

WPF kontrole

Maja Štula

Ak. God. 2011/2012

WPF kontrole

- WPF kontrola je klasa koja predstavlja vidljivi objekt (sa vizualnim prikazom) u aplikaciji.
- Kontrola se dodaje ili preko XAML ili preko programskog koda aplikacije.
- Izgled kontrole može se prilagoditi na tri načina:
 - Postavljanjem vrijednosti svojstva kontrole koji želimo promijeniti (npr. Button Width="400")
 - Kreiranjem stila (preko Style klase) kontrole.
 - Kreiranjem novog ControlTemplate kontrole.

WPF kontrole

- Stil se može postaviti na bilo kojem elementu koji nasljeđuje klasu FrameworkElement ili FrameworkContentElement.
- Stil se definira kao resurs unutar Application.Resource elementa unutar Application elementa ili unutar Window.Resource elementa unutar Window elementa.
- Doseg stila je određen mjestom definicije. Ako je definiran unutar Application elementa može se koristiti bilo gdje u aplikaciji.
- Deklaracija stila se sastoji od objekta Style koji sadrži kolekciju jednog ili više objekata tipa Setter. Svaki Setter objekt se sastoji od Property i Value. Property je ime svojstva elementa na koji se stil primjenjuje.
- Klasa System.Windows.Style omogućava dijeljenje svojstava, resursa i event handlera među instancama istog tipa.

WPF kontrole - stil

- PRIMJER:

<!--Stil se primijenjuje na sve TextBlocks-->

<Style TargetType="TextBlock">

<Setter Property="HorizontalAlignment" Value="Center" />

<Setter Property="FontFamily" Value="Comic Sans MS"/>

<Setter Property="FontSize" Value="14"/>

</Style>

WPF kontrole - stil

- Definiranje stila:

```
<Style x:Key="Style1"><Setter  
    Property="Control.Background"  
    Value="Yellow"/></Style>
```

- Primjena definiranog stila:

```
<Label Width="100" Height="40"  
    Style="{StaticResource  
    Style1}">Labela</Label>
```

WPF kontrole - stil

- Atribut x:Key se koristi za jedinstvenu identifikaciju elementa definiranog u resursu.
- Pored atributa x:Key postoji i atribut x:Name koji ima istu ulogu kao i Name attribute, jedinstveno identificira instancu elementa.
- Kada se u WPF-u koriste resursi kao stilovi, predlošci i sl. oni mogu biti specificirani kao StaticResource ili kao DynamicResource.

WPF kontrole - stil

- `StaticResource` će se postaviti i dodjeliti svojstvu uz koje je vezan tijekom učitavanja XAML prije no što se aplikacija ustvari počme izvoditi. Sve buduće promjene na rječniku resursa se ignoriraju.
- `DynamicResource` dodjeljuje svojstvu vrijednost tijekom učitavanja XAML, ali se vrijednost ustvari određuje prilikom izvođenja aplikacije jer je vrijednost ustvari izraz koji se treba izračunati.

WPF kontrole - stil

- XAML

<Grid>

<!-- Uses a dynamic resource to set the background of a button. If the desktop brush changes while this application is running, this button will be updated. -->

<Button Background="{DynamicResource {x:Static SystemColors.DesktopBrushKey}}" Content="Hello, World!" />

</Grid>

- Personalize -> Window color -> Advanced appearance settings -> Desktop color

WPF kontrole

- Kontrole nasljeđuje klasu `System.Windows.Controls.Control`.
- Klase koje nasljeđuju klasu `Control` sadrže i svojstvo `Template` tipa klase `System.Windows.Controls.ControlTemplate`, koje korisniku kontrole omogućava izmjenu izgleda kontrole bez kreiranja nove izvedene klase.
- Autor kontrole definira defaultni `ControlTemplate`, a korisnik kontrole ga može pregaziti sa svojim definiranim `ControlTemplateom` da bi promijenio izgled kontrole.

WPF kontrole

- Klasa Control predstavlja temeljnu klasu za elemente korisničkog sučelja koji koriste klasu ControlTemplate za iscrtavanje kontrole tj. definiranje izgleda kontrole.
- Svojstvo Template (tipa klase ControlTemplate) specificira izgled kontrole.
- Kada se mijenja izgled kontrole uz zadržavanje funkcionalnosti kontrole to se radi kreiranjem novog ControlTemplate, a ne kreiranjem nove klase.
- Tek kada radite proširenje ponašanja kontrole potrebno je kreirati novu klasu koja nasljeđuje kontrolu.

