Windows Presentation Foundation

Michal Radecký
Marek Běhálek

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Má smysl se (na)učit WPF?

- GUI vývojový rámec pro *Windows* postavený na .NET (*Core*) Framework
  - Pokud použiju .NET Core, bude GUI fungovat všude?
    - Ne úplně, .NET Core je použit zejména kvůli rychlosti. Pořád jde primárně o GUI pro Windows.
    - Obecně je technologie aktuálně open source a existují implementace i pro jiné systémy (https://avaloniaui.net/).
  - Proč se učit technologii, která je primárně určená *jen pro desktopové* aplikace ve Windows?
    - Zajímavá je koncepce, kdy je oddělena definice vzhledu a logika. Tato koncepce je součástí mnoha technologií (Java FX, Qt, …).
    - Primární součástí je jazyk XAML (eXtensible Application Markup Language), který slouží k definici *vzhledu.* Tento je použit i v jiných technologií (Universal Windows Platform, Xamarin, Silverlight, Worflow Foudation) a počítá se s ním i do budoucna (.NET Multi-platform App UI - https://devblogs.microsoft.com/dotnet/introducing-net-multi-platform-app-ui/).

Alternativy k WPF pro desktopové aplikace?

- Starší technologie `Windows Forms`
  - V mnoha ohledech vhodná alternativa k WPF.
  - Poběží na stejných verzích Windows.
  - XAML přináší větší komfort – lépe odděluje logiku aplikace a její vzhled.
  - WPF je rychlejší.
- Univerzálnější řešení
  - UWP - v mnoha ohledech nástupce WPF
    - Omezení pouze na novější systémy Windows 10, na druhou stranu, širší podpora mobilních systému či Xbox One.
    - Pokud potřebuji složitější GUI pouze pro desktop, pořád je doporučovaným nástrojem WPF.
  - Blazor – či využití jiné webové technologie pro desktop.
    - Snadný přechod mezi desktopovou aplikací a webovou.
  - Xamarin, MAUI, …

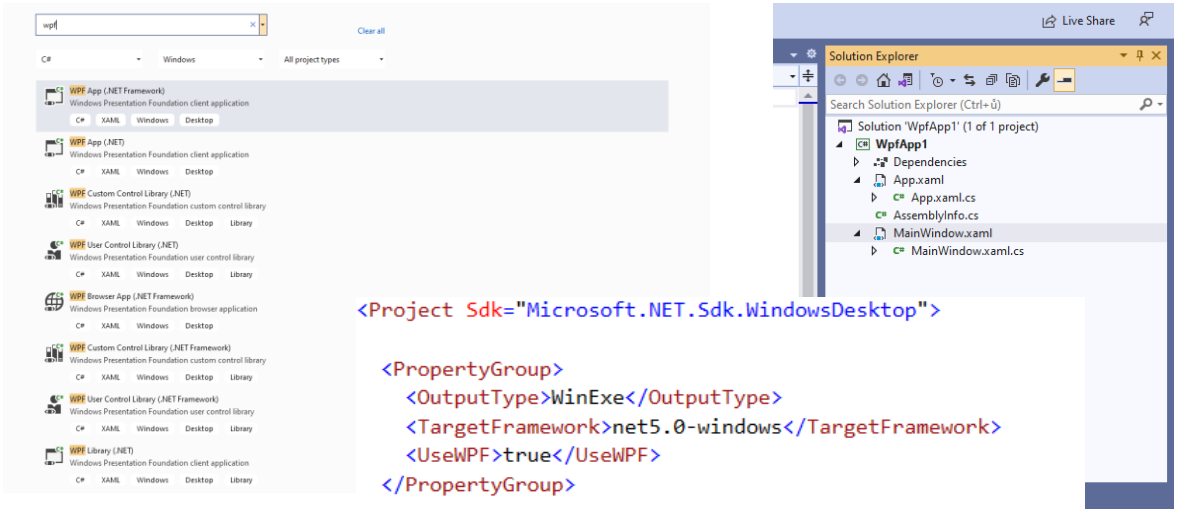
WPF - Základní vlastnosti (1)

- Jmenný prostor `System.Windows`
  - Podobné komponenty jako `Windows.Forms`
  - Navíc *dependency property, routed events.*
- Aplikace je možné vyvíjet jak pomocí značkovacího jazyka tak pomocí kódu. Obvyklé řešení je kombinace obou přístupů.
  - Značkovací jazyk XAML
  - *Code-behind* – kterýkoliv jazyk .NET platformy
  - Obvyklé je definovat vzhled (statické rozložení GUI) pomocí XAML a pak připojit definice chování v nějakém programovacím jazyce třeba C#.
  - Propojení dat a zobrazení – binding.

WPF - Základní vlastnosti (2)

- Základní typy projektů
  - Samostatná WPF aplikace – klasické desktopové řešení na .NET nebo .NET Core
  - XAML Browser Application (XBAPs) – stránky jsou spustitelné assembly, které lze *zobrazit* ve webovém prohlížeči.
  - Custom Control Libraries
  - Class Libraries
- Kromě základní tvorby GUI podporuje: komponenty na *kreslení*, animace, podpora médií, podpora *stylů*, lokalizace,…
- Součástí knihoven jsou i komponenty řešící další činnosti jako: management aplikace, správu zdrojů, podporu vývoje a nasazení aplikace.

