WPF

NAPREDNI GUI DIZAJN

KOMANDE, STILOVI I OKIDAČI

Komande

- Mehanizam za unos u program koji omogućava rukovanje unosima na višem semantičkom nivou od unosa sa uređaja.
- *Primeri: Cut, Copy, Paste*

Komande

- Namena:
  - Odvajanje semantike i objekta koji poziva komandu od logike koja izvršava komandu
  - Omogućava da višestruki i različiti izvori pozivaju istu naredbenu logiku i prilagođavanje logike naredbe za različite ciljeve
  - Primer: *Copy, Cut i Paste* mogu da se pozovu upotrebom različitih korisničkih radnji
- Možete povezati svaku vrstu korisničke akcije sa istom logikom
- Označava da li je neka radnja dostupna
  - Komanda može da označi da li je neka radnja dostupna upotrebom *CanExecute* metode
  - Dugme se pretplati na događaj *CanExecuteChanged* i u zavisnosti od povratne vrednosti metode *CanExecute* može biti omogućeno ili onemogućeno

Komande

- Semantika naredbe može da bude konzistentna u svim aplikacijama i klasama, ali logika akcije je specifična za određene objekte u zavisnosti od objekta na koji se naredba primenjuje.
- *RoutedCommand* omogućava primenu logike.
- Kada aplikacija obrađuje izvršen događaj, ona ima pristup cilju naredbe i može preduzeti odgovarajuće akcije u zavisnosti od tipa cilja.

Komande

- Vrste komandi:
  - Ugrađene komande
  - Korisničke komande

Komande

- Komande se kreiraju primenom *ICommand* interfejsa, koji omogućava:
- Metode:
  - ***Execute*** *– Metoda koja sadrži izvršavanje naredbe. Automatski se pokreće kada korisnik pravilno komunicira sa kontrolom ili pritisne kombinaciju tastera koja je povezana sa komandom.*
  - ***CanExecute*** *– Poziva se da bi se utvrdilo da je komanda trenutno dostupna ili ne. Izvršava se kada korisnik pokuša da pokrene komandu i kao odgovor na određene događaje. Ako metoda vrati false, metoda Execute neće biti pozvana.*

- Događaj
  - ***CanExecuteChanged*** *događaj – Aktivira se kada se promeni status atributa CanExecute.*

Komande

- Ugrađene komande
  - Najjednostavniji način za upotrebu naredbi u WPF-u
  - Kontrole imaju izvornu podršku za rukovanje i pozivanje naredbe

- Skup uobičajenih komandi: *MediaCommands*, *ApplicationCommands*, *NavigationCommands*, *ComponentCommands i EditingCommands*

Komande

```
// Creating the UI objects
StackPanel mainStackPanel = new StackPanel();
TextBox pasteTextBox = new TextBox();
Menu stackPanelMenu = new Menu();
MenuItem pasteMenuItem = new MenuItem();

// Adding objects to the panel and the menu
stackPanelMenu.Items.Add(pasteMenuItem);
mainStackPanel.Children.Add(stackPanelMenu);
mainStackPanel.Children.Add(pasteTextBox);

// Setting the command to the Paste command
pasteMenuItem.Command = ApplicationCommands.Paste;

// Setting the command target to the TextBox
pasteMenuItem.CommandTarget = pasteTextBox;
```

```
<StackPanel>
  <Menu>
    <MenuItem Command="ApplicationCommands.Paste" />
  </Menu>
  <TextBox />
</StackPanel>
```

Primer: Prikaz kako postaviti *MenuItem* tako da kada se klikne na njega pozove se naredba *Paste* na *TextBox-u*, pod pretpostavkom da *TextBox* ima fokus.

