Module 3
Ovládání a reakce na uživatele v UI

MGR. TOMÁŠ HAVETTA - MCT

Obsah modulu

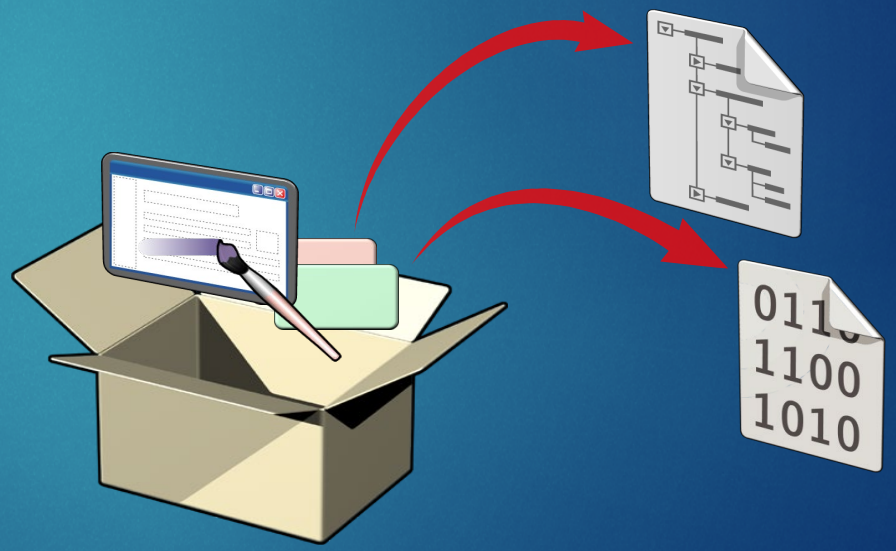
- WPF Resources
- Styly
- Control Template
- Routed Events
- Commands

Blok 1: WPF Resources

- Co to je?
- Vytvoření WPF Resources
- Referencování v XAML
- Referencování v kódu
- Knihovny WPF Resources

Co to je?

- WPF Resources definují elementy, které můžete v aplikaci opakovaně používat
- Typicky se jedná o
  - Barvy
  - Štětce a pera
  - Konvertory
  - Pomocné objekty



Vytvoření WPF Resources

- Kterýkoliv WPF element může definovat Resources
- Typicky se definují na úrovni okna nebo aplikace
- WPF Resources definují Dictionary<string, object>
  - Klíč Dictionary je definován atributem x:Key

```
<Window.Resources>
  <SolidColorBrush x:Key="blueBrush" Color="Blue"/>
  <SolidColorBrush x:Key="whiteBrush" Color="White"/>
  <sys:Double x:Key="myValue">100</sys:Double>
</Window.Resources>
```

Referencování v XAML

- StaticResource => vazba přímo na objekt
- DynamicResource => vazba přes klíč Dictionary

```
<Button Background="{StaticResource blueBrush}"
        Foreground="{DynamicResource whiteBrush}">
  OK
</Button>
```