WPF kontrole

- PRIMJER:

```
<Style TargetType="Button">  
<Setter Property="Template">  
  <Setter.Value>  
    <ControlTemplate TargetType="Button">  
      <Grid>  
        <Ellipse Fill="Red"/>  
        <ContentPresenter HorizontalAlignment="Center"  
          VerticalAlignment="Center"/> </Grid>  
      </ControlTemplate>  
    </Setter.Value>  
  </Setter>  
</Style>
```

XAML atributi

- XAML elementi imaju dva tipa atributa. Tzv. ovisna svojstva (*dependency properties*) su elementi iz CLR klase, koja odgovara promatranom XAML elementu, a izvedeni su iz DependencyProperty klase i imaju deklarirane CLR pristupne metode.
- Vrijednost takvih svojstava ovisi (*dependent*) ili o vrijednosti drugih varijabli, resursa ili se čak može proračunavati prilikom izvršavanja aplikacije poput svojstva Template.
- Drugi tip atributa su “uobičajena” CLR svojstva (*Common language runtime properties*). CLR svojstva su standardna svojstva CLR klase kojima se može pristupiti direktno.

XAML atributi

- WPF pruža skup servisa kojima se proširuje funkcionalnost “uobičajnih” CLR svojstava uvodeći ovisna svojstva. Ti se servisi zajednički zovu WPF sustav svojstva (property system).
- Vrijednost *dependency properties* se određuje iz sljedećih izvora jer WPF sustav svojstva traži vrijednost takvog svojstva na sljedećim mjestima:
 - Lokalno postavljena vrijednost (i.e., <Object Property="value">)
 - Postavljeni Style
 - Nasljeđena vrijednost (od XAML parent elementa, ne nadklase!!)
 - DefaultValue specified when you registered the property (or override metadata)
 - Storyboards or event triggers that start an animation; property values set by an animation override even local values
 - Property triggers
 - TemplatedParent 's template (i.e., that template includes <Setter>)
 - ThemeStyle


XAML atributi

- Atributi tipa zavisnih svojstava podržavaju definiranje vrijednosti preko izraza, defaultne vrijednosti, nasljeđivanja, povezivanja sa izvorima podatka itd. (*value expressions, property invalidation, default values, inheritance, data binding, animation, and styling*).
- WPF sustav svojstva pruža jednostavne get i set pristupnike za dohvaćanje i postavljanje vrijednosti takvih svojstava.
- Npr. ako se u atributu vrijednost navede u zagradama {}, XAML parser to interpretira na način da string naveden u {} zamijeni sa prethodno instanciranim odgovarajućim elementom.

XAML atributi

- Npr.

```
<Style TargetType="Button">  
<Setter Property="Template"> <Setter.Value>  
    <ControlTemplate TargetType="Button"><Grid>  
        <Ellipse Fill="{TemplateBinding Background}"/>  
        <ContentPresenter HorizontalAlignment="Center"  
            VerticalAlignment="Center"/>  
    </Grid></ControlTemplate>  
</Setter.Value></Setter></Style>
```



Ovdje će se izračunati Background od parent elementa

WPF kontrole

- Layout kontrole se koriste za određivanje dimenzija, te pozicioniranje elemenata smještenih na kontrolu:
 - Border, BulletDecorator, Canvas, DockPanel, Expander , Grid , GridSplitter, GroupBox, Panel, ResizeGrip , Separator, ScrollBar, ScrollViewer, StackPanel, Thumb, Viewbox, VirtualizingStackPanel, Window, WrapPanel

Grid primjer

- Definicije redaka i kolona se moraju napraviti prije dodavanja child elemenata preko dva svojstva Grid.Column i Grid.Row.

```
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="Auto" />
<RowDefinition Height="*" />
<RowDefinition Height="28" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto" />
<ColumnDefinition Width="200" />
</Grid.ColumnDefinitions>
</Grid>
```

- Pozicioniranje child elementa na gridu se radi preko svojstava Grid.Column i Grid.Row

```
<Label Grid.Row="0" Grid.Column="0"
      Content="Name:"/>
<Label Grid.Row="1" Grid.Column="0"
      Content="E-Mail:"/>
<Label Grid.Row="2" Grid.Column="0"
      Content="Comment:"/>
<TextBox Grid.Column="1" Grid.Row="0"
        Margin="3" />
<TextBox Grid.Column="1" Grid.Row="1"
        Margin="3" />
<TextBox Grid.Column="1" Grid.Row="2"
        Margin="3" />
<Button Grid.Column="1" Grid.Row="3"
        HorizontalAlignment="Right"/>
```