Komande - *RoutedCommand*

- *RoutedCommand*
  - Metode *Execute* i *CanExecute* ne sadrže logiku aplikacije za naredbu, već podižu usmerene događaje koji se tuneliraju i provlače kroz stablo elemenata sve dok ne naiđu na objekat koji ima *CommandBinding* za tu određenu naredbu.
    - *Execute -> PreviewExecuted i Executed*
    - *CanExecute -> CanExecute i PreviewCanExecute*

Komande

- *RoutedCommand-a* može da se podeli na 4 glavna koncepta:
  - Komanda – akcija koja treba da se izvrši.
  - Izvor naredbe – objekat koji poziva naredbu.
  - Cilj naredbe – objekat na kojem se komanda izvršava.
  - Vezivanje naredbe (*CommandBinding*) – objekat koji preslikava logiku naredbe na naredbu. Obično ga isporučuje kontrola koja je cilj naredbe, ali ne mora to uvek da bude slučaj. Često moraju da ga kreiraju programeri ili bi onda on mogao biti vezan za pretka ciljne naredbe.

Komande – izvor naredbe

- Objekat koji poziva naredbu.
  - *Primer: MenuItem, Button, KeyGesture*
- Primenjuju *ICommandSource* interfejs, koji poseduje tri svojstva:
  - *Command* – naredba koja treba da se izvrši kada je izvor naredbe pozvan.
  - *CommandTarget* – objekat nad kojem treba izvršiti naredbu.
  - *Postavljen:*
    - *RoutedCommand komanda -> primenjuje*
    - *Komanda nije RoutedCommand -> CommandTarget se zanemaruje*
  - *Nije postavljen -> element sa fokusom na tastaturi će biti cilj naredbe.*
- *CommandParameter* – korisnički definisani tip podataka koji se koriste za prosleđivanje implementiranim obrađivačima komande.
- *Primer WPF klasi: ButtonBase, MenuItem i Hyperlink (pozivaju naredbe kada se klikne na njih) i InputBinding (poziva naredbe kada se izvrši InputGesture)*

Komande – izvor naredbe

- Izvor naredbe osluškuje događaj ***CanExecuteChanged***. Ovaj događaj obaveštava da se sposobnost komande za izvršavanje na trenutnom cilju naredbe možda promenila. Trenutni status naredbe izvor može da proveri pomoću metode ***CanExecute***. Ako naredba ne može da se izvrši onda će izvor naredbe biti onemogućen.
- *InputeGesture*
  - *KeyGesture -> prečica na tastaturi, sastoji se od Key i ModifierKeys*
  - *MouseGesture -> sastoji se od MouseAction-a i opcionog skupa ModifierKeys*
  - *Da bi delovao kao izvor naredbe, mora biti povezan sa naredbom. Za povezivanje može da se koristi InputBinding.*

Komande - *CommandBinding*

- Povezuje komandu sa obrađivačem događaja koji implementira komandu.
- Ova klasa sadrži:
  - Atribut:
    - *Command – naredba sa kojom je CommandBinding povezuje.*
  - Događaje:
    - *PreviewExecuted – implementira logiku naredbe*
    - *Executed - implementira logiku naredbe*
    - *PreviewCanExecute – određuje da li se naredba može izvršiti na trenutnom cilju naredbe*
    - *CanExecute - određuje da li se naredba može izvršiti na trenutnom cilju naredbe*

Komanda - *CommandBinding*

Kreiranje CommanBinding na korenskom elementu Window aplikacije.

```
<Window.CommandBindings>
  <CommandBinding Command="ApplicationCommands.Open"
                  Executed="OpenCmdExecuted"
                  CanExecute="OpenCmdCanExecute"/>
</Window.CommandBindings>
```

```
// Creating CommandBinding and attaching an Executed and CanExecute handler
CommandBinding OpenCmdBinding = new CommandBinding(
    ApplicationCommands.Open,
    OpenCmdExecuted,
    OpenCmdCanExecute);

this.CommandBindings.Add(OpenCmdBinding);
```

```
void OpenCmdExecuted(object target, ExecutedRoutedEventArgs e)
{
    String command, targetobj;
    command = ((RoutedCommand)e.Command).Name;
    targetobj = ((FrameworkElement)target).Name;
    MessageBox.Show("The " + command +  " command has been invoked on target object " + targetobj);
}
```

OpenCmdExecuted implementira logiku naredbe, a OpenCmdCanExecute kaže da naredba može da se izvrši na trenutnom cilju.

```
void OpenCmdCanExecute(object sender, CanExecuteRoutedEventArgs e)
{
    e.CanExecute = true;
}
```

Komanda - *CommandBinding*

- *CommandBinding* se povezuje na određeni objekat, kao što je objekat *Window*. Objekat za koji vezan definiše obim vezivanja.
  - Nekada je povezan na sam cilj naredbe.
  - *TextBox -> Cut, Copy i Paste*

Pogodnije je da CommandBinding povežete sa pretkom cilja naredbe, posebno ako se isti može koristiti za više ciljeva komande.