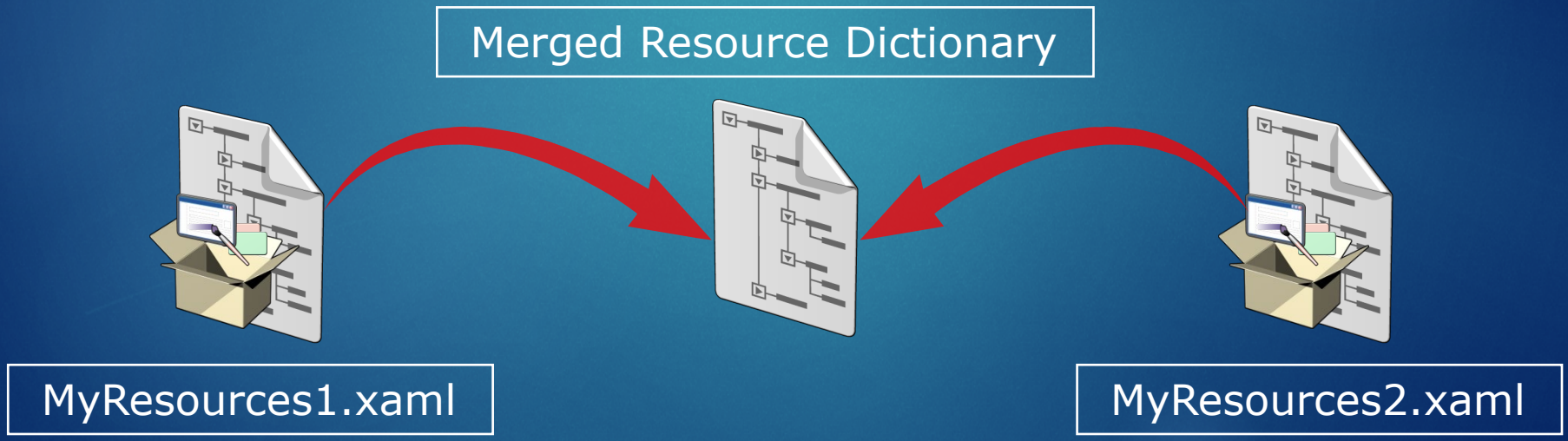
Referencování v kódu

- **FindResorce** a **TryFindResorce**
  - Hledá podle klíče postupně ve stromě směrem k aplikační úrovni
- **SetResourceReference**
  - Odpovídá DynamicResource v XAML
- Přes vlastnost **Resources**
  - Programátor musí přesně znát místo deklarace

```
SolidColorBrush brush = (SolidColorBrush)
  this.Resources["whiteBrush"];
```

Knihovny WPF Resources

```
<Page.Resources>
  <ResourceDictionary.MergedDictionaries>
    <ResourceDictionary
      Source="Resources\MyResources1.xaml" />
    <ResourceDictionary
      Source="Resources\MyResources2.xaml" />
  </ResourceDictionary.MergedDictionaries>
</Page.Resources>
```



Blok 2: Styly

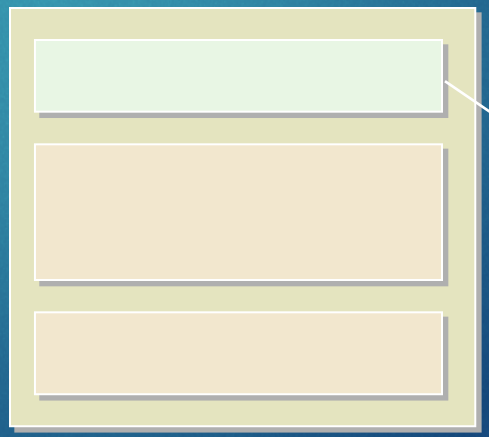
- WPF Styly
- Definování stylu v XAML
- Dědění stylů
- Nastavení stylu v kódu

WPF styly

- Pomocí stylů (podobně jak v CSS) nastavujete vlastnosti controlů pro dasažení stejného vzhledu či chování
- Styly se typicky definují v XAML WPF Resources

<Resources />

Style



Control

Definování stylu v XAML

- Definovat **Style** element v Resources
- Vlastnost **TargetType** určí na jaký typ je styl určen
- **Setter** element definuje dvojici **Property** a **Value**

```
<Page.Resources>
  <Style x:Key="myStyle" TargetType="{x:Type Label}">
    <Setter Property="Background" Value="Blue" />
    <Setter Property="Foreground" Value="White" />
  </Style>
</Page.Resources>
```

Dědění stylů

- Pomocí vlastnosti **BasedOn**

```
<Page.Resources>
  ...
  <Style x:Key="headerText"
         BasedOn="{StaticResource myStyle}"
         TargetType="{x:Type Label}">
    ...
  </Style>
</Page.Resources>
...
<StackPanel>
 <Label Content="Title Text"
         Style="{StaticResource headerText}" />
 <Label Content="Hello world"
         Style="{StaticResource myStyle}" />
```

Nastavení stylu v kódu

- Style je typ jako jakýkoliv jiný objekt
- Lze ho vytvořit v kódu, ale častěji se získává z Resources kolekce
- Nastavit vlastnost Style na požadovaném prvku

