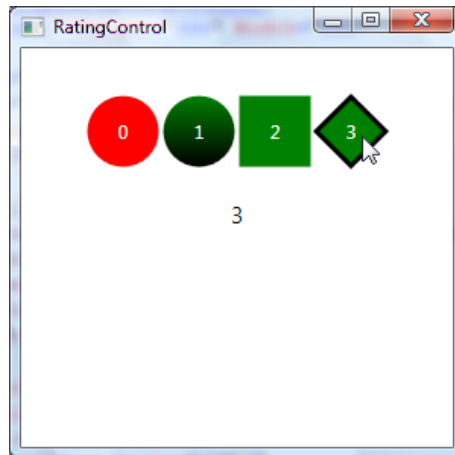
```

### Použití - Window1.xaml

```

<StackPanel Margin="30">
    <rc:Rating Width="200" Height="50" x:Name="myRating">
        <rc:Rating.Items>
            <Ellipse Fill="Red" />
            <Ellipse>
                <Ellipse.Fill>
                    <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
                        <GradientStop Offset="0" Color="Green" />
                        <GradientStop Offset="1" Color="Black" />
                    </LinearGradientBrush>
                </Ellipse.Fill>
            </Ellipse>
            <Rectangle Fill="Green" />
            <Rectangle Fill="Green">
                <Rectangle.LayoutTransform>
                    <RotateTransform Angle="45" />
                </Rectangle.LayoutTransform>
            </Rectangle>
        </rc:Rating.Items>
    </rc:Rating>
    <TextBlock Text="{Binding ElementName=myRating, Path=CurrentRating}"
        TextAlignment="Center" Margin="20" FontSize="15" />
</StackPanel>

```



Rating.xaml.cs

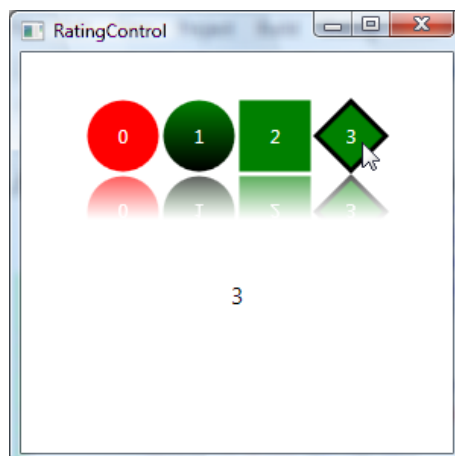
1. Hodnocení již není uloženo ve *fieldu*, ale je nyní uloženo v *CurrentRatingProperty*. Protože se jedná o *DependencyProperty*, hodnotu lze číst nebo ukládat pouze pomocí metod *GetValue()* a *SetValue()*.

Window1.xaml

2. Vytvořili jsme v aplikaci *TextBlock*, do kterého zobrazíme hodnocení. Protože *CurrentRating* je nyní *DependencyProperty*, nabilovali jsme *TextBlock.Text* na tuto vlastnost.

## Odlesk

Využijeme grafické možnosti WPF a přidáme jednoduchý efekt – odlesk. Odlesk je pouze vykreslení kontroly znovu, do ztracena a obráceně.



## Rating.xaml

```
...
</Style>
</UserControl.Resources>
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition />
    <RowDefinition />
  </Grid.RowDefinitions>
  <!-- Zde jsou tvary -->
  <Grid Name="MainPanel" Grid.Row="0" />
  <!-- Zde je vykreslen odlesk -->
  <Border Grid.Row="1">
    <Border.Background>
      <VisualBrush Visual="{Binding ElementName=MainPanel}">
        <!-- Vykreslit obráceně -->
        <VisualBrush.RelativeTransform>
          <TransformGroup>
            <ScaleTransform ScaleY="-1" CenterY="0.5" />
          </TransformGroup>
        </VisualBrush.RelativeTransform>
      </VisualBrush>
    </Border.Background>
    <!-- Vykreslit do ztracena -->
    <Border.OpacityMask>
      <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
        <GradientStop Color="#AA000000" Offset="0" />
        <GradientStop Color="Transparent" Offset="0.6" />
      </LinearGradientBrush>
    </Border.OpacityMask>
  </Border>
</Grid>
</UserControl>
```

1. Vytvořili jsme *Grid* o dvou řádcích, v prvním bude celé rozhraní kontroly, ve druhém odlesk.
2. Vložili jsme *Border*, na jehož pozadí jsme nabindovali *MainPanel*. Výhoda použití bindování je, že automaticky reflektuje změny kontroly v odlesku (například rámeček po najetí myši nad jakýkoliv tvar se objeví i v odrazu)
3. Pozadí jsme otočili a přidali *OpacityMask* → lineární gradient definující průhlednost

## Směr vykreslení

Poslední vlastnost, kterou budeme implementovat, je směr vykreslení kontroly. Programátor bude mít možnost si vybrat, zda chce aby se kontrola vykreslila Horizontálně nebo Vertikálně.

## Rating.xaml.cs