Komanda – *Command Target*

- Element na kojem se komanda izvršava.
- Može eksplicitno da se postavi upotrebom atributa *CommandTarget*. Ukoliko nije postavljen onda se element koji ima fokus koristi kao cilj naredbe.

```
<StackPanel>
  <Menu>
    <MenuItem Command="ApplicationCommands.Paste"
              CommandTarget="{Binding ElementName=mainTextBox}" />
  </Menu>
  <TextBox Name="mainTextBox"/>
</StackPanel>
```

```
// Creating the UI objects
StackPanel mainStackPanel = new StackPanel();
TextBox pasteTextBox = new TextBox();
Menu stackPanelMenu = new Menu();
MenuItem pasteMenuItem = new MenuItem();

// Adding objects to the panel and the menu
stackPanelMenu.Items.Add(pasteMenuItem);
mainStackPanel.Children.Add(stackPanelMenu);
mainStackPanel.Children.Add(pasteTextBox);

// Setting the command to the Paste command
pasteMenuItem.Command = ApplicationCommands.Paste;

// Setting the command target to the TextBox
pasteMenuItem.CommandTarget = pasteTextBox;
```

Komande - *RoutedCommand*

- Kreiranje *RoutedCommand-e*:
  - *Definisanje komande i njeno instanciranje*

```
public static RoutedCommand CustomRoutedCommand = new RoutedCommand();
```

  - *Da bi se koristila komanda potrebno je kreirati i obrađivače događaja koji definišu šta komanda radi.*

```
private void ExecutedCustomCommand(object sender,
    ExecutedRoutedEventArgs e)
{
    MessageBox.Show("Custom Command Executed");
}
```

```
// CanExecuteRoutedEventHandler that only returns true if
// the source is a control.
private void CanExecuteCustomCommand(object sender,
    CanExecuteRoutedEventArgs e)
{
    Control target = e.Source as Control;

    if(target != null)
    {
        e.CanExecute = true;
    }
    else
    {
        e.CanExecute = false;
    }
}
```

Komande - *RoutedCommand*

- *Kreiranje CommandBinding objekta koji povezuje naredbu sa obrađivačima događaja*
  - *Kreira se na određenom elementu i on definiše opseg CommandBinding-a u stablu elemenata*

```
<Window x:Class="SDKSamples.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:custom="clr-namespace:SDKSamples"
    Height="600" Width="800"
    >
  <Window.CommandBindings>
    <CommandBinding Command="{x:Static custom:Window1.CustomRoutedCommand}"
                    Executed="ExecutedCustomCommand"
                    CanExecute="CanExecuteCustomCommand" />
  </Window.CommandBindings>
```

```
CommandBinding customCommandBinding = new CommandBinding(
    CustomRoutedCommand, ExecutedCustomCommand, CanExecuteCustomCommand);

// attach CommandBinding to root window
this.CommandBindings.Add(customCommandBinding);
```

- *Pozivanje naredbe*

```
<StackPanel>
  <Button Command="{x:Static custom:Window1.CustomRoutedCommand}"
          Content="CustomRoutedCommand"/>
</StackPanel>
```

```
// create the ui
StackPanel CustomCommandStackPanel = new StackPanel();
Button CustomCommandButton = new Button();
CustomCommandStackPanel.Children.Add(CustomCommandButton);

CustomCommandButton.Command = CustomRoutedCommand;
```

Komande - *RoutedUICommand*

- Za razliku od *RoutedCommand* sadrži i dodatni atribut *Text* koji predstavlja tekst koji će prikazati korisniku.
- Obično se koristi kada za neku komandu imamo i prečicu.
- Postupak za kreiranje je isti kao i kod *RoutedCommand-i*