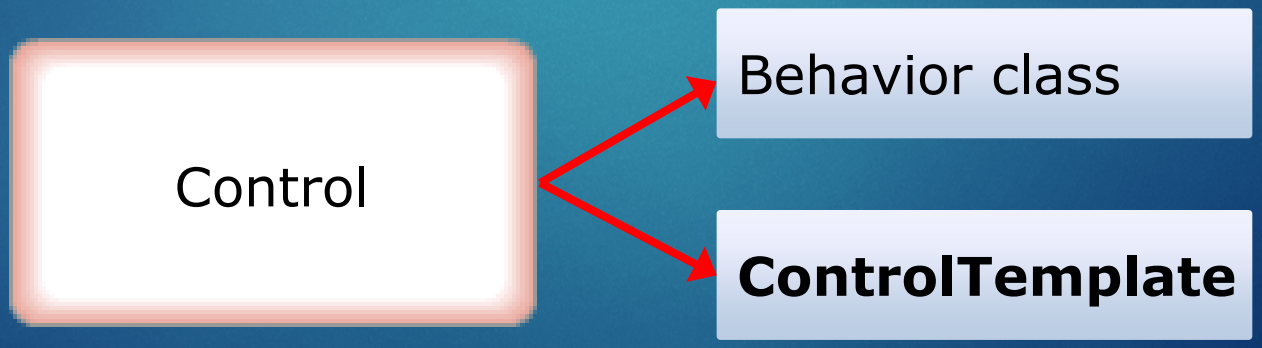
```
textblock1.Style = (Style)(Resources["TitleText"]);
```

Blok 3: Control Template

- Princip
- Control Template pro Content Control
- Control Template pro Items Control
- Template Binding
- Demo

Princip

- Každý Control má zadefinováno
  - Chování => naprogramováno co má dělat
  - Vzhled => definuje vzhled controlu, nezávisle na chování
- Control Template => určuje vzhled prvku



Control Template pro Content Control

- Vytvořit **Styl** pro daný **TargetType**
- Udělat **Setter** pro vlastnost **Template**
- Vytvořit **ControlTemplate**
- Pomocí **ContentPresenter** určit polohu a vzhled obsahu

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

Control Template pro Items Control

- IsItemsHost
  - Identifikuje panel, který bude obsahovat itemy
- ItemsPanelTemplate
  - Určuje panel definující umístění itemů
- ItemsPresenter
  - Určuje, kde bude umístěn ItemsPanelTemplate

Template Binding

- Umožňuje použít v ControlTemplate hodnotu vlastnosti Controlu

```
<Style TargetType="ListBox">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="ListBox">
        <Border Background="{TemplateBinding ListBox.Background}"
                CornerRadius="5">
          ...
        </Border>
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
```

Demo

Blok 4: Routed Events

- WPF a události
- Zpracování události
- Základy Routed Eventů
- Zpracování Routed Eventů

WPF a události

- WPF controls generují standardní údálosti
  - Kliknutí
  - Zadávání textu
  - Výběr prvku v seznamu
  - Získání focusu
  - …

Zpracování události

- V XAML je definován název metody, která bude zavolaná v případě, že událost nastane

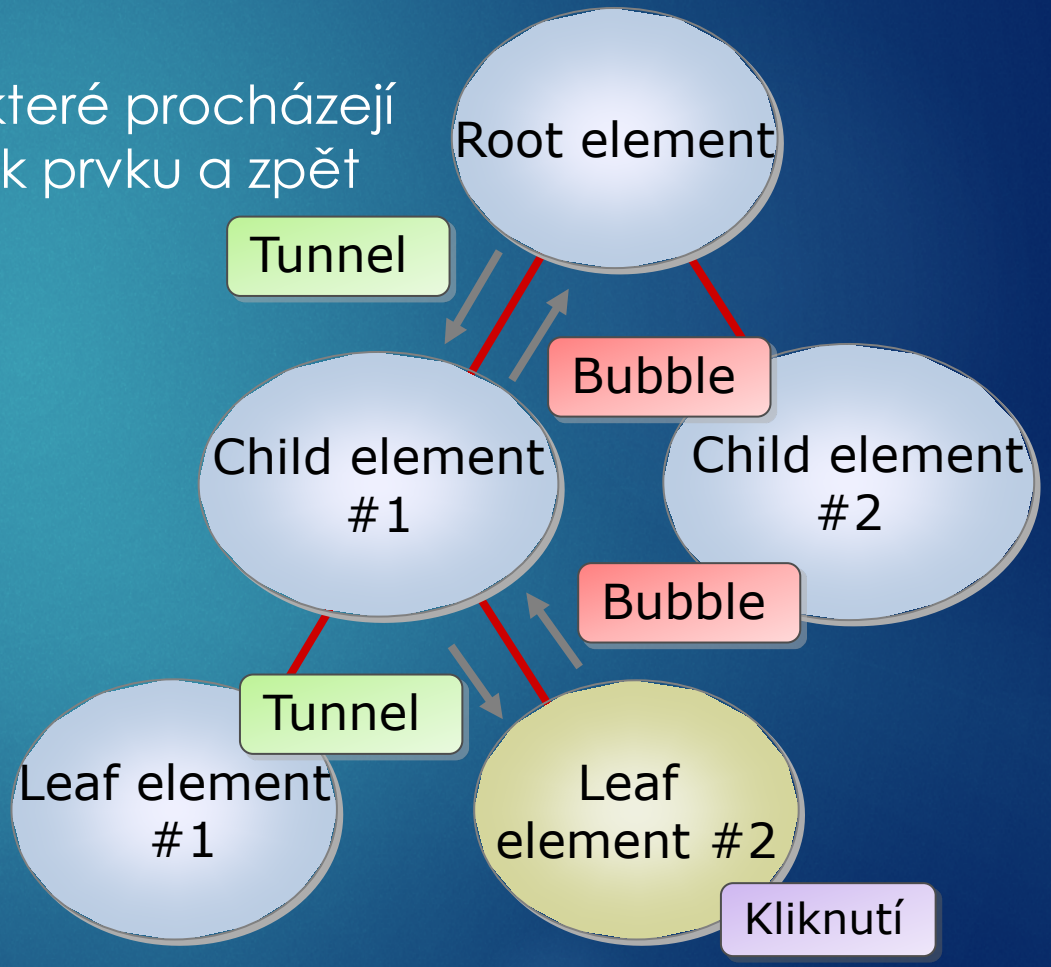
```
<Button Name="Button1" Click="Button1_Click">
  Click here
</Button>
```