WPF – První aplikace (1)



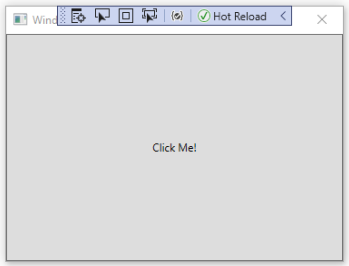

```xml
<Project Sdk="Microsoft.NET.Sdk.WindowsDesktop">

  <PropertyGroup>
    <OutputType>WinExe</OutputType>
    <TargetFramework>net5.0-windows</TargetFramework>
    <UseWPF>true</UseWPF>
  </PropertyGroup>

</Project>
```

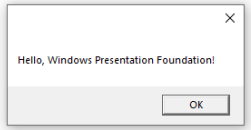
```xml
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.AWindow"
    Title="Window with Button"
    Width="250" Height="100">

  <!-- Add button to window -->
  <Button Name="button" Click="button_Click">Click Me!</Button>

</Window>
```

```cs
using System.Windows; // Window, RoutedEventArgs, MessageBox

namespace SDKSample
{
    public partial class AWindow : Window
    {
        public AWindow()
        {
            // InitializeComponent call is required to merge the UI
            // that is defined in markup with this class, including
            // setting properties and registering event handlers
            InitializeComponent();
        }

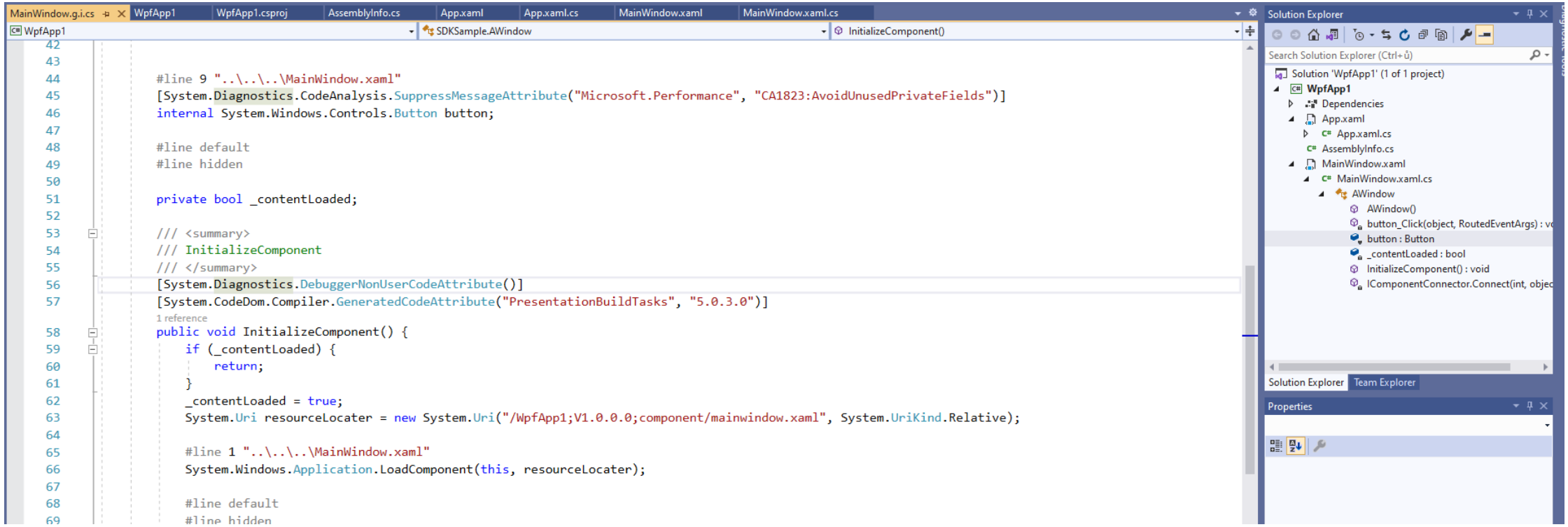
        void button_Click(object sender, RoutedEventArgs e)
        {
            // Show message box when button is clicked.
            MessageBox.Show("Hello, Windows Presentation Foundation!");
        }
    }
}
```





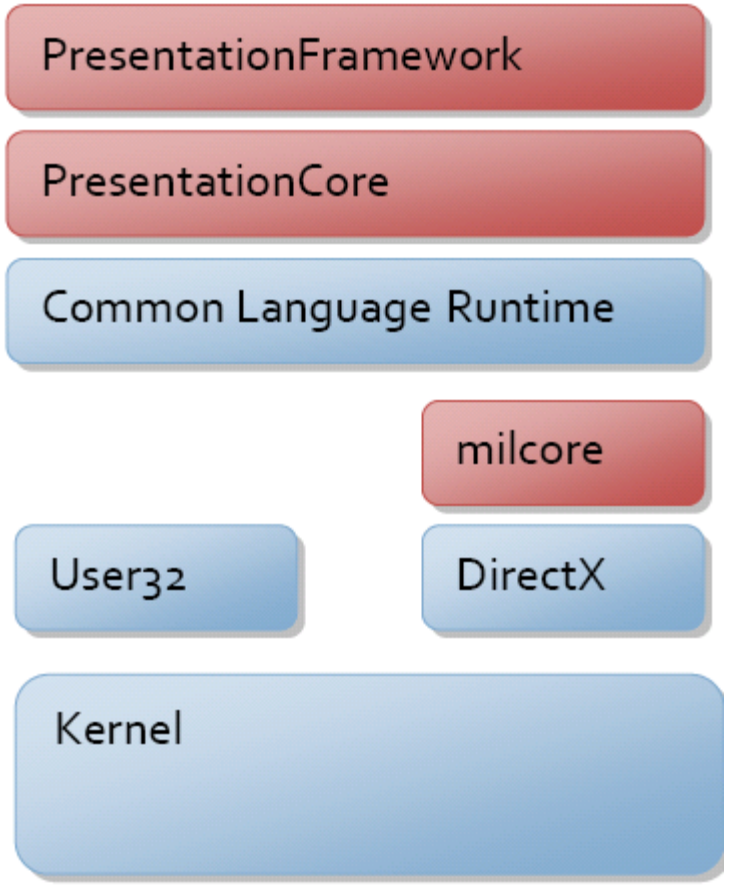
WPF – První aplikace (2)

- Interně je XAML převedeno na kód.
  - Podobný tomu, co bychom vytvořili technologií `Windows.Forms`.