```
namespace RatingControl
{
    public enum OrientationEnum { Vertical, Horizontal }

    public partial class Rating : System.Windows.Controls.UserControl
    {
        public static readonly DependencyProperty OrientationProperty =
            DependencyProperty.Register
                ("Orientation", typeof(OrientationEnum), typeof(Rating));

        public OrientationEnum Orientation
        {
            get { return (OrientationEnum)GetValue(OrientationProperty); }
            set { SetValue(OrientationProperty, value); }
        }

        private List<Shape> items = new List<Shape>();
        public List<Shape> Items
        {
            get { return items; }
        }

        private static readonly DependencyProperty CurrentRatingProperty =
            DependencyProperty.Register
                ("CurrentRating", typeof(int), typeof(Rating));

        public int CurrentRating
        {
            get { return (int)GetValue(CurrentRatingProperty); }
            set { SetValue(CurrentRatingProperty, value); }
        }

        public Rating()
        {
            InitializeComponent();
            CurrentRating = -1;
        }

        public override void EndInit()
        {
            base.EndInit();

            MainPanel.ColumnDefinitions.Clear();
            MainPanel.Children.Clear();

            for (int i = 0; i < items.Count; i++)
            {
                // Rating
                TextBlock rating = new TextBlock();
                rating.Text = i.ToString();
                rating.IsHitTestVisible = false;

                // Shape
                items[i].Style = (Style)FindResource("ShapeStyler");
                items[i].Tag = i;
                items[i].MouseDown +=
                    new MouseButtonEventHandler(ShapeClick);
            }
        }
    }
}
```

```

        // Grid
        MainPanel.ColumnDefinitions.Add(new ColumnDefinition());
        Grid.SetColumn(items[i], i);
        Grid.SetColumn(rating, i);

        // Vlozeni prvku do aplikace
        MainPanel.Children.Add(items[i]);
        MainPanel.Children.Add(rating);
    }
}

void ShapeClick(object sender, MouseButtonEventArgs e)
{
    Shape shape = (Shape)sender;
    CurrentRating = (int)shape.Tag;
}
}
}

```

### Rating.xaml

```

<UserControl x:Class="RatingControl.Rating"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:rc="clr-namespace:RatingControl"
    Width="200" Height="100" Name="MyControl">
    <!-- Orientation -->
    <UserControl.Style>
        <Style TargetType="{x:Type rc:Rating}">
            <Style.Triggers>
                <DataTrigger Binding="{Binding ElementName=MyControl,Path=Orientation}">
                    <DataTrigger.Value>
                        <rc:OrientationEnum>Vertical</rc:OrientationEnum>
                    </DataTrigger.Value>
                    <Setter Property="LayoutTransform">
                        <Setter.Value>
                            <RotateTransform Angle="90" />
                        </Setter.Value>
                    </Setter>
                </DataTrigger>
            </Style.Triggers>
        </Style>
    </UserControl.Style>
    <!-- Resources -->
    <UserControl.Resources>
        <Style TargetType="Shape" x:Key="ShapeStyler">
            <Setter Property="Stroke" Value="Transparent" />
            <Setter Property="StrokeThickness" Value="3" />
            <Style.Triggers>
                <Trigger Property="IsMouseOver" Value="True">
                    <Setter Property="Stroke" Value="Black" />
                </Trigger>
            </Style.Triggers>
        </Style>
        <Style TargetType="TextBlock">
            <Setter Property="Foreground" Value="White" />
            <Setter Property="HorizontalAlignment" Value="Center" />
            <Setter Property="VerticalAlignment" Value="Center" />
            <Setter Property="IsHitTestVisible" Value="True" />
        </Style>
    </UserControl.Resources>

```