WPF kontrole

- Botuni su najčešće korištene kontrole. Obično se na događaj klika mišem na botun generira neka akcija.
 - Button
- Izbornici omogućavaju grupiranje povezanih aktivnosti.
 - Menu
 - ToolBar

WPF kontrole

- Seleksijske kontrole omogućavaju korisniku izbor jednog ili više elemenata:
 - CheckBox
 - ComboBox
 - ListBox
 - ListView
 - TreeView
 - RadioButton
 - Slider

WPF kontrole

- Navigacijske kontrole podržavaju navigaciju kroz aplikaciju.
 - Frame
 - Hyperlink
 - Page
 - NavigationWindow
 - TabControl
- Dijalog prozori (Microsoft.Win32 imenski prostor) podržavaju uobičajene funkcionalnosti poput izbora datoteke sa diska.
 - OpenFileDialog
 - PrintDialog
 - SaveFileDialog

WPF kontrole

- Kontrole za pružanje informacija korisniku:
 - Label
 - ProgressBar
 - StatusBar
 - TextBlock
 - ToolTip
- Specijalizirane kontrole za otvaranje dokumenata:
 - DocumentViewer
 - FlowDocumentPageViewer
 - FlowDocumentReader
 - FlowDocumentScrollViewer
 - StickyNoteControl

WPF kontrole

- Ulazne kontrole omogućavaju korisniku unos sadržaja.
 - TextBox
 - RichTextBox
 - PasswordBox
- Kontrole za rad su audio i video sadržajem:
 - Image
 - MediaElement
 - SoundPlayerAction
- Digital ink kontrole pružaju podršku za rad sa Tablet PC.
 - InkCanvas
 - InkPresenter

WPF kontrole – “dockanje”

- Da bi kontrola zauzela cijelo područje kontrole roditelja (u .NET 2.x se to radilo preko svojstva Dock) potrebno je postaviti sljedeća svojstva:
 - Width i Height svojstvo treba postaviti na Auto
 - HorizontalAlignment i VerticalAlignment treba postaviti na Stretch
 - Margin svojstvo treba postaviti na 0

TRIGERI

Triger

- Sve klase koje nasljeđuju klasu FrameworkElement nasljeđuju i svojstvo (property) Triggers koje dohvaća kolekciju trigeru uspostavljenih na promatranom elementu ili na njegovim child elementima.
- `public TriggerCollection Triggers { get; }`

Triger

- Pojednostavljeno trigeri su “nekakav podskup” događaja koji se može definirati deklarativno u XAML-u. Trigeri se mogu postaviti nad samo jednu instancu elementa ili mogu djelovati na cijelu klasu elemenata.
- Trigeri između ostalog omogućavaju promjenu atributa elementa grafičkog sučelja kada se desi određeni događaj. Npr. ako se želi promijeniti boja teksta kada se miš pozicionira iznad teksta, ili promijeniti veličina botuna nakon što je korisnik kliknu na botun.

Triger

- Postoje tri vrste trigera:
 1. Property Triggers se aktivira kada određeno svojstvo poprimi određenu vrijednost.
 2. Event Triggers se aktivira kada se desi određeni događaj (event).
 3. Data Triggers se aktivira kada vezani izraz (binding expression) poprimi određenu vrijednost.

Triger

- WPF definira svojstva (i to dependency) koja odgovaraju korisničkim akcijama poput `IsMouseOver` svojstva koje postaje `true` kada korisnik prođe mišem iznad `UIElementa` ili `ContentElementa` (sve klase koje nasljeđuju ove klase (praktički sve WPF klase) imaju to svojstvo).
(`public bool IsMouseOver { get; }`)
- Pretavaranje korisničke akcije u svojstvo, uz odgovarajući `Trigger element`, omogućava WPF stilu da promijeni vrijednost bilo kojeg svojstva na osnovu korisničke akcije i to u XAML-u.

Triger

- Property trigger (Trigger) se može definirati samo unutar Style, ControlTemplate ili DataTemplate elementa.
- Na pojedinačnom elementu se može postavljati samo EventTriggers. Ako se na pojedinoj kontroli želi postaviti property trigger možemo ga umetnuti u kod:

```
<Control> <Control.Template> <ControlTemplate> <!--  
    Triggers are available here --> </ControlTemplate>  
</Control.Template> </Control>
```

Property trigger

- Primjer postavljanja property trigeru kroz stil elementa. Potrebno je postaviti i Property i Value.