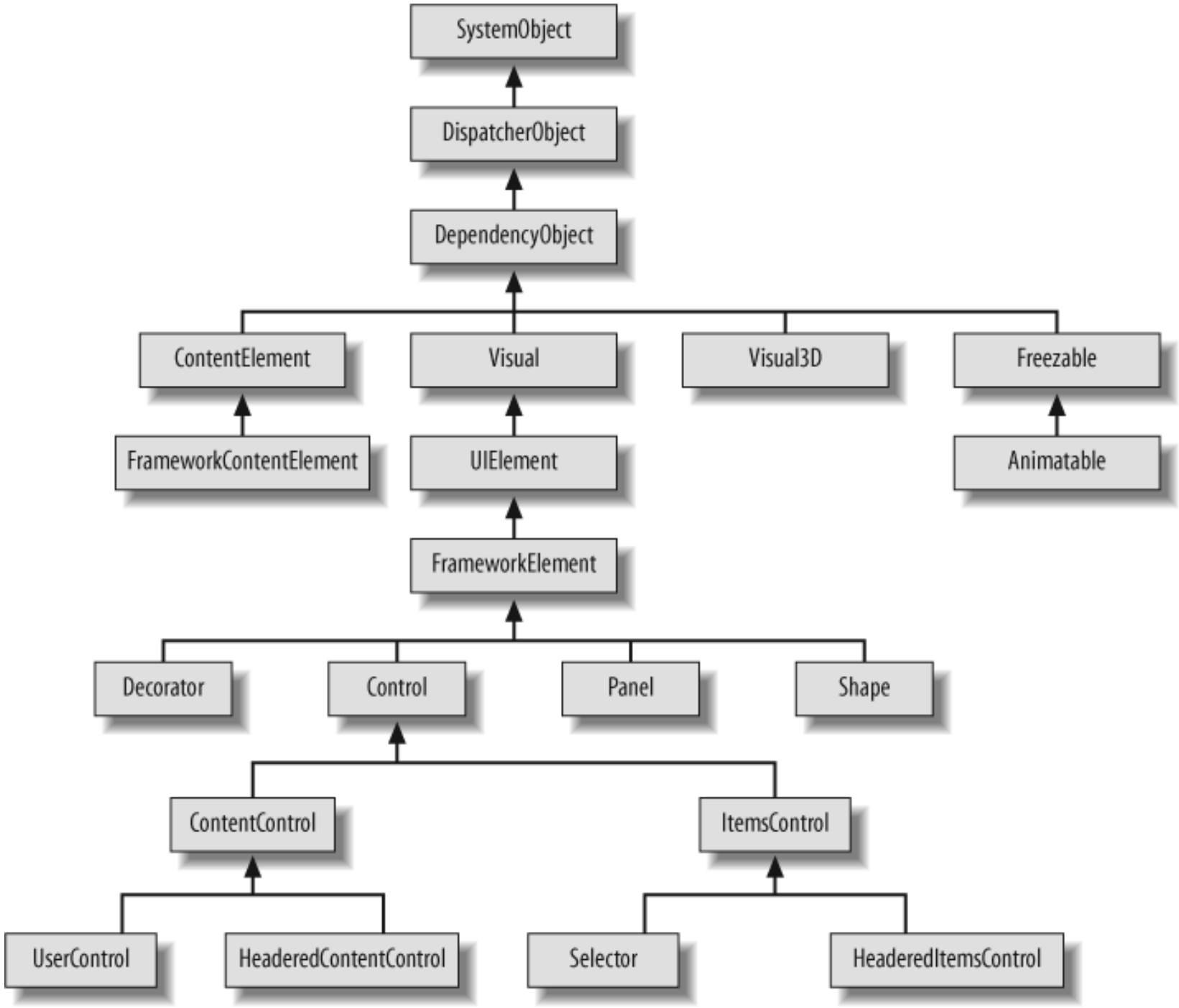
WPF – Architektura (1)

- WPF používá řízený kód.
- System.Threading.DispatcherObject
  - Základní třída pro většinu prvků.
  - Subsystém pro řízení vláken.
- System.Windows.DependencyObject
  - WPF *preferuje* vlastnosti před metodami a událostmi (GUI je vlastně *řízeno* daty).
- System.Windows.Media.Visual
  - Subsystém, který řídí co bude skutečně zobrazeno.
  - Tato funkcionalita je v mnoha ohledech skryta před *uživatelem*.