- Implementace je v code-behind souboru

```
public void Button1_Click(
   object sender, RoutedEventArgs e)
{
    MessageBox.Show("Hello WPF");
}
```

Základy Routed Eventů

- Speciální WPF eventy, které procházejí WPF stromem od rootu k prvku a zpět
- Event tunneling
  - směrem dolů
- Event bubbling
  - směr nahoru



Zpracování Routed Eventů

- Routed Eventy lze zpracovat na kterékoliv úrovni WPF stromu
- **RoutedEventsArgs** obsahuje informace o původci události

```
<StackPanel Button.Click="CommonClickHandler">
  <Button Name="YesButton">Yes</Button>
  <Button Name="NoButton">No</Button>
</StackPanel>
```

```
private void CommonClickHandler(object sender,
                                RoutedEventArgs e)
{
  var b = e.Source as Button;
  ...
}
```

Blok 5: Commands

- Commands
- Systémové Commands
- Použití Commands v XAML

Commands

- Loosely coupled verze událostí (ideál pro MVVM)
- Event handler je pevně svázán s UI
- Command může být vytvořen mimo UI a s UI svázán pomocí vlastnosti **Command**
- Command implementuje **ICommand** interface
  - **void Execute(object param)**
  - **bool CanExecute(object param)**
  - **event EventHandler? CanExecuteChanged**

Systémové Commands

- ApplicationCommands
  - Close, Copy, Cut, Delete, Find, Help, Save, Undo, …
- ComponentCommands
  - MoveDown, MoveLeft, ScrollPageDown, SelectToEnd, …
- MediaCommands
  - ChannelDown, ChannelUp, MuteVolume, Pause, Play, …
- EditingCommands
  - AlignCenter, IncreaseFontSize, MoveDownByLine, …
- SystemCommand
  - CloseWindow, MaximizeWindow, ShowSystemMenu, …

Použití Commands v XAML

```
<Window ...>
    <Window.CommandBindings>
        <CommandBinding
            Command="Close"
            CanExecute="OnCloseCanExecute"
            Executed="OnCloseExecuted"/>
    </Window.CommandBindings>
    <Button Command="Close" Content="Close Application" />
</Window>
```

```
private void OnCloseCanExecute(object sender,
CanExecuteRoutedEventArgs e)
{
    e.CanExecute = true;
}
private void OnCloseExecuted(object sender,
ExecutedRoutedEventArgs e)
{
    this.Close();
}
```

Lab: Vylepšení aplikace převodník

- Cvičení 1:
  - Použijte styly pro TextBox, Label a Button
  - Vytvořte jednu metodu pro zpracování kliknutí na libovolné tlačítko
- Cvičení 2:
  - Předělejte aplikaci aby pracovala s Commandy pomocí CommandBindings