```

</UserControl.Resources>
<!-- Implementace -->
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition />
    <RowDefinition />
  </Grid.RowDefinitions>
  <Grid Name="MainPanel" Grid.Row="0" />
  <Border Grid.Row="1">
    <Border.Background>
      <VisualBrush Visual="{Binding ElementName=MainPanel}">
        <VisualBrush.RelativeTransform>
          <TransformGroup>
            <ScaleTransform ScaleY="-1" CenterY="0.5" />
          </TransformGroup>
        </VisualBrush.RelativeTransform>
      </VisualBrush>
    </Border.Background>
    <Border.OpacityMask>
      <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
        <GradientStop Color="#AA000000" Offset="0" />
        <GradientStop Color="Transparent" Offset="0.6" />
      </LinearGradientBrush>
    </Border.OpacityMask>
  </Border>
</Grid>
</UserControl>

```

### Window1.xaml

```

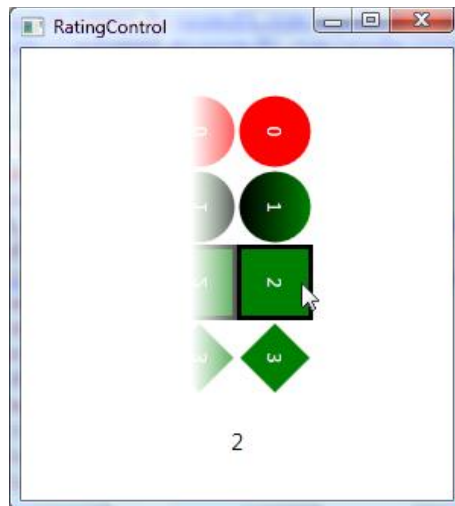
<Window x:Class="RatingControl.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:rc="clr-namespace:RatingControl"
  Title="RatingControl" Height="300" Width="300"
  >
  <StackPanel Margin="30">
    <rc:Rating Width="200" Height="100"
      x:Name="myRating" Orientation="Vertical">
      <rc:Rating.Items>
        <Ellipse Fill="Red" />
        <Ellipse>
          <Ellipse.Fill>
            <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
              <GradientStop Offset="0" Color="Green" />
              <GradientStop Offset="1" Color="Black" />
            </LinearGradientBrush>
          </Ellipse.Fill>
        </Ellipse>
        <Rectangle Fill="Green" />
        <Rectangle Fill="Green">
          <Rectangle.LayoutTransform>
            <RotateTransform Angle="45" />
          </Rectangle.LayoutTransform>
        </Rectangle>
      </rc:Rating.Items>
    </rc:Rating>
    <TextBlock Text="{Binding ElementName=myRating, Path=CurrentRating}"
      TextAlignment="Center" Margin="20" FontSize="15" />
  </StackPanel>
</Window>

```

1. Vytvořili jsme *DependencyProperty* typu *OrientationEnum*.
2. Směr vykreslení jsme implementovali v XAMLu. Použili jsme *DataTrigger* - pokud má vlastnost *Orientation* hodnotu *Vertical*, otočí se kontrola o 90° (protože *DataTrigger* nelze použít přímo u kontroly, použili jsme styl).
3. U *DataTrigru* je nutné hodnotu vlastnosti *Orientation* uvést jako typ *OrientationEnum*  
`<rc:OrientationEnum>Vertical</rc:OrientationEnum>`

Při použití u kontroly v aplikaci můžeme zadat hodnotu jednoduše jako řetězec

```
... Orientation="Vertical">
```



## Závěr

Při implementaci této kontroly byl kladen důraz zejména na použití nových postupů. Naučili jsme se přistupovat k vlastním třídám a enumerátorům v XAMLu, použili jsme nově transformace (rotace) a lineární gradient, programově použití stylů, vytváření *DependencyProperties* a samozřejmě celý vývoj kontroly založený na *UserControl* včetně použití v aplikaci.

Více informací o WPF naleznete na [www.unitedstatesof.net/wpf](http://www.unitedstatesof.net/wpf)

Aleš Šturala, 10. 2. 2007

<http://hidentity.org/hid/CZ123456>