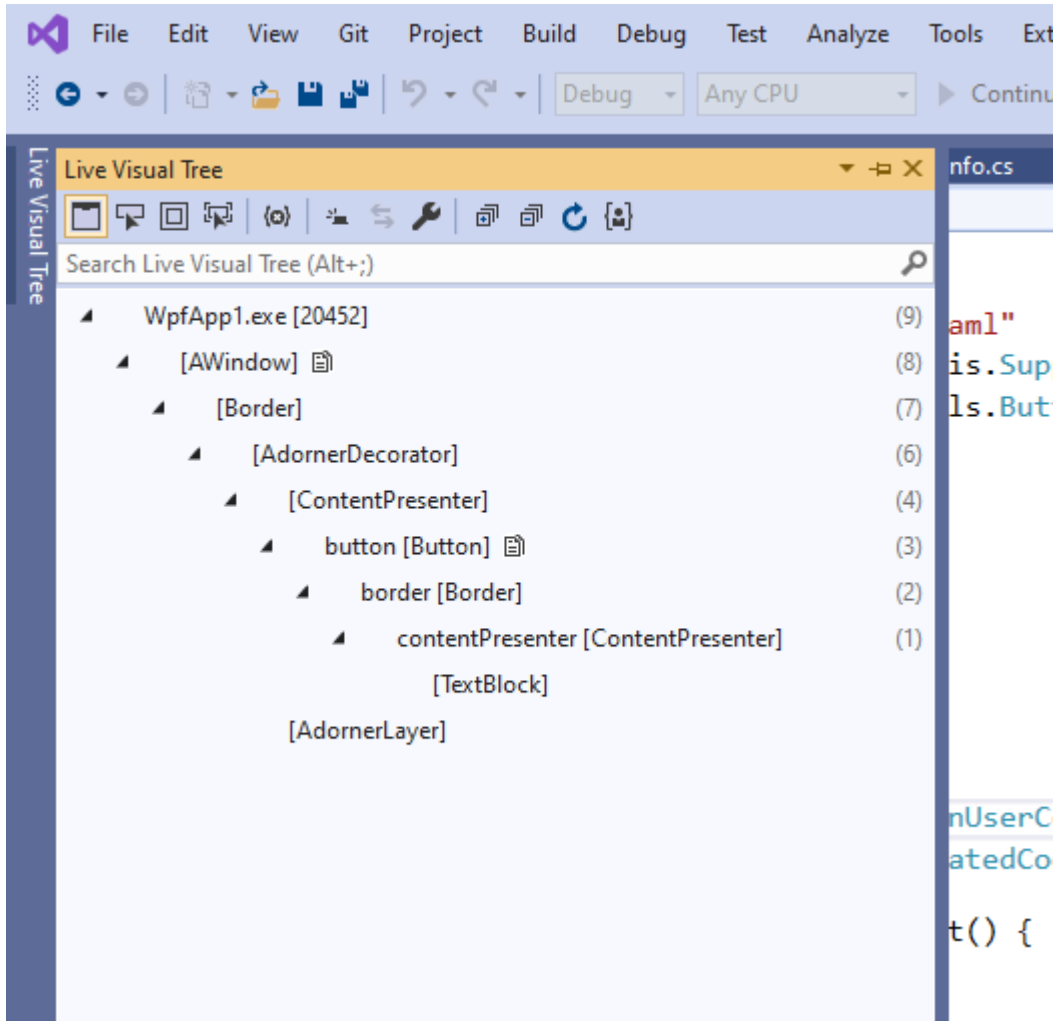
WPF – Architektura (2)

- Většina tříd ve WPF je odvozena z (předek obou je `DependencyObject`):
  - `System.Windows.`**`UIElement`** – dědí také z Visual, zjednodušeně řečeno vyprodukuje něco na obrazovku.
  - `System.Windows.`**`ContentElement`** – nedědí z Visual, očekává se, že bude použit jinými komponentami a poskytuje nějakou funkcionalitu.
- Core-level vs. Framework-level
  - `UIElement` a `ContentElement` jsou z *core* úrovně. Idea je, že na této úrovni není potřeba všechny knihovny. Prakticky ovšem moc oddělit tuto úroveň nejde. Typická aplikace používá třídy z *framework* úrovně.
  - `FrameworkElement` - `FrameworkContentElement`
- `System.Windows.Controls.Control` (dědí z `FrameworkElement`)
  - Systém poskytující vzory (*templates*) pro stylování komponent.
    - `ControlTemplate`
  - Základní třída pro řadu základních komponent (`Object → DispatcherObject → DependencyObject → Visual → UIElement → FrameworkElement → Control → TextBoxBaseTextBox`).


SystemObject
DispatcherObject
DependencyObject
ContentElement
Visual
Visual3D
Freezable
FrameworkContentElement
UIElement
Animatable
FrameworkElement
Decorator
Control
Panel
Shape
ContentControl
ItemsControl
UserControl
HeaderedContentControl
Selector
HeaderedItemsControl

WPF – Architektura (3)

- Struktura GUI je v principu strom.
- Ve WPF existují různé typy *stromů*.
  - Logický strom - `LogicalTreeHelper`
    - Definován vazbami mezi objekty. Objekty mají vazbu na rodiče a potomky.
  - Visuální strom - `VisualTreeHelper`
    - Závisí na zobrazení (třída Visual)
  - Některé mechanismy, například `RoutedEvents`, používají jakýsi hybrid, mezi oběma reprezentacemi.



WPF – XAML - Základní syntaxe

- Syntaxe vychází z XML (dobře utvořené HTML, XHTML)
- <Button Name="CheckoutButton" Click="Button_Click"/>
- <TextBox>This is a Text Box</TextBox>
-
```
<Button>
        <Button.ContextMenu>
                <ContextMenu>
                        <MenuItem Header="1">First item</MenuItem>
                        <MenuItem Header="2">Second item</MenuItem>
                </ContextMenu>
        </Button.ContextMenu>
        Right-click me!
</Button>
```

WPF – XAML – Markup extensions

- Cokoliv v {}.
- `StaticResource, DynamicResource, Binding, RelativeSource, TemplateBiding, ColorConverterBitmap, ComponentResourceKey, ThemeDirectory,…`
- Nejčastější použití: *binding*, statické či dynamické zdroje.
-
```
<Page.Resources>
        <Style TargetType="Border" x:Key="PageBackground">
                <Setter Property="Background" Value="Blue"/>
        </Style>
</Page.Resources>
<StackPanel>
        <Border Style="{StaticResource PageBackground}"></Border>
</StackPanel>
```

- Je možné dělat vlastní rozšíření.