Komande - *RoutedUICommand*

- Kreiranje *RoutedUICommand-e:*
  - *Definisanje komande i njeno instanciranje*
    - *Da bi se koristila komanda potrebno je kreirati i obrađivače događaja koji definišu šta komanda radi.*
  - *Kreiranje CommandBinding objekta koji povezuje naredbu sa obrađivačima događaja*
    - *Kreira se na određenom elementu i on definiše opseg CommandBinding-a u stablu elemenata*
  - *Pozivanje naredbe*

Komande – ICommand

- Implementacija interfejsa *ICommand* je još jedan način za kreiranje komande.
- Kreiranje komande:
  - Napraviti klasu koja će naslediti *ICommand* interfejs
    - Potrebno je da se kreiraju metode *Execute* i *CanExecute* i događaj *CanExecuteChanged*
    - *Napravljena klasa predstavlja komandu za višekratnu upotrebu koja sadrži sopstvenu funkcionalnost i znanje kada može da se izvrši.*
  - Obezbediti način da se komanda pozove
    - Može da se izloži kao atribut neke klase i da je tako koristimo.
  - Povezati komandu pomoću CommandBinding-a sa izvorom

Stilovi

- Odnosi se na skup funkcija koje programerima i dizajnerima omogućavaju da naprave vizuelno ubedljive efekte i dosledan izgled za svoj program.
- Može da se zamisli kao pogodan način primene skupa vrednosti atributa na više elemenata. Stil može da se koristi na bilo kom elementu koji potiče od *FrameworkElement* ili *FrameworkContentElement* klase.
- Najčešći način za deklarisanje stila je kao resurs u okviru *XAML* fajla. Pošto su to resursi onda oni poštuju ista pravila opsega koja važe i za sve resurse.
- Mesto deklaracije stila utiče na to gde se stil može primeniti.

Stilovi – XAML

```
<Application x:Class="IntroToStylingAndTemplating.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="clr-namespace:IntroToStylingAndTemplating"
             StartupUri="WindowExplicitStyle.xaml">
    <Application.Resources>
        <ResourceDictionary>

            <Style x:Key="Header1" TargetType="TextBlock">
                <Setter Property="FontSize" Value="15" />
                <Setter Property="FontWeight" Value="ExtraBold" />
            </Style>

        </ResourceDictionary>
    </Application.Resources>
</Application>
```

TargetType određuje na koji UI element se primenjuje stil

Setter elementi postavljaju atribute, *Property* određuje koje svojstvo se postavlja, a *Value* vrednost tog svojstva

Stilovi - XAML

- Nasleđivanje stila

```
<Window.Resources>
    <!-- .... other resources .... -->

    <!--A Style that affects all TextBlocks-->
    <Style TargetType="TextBlock">
        <Setter Property="HorizontalAlignment" Value="Center" />
        <Setter Property="FontFamily" Value="Comic Sans MS"/>
        <Setter Property="FontSize" Value="14"/>
    </Style>

    <!--A Style that extends the previous TextBlock Style with an x:Key of TitleText-->
    <Style BasedOn="{StaticResource {x:Type TextBlock}}"
           TargetType="TextBlock"
           x:Key="TitleText">
        <Setter Property="FontSize" Value="26"/>
        <Setter Property="Foreground">
            <Setter.Value>
                <LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">
                    <LinearGradientBrush.GradientStops>
                        <GradientStop Offset="0.0" Color="#90DDDD" />
                        <GradientStop Offset="1.0" Color="#5BFFFF" />
                    </LinearGradientBrush.GradientStops>
                </LinearGradientBrush>
            </Setter.Value>
        </Setter>
    </Style>
</Window.Resources>
```

Definiše koji stil smo nasledili

Stil koji se primenjuje na *TextBlock* i koji će se dalje nasleđivati

Stil koji nasleđuje prethodni i dodaje još neke atribute

Stilovi

- Primena stilova:
  - Eksplicitna
    - Ako se za stil definiše atribut *x:Key* onda se stil neće implicitno primeniti na sve elemente. Primeniće se samo na elemente koji eksplicitno upućuju na stil.
  - Implicitna
    - Ako ne navedemo *x:Key* atribut onda će se stil primeniti na sve elemente definisane pomoću atributa *TargetType*.
  - Programski
    - Pisanjem C# koda
- Jednom kada se stil primeni, eksplicitno ili implicitno, on postaje zapečaćen i ne može se promeniti. Ako želite da promenite stil koji je primenjen, napravite novi stil koji će zameniti postojeći.