- PRIMJER:

```
<Style TargetType="Button">
```

```
<Style.Triggers>
```

```
  <Trigger Property="IsMouseOver" Value="true">
```

```
    <Setter Property = "Foreground" Value="Green"/>
```

```
    <Setter Property = "Background" Value="Red"/>
```

```
  </Trigger>
```

```
</Style.Triggers>
```

```
</Style>
```

Triger

- Svojstva koja ovise o trigeru se automatski vraćaju u prethodno stanje kada triger više ne postoji.
- Trigeri su optimizirani za prijelazna stanja za koja se očekuje da traju neko vrijeme, a onda prestanu, kao npr. `IsPressed` na `Button` ili `IsSelected` na `ListBoxItem`. Tako `Trigger` objekt ima uz svojstvo `Setters`, i `EnterAction` i `ExitAction` koji primjenjuju akciju na osnovu stanja svojstava.

Triger

- Trigeri su uvjetovani. To je ustvari implementacija `if...then` logike u XAML-u.
- Triger provjerava da li je zadovoljen određeni *if* uvjet i ako je, implementira određeni stila na element. Npr. *if* ako je kursor iznad botuna, *then* promjeni boju pozadine botuna.

EventTrigger

- EventTrigger je trigger koji primjenjuje neku akciju kao odgovor na neki događaj.
- EventTrigger pokreće akciju kada se desi navedeni rutirani događaj i za razliku od Trigger objekta nema pojam završetka akcije.
- TriggerAction klasa opisuje akciju koja će se izvesti na određeni trigger. To je zavisno svojstvo (public abstract class TriggerAction : DependencyObject)

EventTrigger

- Svojstvo trigger se može postavljati u XAML-u kroz sintaksu

```
<Window.Triggers>
```

```
<EventTrigger RoutedEvent="Button.Click"  
  SourceName="clickButton">
```

```
<EventTrigger.Actions><SoundPlayerAction  
  Source="C:\Windows\winsxs\x86_microsoft-windows-shell-  
  sounds_31bf3856ad364e35_6.1.7600.16385_none_16e8d2  
  5616dd2c98\chimes.wav"/></EventTrigger.Actions>
```

```
</EventTrigger></Window.Triggers>
```

- Ili kroz poziv metode Add na kolekciji trigerera.
- Svojstvo je read-only, dok je sama kolekcija read-write.

WPF OSNOVNE ANIMACIJE

Animacije

- WPF animacije implementira kroz *Storyboard* klasu.
- Svaka instanca ove klase može sadržati jednu ili više vremenskih niti izvođenja animacija i svaka od ovih može sadržati jednu ili više animacija.
- Postoje razni tipovi animacija sa različitim opcijama.

Animacije

- Najjednostavnija animacija je *DoubleAnimation*.
- Ona mijenja vrijednost svojstva od određene do određene vrijednosti u nekom vremenskom razmaku.
- Općenito se ova animacija koristi u linearnoj zamjeni vrijednosti, no može se također koristiti i diskretno, tako da animacija skače od vrijednosti do vrijednosti.

Animacije

```
<Button Content="Button" Height="23" HorizontalAlignment="Left" Name="button1"
    VerticalAlignment="Top" Width="75" Click="button1_Click" >
<Button.RenderTransform> <TransformGroup>
<RotateTransform Angle="0"></RotateTransform>
<TranslateTransform X="0"></TranslateTransform>
</TransformGroup></Button.RenderTransform>
<Button.Triggers><EventTrigger RoutedEvent="Button.Click">
<BeginStoryboard><Storyboard AutoReverse="True">
<DoubleAnimation From="0" To="360" Storyboard.TargetName="button1"
    Storyboard.TargetProperty="(Button.RenderTransform).(TransformGroup.Children)
    [0].Angle" Duration="0:0:3"></DoubleAnimation>
<DoubleAnimation From="0" To="200" Storyboard.TargetName="button1"
    Storyboard.TargetProperty="(Button.RenderTransform).(TransformGroup.Children)
    [1].X" Duration="0:0:3"></DoubleAnimation>
</Storyboard> </BeginStoryboard>
</EventTrigger></Button.Triggers>
</Button>
```