WPF – XAML – Příklady rozšíření

```
Foreground = "{StaticResource myBrush}"
```

```
Text = "{Binding ElementName=mainWindow, Path=Config.ServerUri,
UpdateSourceTrigger=PropertyChanged}"
```

```
ToolTip="{Binding ToolTipSource, UpdateSourceTrigger=PropertyChanged}"
```

```
Value="{Binding RelativeSource={x:Static RelativeSource.Self}}"
```

```
Background="{Binding Path=Background, RelativeSource={RelativeSource
Mode=FindAncestor,AncestorType={x:Type ListBoxItem}}}"
```

```
Source="{Binding Active, Converter={StaticResource boolConverter}}"
```

```
Color="{DynamicResource {x:Static SystemColors.DesktopColorKey}}"
```

WPF – XAML – Typová konverze

- WPF podporuje automatickou konverzi hodnot.
  - `TypeConverterAttribute` – referuje třídu implementující `TypeConverter`.
  - Může zohlednit například jazyk atd.

- `<Button Margin="10,20,10,30" Content="Click me"/>`

- ```
  <Button Content="Click me">
      <Button.Margin>
          <Thickness Left="10" Top="20" Right="10" Bottom="30"/>
      </Button.Margin>
  </Button>
  ```
  ```

WPF – XAML – Jmenné prostory

- Definice jmenného prostoru pro jazyk XAML

```
<Window
        xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
        xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
</Window>
```

  - Základní použití: `x:Key, x:Class, x:Name, x:Static, x:Type`
  - Další například `x:Code` (ale má různá omezení, není doporučováno).

- Uživatelem odkazované zdroje

```
xmlns:custom="clr-namespace:NumericUpDownCustomControl;assembly=CustomLibrary"
```

- Jmenný prostor není nutné někdy uvádět (podobně jako v normálním kódu).
  - Name referuje nejčastěji `x:Name`

WPF – XAML – Attached (připojene?) vlastnosti

- Některé atributy je možné použít v libovolném kontextu.

```
<DockPanel>
        <Button DockPanel.Dock="Left" Width="100" Height="20">
                I am on the left
        </Button>
        <Button DockPanel.Dock="Right" Width="100" Height="20">
                I am on the right
        </Button>
</DockPanel>
```

WPF – Dependency properties (1)

- Normální vlastnost, vnitřně referuje registrovanou vlastnost (dle konvencí) stejného jména.
- Řeší různé problém, primárně stejné hodnoty, například velikost fontu, v celém podstromu GUI.
- Oproti normální vlastnostem:
  - Mohou používat {} – nastavení pomocí *data binding, resource* (statický i dynamický), styly, animace,…
  - Dědičnost mezi vlastnostmi
    - Některé dědí hodnotu z předka (DataContext)
    - Není nutné znovu definovat vs. dopad na výkon
  - Automatická reakce na změnu nebo validace.
  - Podpora v nástrojích pro tvorbu GUI.
  - Metadata – můžeme například ovlivnit systém vykreslování.

```
public static readonly DependencyProperty IsSpinningProperty =
    DependencyProperty.Register(
    "IsSpinning", typeof(Boolean),
    typeof(MyCode)
    );
public bool IsSpinning
{
    get { return (bool)GetValue(IsSpinningProperty); }
    set { SetValue(IsSpinningProperty, value); }
}
```

WPF – Dependency properties

- Při vyhodnocování, jakou hodnotu použít se tuto vlastnost vlastně řídí *prioritou* (postupně procházím jednotlivé úrovně priority, případně uzly ve stromě).
- Priorita:
  - *Property system coercion* (uživatel může vynutit přes: CoerceValueCallback)
  - Animace
  - Lokální hodnota
  - Vzor definovaný v `TemplatedParent` (`ControlTemplate` nebo `DataTemplate`)
  - Implicitní styl (vlastnost `Style`)
  - *Trigger* pro styl
  - *Trigger* ze vzoru
  - Hodnota z položky `Setter` z aktuálního okna (stránky) nebo aplikace
  - Aktuální styl (téma) celé aplikace
  - Dědičnost
  - Defaultní hodnota vlastnosti

```
    <Button Background="Red">
      <Button.Style>
        <Style TargetType="{x:Type Button}">
          <Setter Property="Background" Value="Green"/>
          <Style.Triggers>
            <Trigger Property="IsMouseOver" Value="True">
              <Setter Property="Background" Value="Blue" />
            </Trigger>
          </Style.Triggers>
        </Style>
      </Button.Style>
Click
    </Button>
```

WPF – Routed Events

- WPF používá systém řízených událostí.
  - Nevíme, kdo bude přesně konzumovat událost.
  - Co když kliknu na obrázek v tlačítku. Obvykle chci spustit obsluhu události `Click` tlačítka.
- Typy Routed Event
  - *Bubbling* – událost postupně prochází od zdroje do kořene
  - *Direct* – událost zpracuje jen zdroj události
  - *Tunneling* – událost první vyvolá element v kořenu, pak jsou postupně procházeny elementy až do zdroje události.

```
void Onb2Click(object sender, RoutedEventArgs e)
{
    var a = e.RoutedEvent.RoutingStrategy;
    e.Handled = true;
}
```

```
public static readonly RoutedEvent TapEvent = EventManager.RegisterRoutedEvent(
    "Tap", RoutingStrategy.Bubble, typeof(RoutedEventHandler), typeof(MyButtonSimple));

// Provide CLR accessors for the event
public event RoutedEventHandler Tap
{
        add { AddHandler(TapEvent, value); }
        remove { RemoveHandler(TapEvent, value); }
}
```

WPF – Vlákna

- WPF aplikace má dvě vlákna, jedno vlákno které se stará o renderování, druhé, které řeší obsluhu UI komponent – **`Dispatcher.`**
- Dispatcher řeší frontu požadavků od jednotlivých komponent, kde na základě *vlastních* pravidel je vybírá a řeší.
  - Toto vlákno řeší i obsluhu událostí (není dobrý nápad provádět zde složitý výpočet).
  - Toto vlákno je jediné, které řeší změny ve stromě komponent.
  - Pokud potřebuju použít více vláken, musím změny v GUI řešit přes něj.