Stilovi – eksplicitna primena

```
<Window.Resources>
    <Style x:Key="TitleText" TargetType="TextBlock">
        <Setter Property="HorizontalAlignment" Value="Center" />
        <Setter Property="FontFamily" Value="Comic Sans MS"/>
        <Setter Property="FontSize" Value="14"/>
    </Style>
</Window.Resources>
```

```
<StackPanel>
    <TextBlock Style="{StaticResource TitleText}">My Pictures</TextBlock>
    <TextBlock>Check out my new pictures!</TextBlock>
</StackPanel>
```

Primer: prvo se definiše stil i za njega atribut x:Key, a zatim se stil pomoću atributa Style poveže sa određenim UI elementom.

Stilovi – implicitna primena

```
<Window.Resources>
    <!--A Style that affects all TextBlocks-->
    <Style TargetType="TextBlock">
        <Setter Property="HorizontalAlignment" Value="Center" />
        <Setter Property="FontFamily" Value="Comic Sans MS"/>
        <Setter Property="FontSize" Value="14"/>
    </Style>
</Window.Resources>
```

```
<StackPanel>
    <TextBlock>My Pictures</TextBlock>
    <TextBlock>Check out my new pictures!</TextBlock>
</StackPanel>
```

Primer: potrebno je da samo definišemo stil koji u sebi neće imati definisan x:Key atribut.

Stilovi – programski

- Da bi se stil programski postavio na neki element potrebno je da stil preuzmete iz rečnika resursa i dodelite ga atributu *Style* elementa.

```
<Window.Resources>
    <Style x:Key="TitleText" TargetType="TextBlock">
        <Setter Property="HorizontalAlignment" Value="Center" />
        <Setter Property="FontFamily" Value="Comic Sans MS"/>
        <Setter Property="FontSize" Value="14"/>
    </Style>
</Window.Resources>
```

```
textblock1.Style = (Style)Resources["TitleText"];
```

Stilovi - *Trigger*

- Postavljaju vrednosti atributa ili započinju akciju na osnovu vrednosti atributa.

```
<Window.Resources>
    <!-- .... other resources .... -->

    <Style TargetType="ListBoxItem">
        <Setter Property="Opacity" Value="0.5" />
        <Setter Property="MaxHeight" Value="75" />
        <Style.Triggers>
            <Trigger Property="IsSelected" Value="True">
                <Trigger.Setters>
                    <Setter Property="Opacity" Value="1.0" />
                </Trigger.Setters>
            </Trigger>
        </Style.Triggers>
    </Style>
</Window.Resources>
```

Stilovi - *EventTrigger*

- Na osnovu pojave događaja pokreće skup akcija

- Primer: objekti *EveneTrigger* u primeru navode da kada pokazivač miša pređe preko *ListBoxItem-a*, atribut *MaxHeight* animira se na vrednost 90 tokom 2 sekunde. Kada se miš odmakne od stavke onda se vraća na početnu vrednost u roku od 1 sekunde.

```
<Style.Triggers>
    <Trigger Property="IsSelected" Value="True">
        <Trigger.Setters>
            <Setter Property="Opacity" Value="1.0" />
        </Trigger.Setters>
    </Trigger>
    <EventTrigger RoutedEvent="Mouse.MouseEnter">
        <EventTrigger.Actions>
            <BeginStoryboard>
                <Storyboard>
                    <DoubleAnimation
                        Duration="0:0:0.2"
                        Storyboard.TargetProperty="MaxHeight"
                        To="90"  />
                </Storyboard>
            </BeginStoryboard>
        </EventTrigger.Actions>
    </EventTrigger>
    <EventTrigger RoutedEvent="Mouse.MouseLeave">
        <EventTrigger.Actions>
            <BeginStoryboard>
                <Storyboard>
                    <DoubleAnimation
                        Duration="0:0:1"
                        Storyboard.TargetProperty="MaxHeight"  />
                </Storyboard>
            </BeginStoryboard>
        </EventTrigger.Actions>
    </EventTrigger>
</Style.Triggers>
```