```
Application.Current.Dispatcher.BeginInvoke(
        DispatcherPriority.Background,
        new Action(() => {
                this.progressBar.Value = 50;
        }));
```

WPF – Základní komponenty (1)

- Návrh rozložení komponent (Layout)
  - `Border, BulletDecorator, Canvas, DockPanel, Expander, Grid, GridSplitter, GroupBox, Panel, ResizeGrip, Separator, ScrollBar, ScrollViewer, StackPanel, Thumb, Viewbox, VirtualizingStackPanel, Window, WrapPanel`
- Tlačítka – `Button, RepeatButton`
- Zobrazení *skupiny* dat – `DataGrid, ListView, TreeView`
- Zobrazení data – `Calendar, DatePicker`
- Menu – `ContextMenu, Menu, ToolBar`
- Výběr – `CheckBox, ComboBox, ListBox, RadioButton, Slider`
- Navigace – `Frame, Hyperlink, Page, NavigationWindow, TabControl`
- Zobrazení informací (uživatel nemůže typicky měnit tyto komponenty)
  - `AccessText, Label, Popup, ProgressBar, StatusBar, TextBlock, ToolTip`
- Vstup – `TextBox, RichTextBox, PasswordBox`
- Media – `Image, MediaElement, SoundPlayerAction`
- Další komponety…
  - `InkCanvas, InkPresenter, DocumentViewer, FlowDocumentPageViewer, FlowDocumentReader, FlowDocumentScrollViewer, StickyNoteControl`

WPF – Základní komponenty (2)

- **Příklad různých komponent:**

  https://github.com/microsoft/WPF-Samples

```
<Button FontSize="14" FontWeight="Bold">
    View message
</Button>
```

- Nastavení stylu komponenty
  - Přímé nastavení vlastností
  - Vlastnost `Style`
  - Nastavení `ControlTemplate`

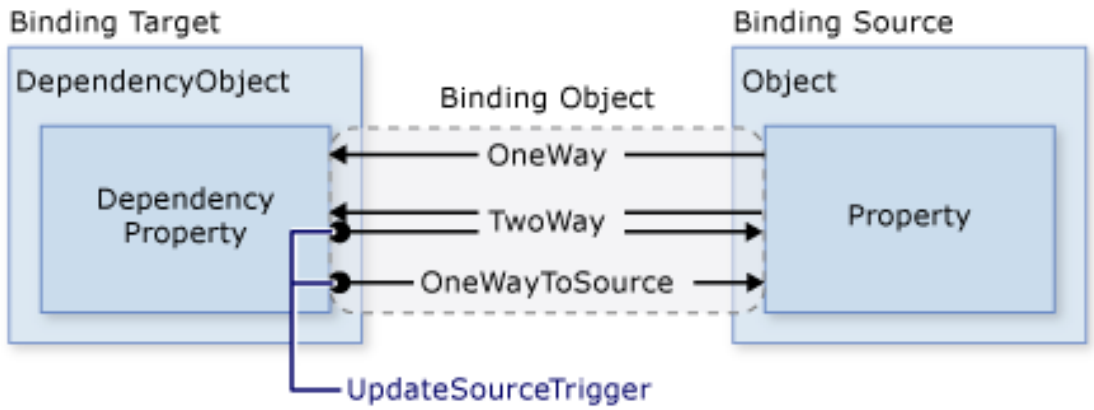
```
<Style TargetType="Button">
  <Setter Property="FontSize" Value="14"/>
  <Setter Property="FontWeight" Value="Bold"/>
  <Setter Property="Background">
    <Setter.Value>
      <LinearGradientBrush StartPoint="0,0.5"
                           EndPoint="1,0.5">
        <GradientStop Color="Green" Offset="0.0" />
        <GradientStop Color="White" Offset="0.9" />
      </LinearGradientBrush>

    </Setter.Value>
  </Setter>
</Style>
```

- Vzhled může být nastavený na různých úrovních
  - Přímé nastavení korespondující vlastnosti.
  - Přiřazení pojmenovaného zdroje - {StaticResource}
  - Předefinování stylu na *lokální – globální* úrovni.

WPF – Propojení s daty (data binding) (1)

- Reprezentuje propojení dat a GUI aplikace.
  1. Základní myšlenka je, staticky definuji GUI a projím jej s daty.
  2. GUI pak může reagovat na změnu dat.
  3. Úprava data v GUI ovlivní datové objekty.
- Mechanismus pro propojení dat pak poskytuje další funkce, například validaci dat.
- Možné scénáře propojení: *jednosměrné, dvojsměrné nebo jednosměrné do zdroje*.
- Aktualizace dat může být (UpdateSourceTrigger):
  - LostFocus
  - PropertyChanged
  - Explicit (aplikace vyvolá UpdateSource)



WPF – Propojení s daty (data binding) (2)

- Defaultní zdroj pro data je vlastnost `DataContext`
  - Pokud není definovaná, je tato vlastnost je děděná z rodiče.
- Můžeme specifikovat zdroj
  - Pak používáme atribut `Path`
  - Zdrojem může být nějaký zdroj (`Source`) či jiný objekt (`Name`)

```
<DockPanel xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
           xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
           xmlns:c="clr-namespace:SDKSample">
    <DockPanel.Resources>
        <c:MyData x:Key="myDataSource"/>
    </DockPanel.Resources>
    <DockPanel.DataContext>
        <Binding Source="{StaticResource myDataSource}"/>
    </DockPanel.DataContext>
    <Button Background="{Binding Path=ColorName}"
            Width="150" Height="30">
        I am bound to be RED!
    </Button>
</DockPanel>
```

```
<ListBox ItemsSource="{Binding}"
         IsSynchronizedWithCurrentItem="true"/>
```

```
<DockPanel xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
           xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
           xmlns:c="clr-namespace:SDKSample">
    <DockPanel.Resources>
        <c:MyData x:Key="myDataSource"/>
    </DockPanel.Resources>
    <Button Background="{Binding Source={StaticResource myDataSource}, Path=ColorName}"
            Width="150" Height="30">
        I am bound to be RED!
    </Button>
</DockPanel>
```

WPF – Propojení s daty (data binding) (3)

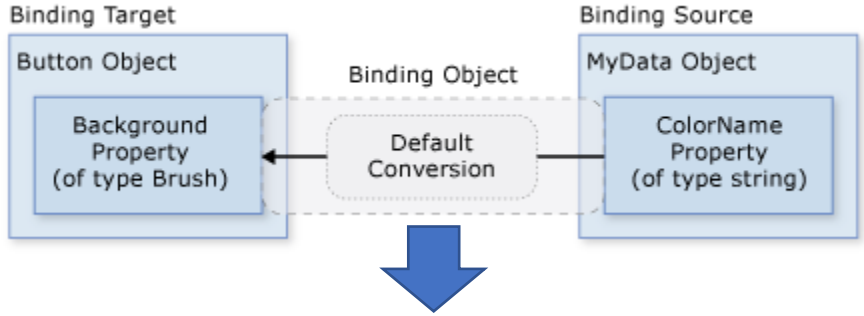
- Můžeme měnit propojení i v programu.
- Propojení je ovlivněno konverzi hodnot.
- Je-li použit celý objekt
  - jde použít vlastnosti, vlastnosti vlastností, indexery
  - rozhraní `INotifyPropertyChange`

```
// Make a new source
var myDataObject = new MyData();
var myBinding = new Binding("ColorName")
{
    Source = myDataObject
};

// Bind the data source to the TextBox control's Text dependency property
myText.SetBinding(TextBlock.TextProperty, myBinding);
```

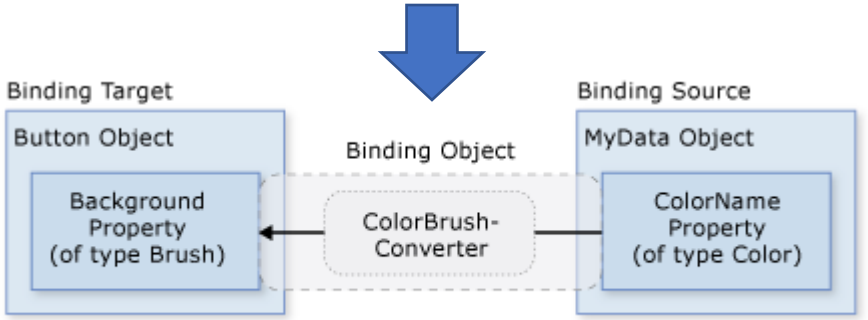
```
public class MainWindowData : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    8 references
    private void OnPropertyChanged([CallerMemberName] string name=null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
    }
```




```
[ValueConversion(typeof(Color), typeof(SolidColorBrush))]
public class ColorBrushConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
    {
        Color color = (Color)value;
        return new SolidColorBrush(color);
    }

    public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
    {
        return null;
    }
}
```



WPF – Propojení s daty (data binding) (4)

- Propojení kolekce dat
  - ListBox, ListView, TreeView
  - Mají ItemsSource, kde můžu *propojit* kolekci.
  - Nejsnazší je propojení s ObservableCollection<T>, další možnosti jsou třídy (rozšiřující) List<T>, Collection<T> či BindingList<T>.
- Collection views
  - Pokud propojím *view* s kolekcí, vznikne objekt starající se o zobrazení dat.
  - Umí aktualizovat data (metoda `Refresh()`, událost `INotifyCollectionChanged`).
  - Například pro `List` to bude instance `ListCollectionView`.
  - Hlavní idea je, pokud chci například třídit zobrazená data, neměním kolekci, ale setřídím data pomocí příslušného `ICollectionView`.
  - Kromě třídění je podporováno filtrování či shlukování.
  - Pokud nespecifikuji, bude po propojení view a kolekce vygenerována defaultní instance (`CollectionViewSource.GetDefaultView()`)

WPF – Propojení s daty (data binding) (4)

```
<Window.Resources>
    <CollectionViewSource
      Source="{Binding Source={x:Static Application.Current}, Path=AuctionItems}"
      x:Key="listingDataView" />
</Window.Resources>
```

```
<ListBox Name="Master" Grid.Row="2" Grid.ColumnSpan="3" Margin="8"
         ItemsSource="{Binding Source={StaticResource listingDataView}}" />
```

```
var data = this.DataContext as MainWindowData;
var view = CollectionViewSource.GetDefaultView(data?.Database?.People);
if (view != null && view.Filter == null)
{
    view.Filter += (o) =>
    {
        return true;
    };
}
```

```
this.customerListView = new ListCollectionView(this.Customers);
this.customerListView.Filter = null;
this.customerListView.Filter = this.FilterCustomersBasedOnSurename;
this.customersList.ItemsSource = this.customerListView;
```

```
private void ListingDataView_Filter(object sender, FilterEventArgs e)
{
    // Start with everything excluded
    e.Accepted = false;

    // Only inlcude items with a price less than 25
    if (e.Item is AuctionItem product && product.CurrentPrice < 25)
        e.Accepted = true;
}
```

WPF – Příklad ListView

```
<ListView ItemsSource="{Binding People, UpdateSourceTrigger=PropertyChanged}">
        <ListView.ItemTemplate>
                <DataTemplate>
                        <StackPanel Orientation="Horizontal">
                                <TextBlock Text="{Binding Name}" Margin="0,0,5,0"/>
                                        ...
                        </StackPanel>
                </DataTemplate>
        </ListView.ItemTemplate>
</ListView>
```

WPF – Validace dat (1)

- WPF nativně podporuje validaci dat.
- Třída `ValidationRule` a binding `ValidationRules`
- Existující validátory: `ExceptionValidationRule` nebo `DataErrorValidationRule`
- Vlastní validace: `IDataErrorInfo`

```
public class FutureDateRule : ValidationRule
{
    public override ValidationResult Validate(object value, CultureInfo cultureInfo)
    {
        // Test if date is valid
        if (DateTime.TryParse(value.ToString(), out DateTime date))
        {
            // Date is not in the future, fail
            if (DateTime.Now > date)
                return new ValidationResult(false, "Please enter a date in the future.");
        }
        else
        {
            // Date is not a valid date, fail
            return new ValidationResult(false, "Value is not a valid date.");
        }

        // Date is valid and in the future, pass
        return ValidationResult.ValidResult;
    }
```

```
<TextBox Name="StartDateEntryForm" Grid.Row="3"
         Validation.ErrorTemplate="{StaticResource validationTemplate}"
         Style="{StaticResource textStyleTextBox}" Margin="8,5,0,5" Grid.ColumnSpan="2">
    <TextBox.Text>
        <Binding Path="StartDate" UpdateSourceTrigger="PropertyChanged"
                 Converter="{StaticResource dateConverter}" >
            <Binding.ValidationRules>
                <src:FutureDateRule />
            </Binding.ValidationRules>
        </Binding>
    </TextBox.Text>
</TextBox>
```

WPF – Validace dat (2)

- Obecně je validace dat poměrně komplikovaná. Komplikované může být i zobrazení dat z validace.

```
<Style x:Key="textStyleTextBox" TargetType="TextBox">
    <Setter Property="Foreground" Value="#333333" />
    <Setter Property="MaxLength" Value="40" />
    <Setter Property="Width" Value="392" />
    <Style.Triggers>
        <Trigger Property="Validation.HasError" Value="true">
            <Setter Property="ToolTip"
                    Value="{Binding RelativeSource={RelativeSource Self}, Path=(Validation.Errors)[0].ErrorContent}"/>
        </Trigger>
    </Style.Triggers>
</Style>
```

WPF – Příkazy (1)

- Ve skutečných aplikacích často nechceme řešit kliknutí na konkrétní tlačítko, ale komplexnější příkazy.
  - WPF obsahuje mechanismus, řešící tuto situaci.
  - `ICommand` - `CanExecute` a `Execute`.
  - WPF obsahuje asi 100 předdefinovaných příkazů: uložit, vyjmout, vložit,…
  - `CommandTarget` specifikuje cíl příkazu, jinak je to objekt, který má „focus“.

```
<Window x:Class="Commands.UsingCommandsSample"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="UsingCommandsSample" Height="100" Width="200">
    <Window.CommandBindings>
        <CommandBinding Command="ApplicationCommands.New" Executed="NewCommand_Executed" CanExecute="NewCommand_CanExecute" />
    </Window.CommandBindings>

    <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
        <Button Command="ApplicationCommands.New">New</Button>
    </StackPanel>
</Window>
```

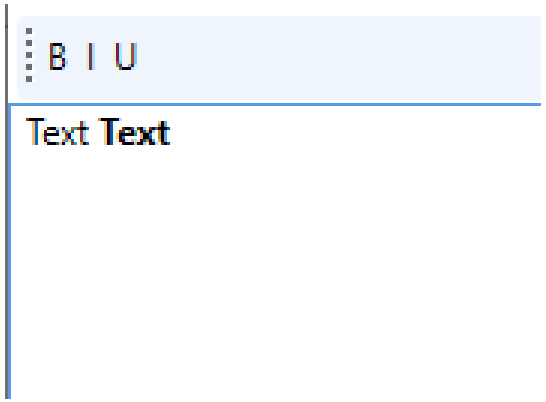
```
1 reference
private void NewCommand_CanExecute(object sender, CanExecuteRoutedEventArgs e)
{
    e.CanExecute = true;
}

1 reference
private void NewCommand_Executed(object sender, ExecutedRoutedEventArgs e)
{
    MessageBox.Show("The New command was invoked");
}
```

WPF – Příkazy (2)

- Příkazy jsou integrovány i do základních komponent.
  - Nejsnazší řešení jak s nimi zacházet.
- Můžeme definovat vlastní příkazy.

```
<StackPanel Orientation="Vertical">
    <ToolBar Name="mainToolBar" Height="30">
        <Button Command="EditingCommands.ToggleBold" ToolTip="Bold">
            B
        </Button>
        <Button Command="EditingCommands.ToggleItalic" ToolTip="Italic">
            I
        </Button>
        <Button Command="EditingCommands.ToggleUnderline" ToolTip="Underline">
            U
        </Button>
    </ToolBar>
    <RichTextBox VerticalAlignment="Center"
         AcceptsReturn="True" AcceptsTab="True"
        MinHeight="300" MaxHeight="450"/>
</StackPanel>
```



WPF – Další kapitoly (mimo rozsah této přednášky)

- Vstup dat (různé možnosti: text, myš, dotykový display)
- Lokalizace a globalizace dat
- Práce s médii a *kreslení*
- Bezpečnost
- …

Zdroje

- https://docs.microsoft.com/cs-cz/dotnet/
- https://devblogs.microsoft.com/
- https://www.codeguru.com/csharp/

- YAMIKANI FUKIZI, Kenneth, Jason DE OLIVEIRA a Michel BRUCHET. *Learn ASP.NET Core 3: Develop modern web applications*. Second edition. Packt Publishing, 2019. ISBN 978-1789610130.
- ALBAHARI, Joseph. *C# 10.0 in a Nutshell: The Definitive Reference*. O'Reilly Media; 1st edition, 2022. ISBN 978-1098121952.
- ALBAHARI, Joseph. *C# 10 and .NET 6 – Modern Cross-Platform Development: Build apps, websites, and services with ASP.NET Core 6, Blazor, and EF Core 6 using Visual Studio 2022 and Visual Studio Code*. 6th edition. Packt Publishing, 2021. ISBN 978-1801077361.