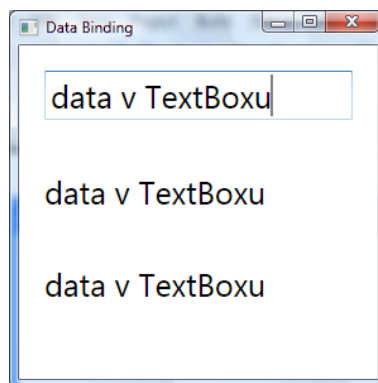


5. WPF - Data Binding

V XAMLu máme k dispozici několik dílčích technologií - pro změnu vzhledu používáme styly a šablony, abychom mohli reagovat na události používáme trigry a nyní se podíváme na *Data Binding*, který nám umožní v XAMLu pracovat s daty.

Příklad:

Vytvořte dvě textová pole, ve kterých bude zobrazen text zadaný do *TextBoxu*.



```
<StackPanel>
  <!-- Zdroj -->
  <TextBox Name="mySource" Text="muj text" />
  <!-- 1. TextBlock -->
  <TextBlock>
    <TextBlock.Text>
      <Binding ElementName="mySource" Path="Text" />
    </TextBlock.Text>
  </TextBlock>
  <!-- 2. TextBlock -->
  <TextBlock Text="{Binding ElementName=mySource, Path=Text}" />
</StackPanel>
```

* Pozor, rozlišujte *TextBox* a *TextBlock*. *TextBoxy* slouží pro zadávání textu (input pole), naopak pomocí *TextBlocku* pouze text zobrazujeme.

Data Binding se postará o provázání dat mezi textovými bloky a *TextBoxem* (zdrojem), takže text zadaný do *TextBoxu* se okamžitě zobrazí v obou textových blocích.

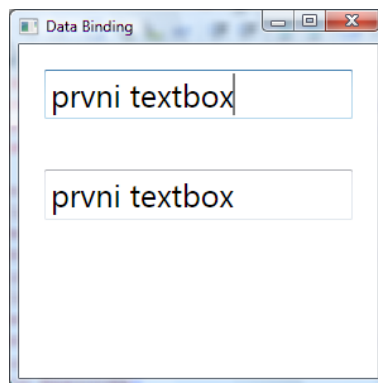
- 1) abychom k vlastnostem *TextBoxu* mohli při bindování přistupovat, pojmenovali jsme ho "mySource"
- 2) v obou *TextBlock* elementech jsme hodnotu *Text* nabíndovali; v druhém elementu jsme použili kratší způsob zápisu, který je ekvivalentní s prvním zápisem
 - *ElementName* - jméno elementu, jehož vlastnost chceme použít při bindování
 - *Path* - název vlastnosti

** U bindování se používá pouze kratší způsob zápisu (kód je přehlednější a zápis je rychlejší). Dále budeme používat pouze tento způsob zápisu.*

Jak se nabíndované vlastnosti *TextBlock* elementů chovají? Jakmile začneme psát nebo měnit text v *TextBoxu*, změna se okamžitě projeví i v nabíndovaných vlastnostech, tzn. že nabíndované hodnoty jsou automaticky synchronizovány se zdrojovou hodnotou.

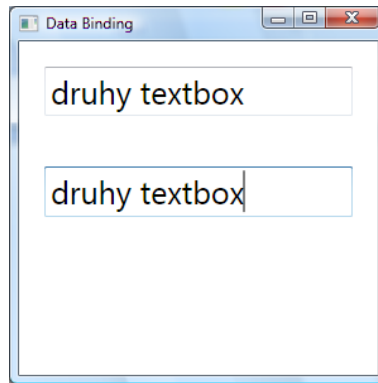
Příklad:

Vytvořte dva *TextBoxy*, první bude zdroj a text druhého *TextBoxu* nabíndujte.



```
<StackPanel>
  <!-- Zdroj -->
  <TextBox Name="myTextBoxSource" Text="prvni textbox" />
  <!-- Bindovani -->
  <TextBox Text="{Binding ElementName=myTextBoxSource, Path=Text}" />
</StackPanel>
```

Nyní již víme, že pokud uděláme změnu ve zdroji (první *TextBox*), ke změně dojde i v druhém *TextBoxu*. Co kdybychom ale změnili hodnotu nabíndované vlastnosti, projeví se změna ve zdroji? Ano.



Bindování je tedy implicitně obousměrná synchronizace hodnot.

** Při změně zdrojové hodnoty dojde okamžitě k aktualizaci nabindovaných hodnot, naopak při změně nabindované hodnoty dojde ke změně ve zdroji až ve chvíli, kdy druhý TextBox přestane být aktivní (LostFocus), tzn. například kliknutím do prvního TextBoxu*

Další atributy

Ovlivnit chování bindování můžeme pomocí těchto atributů

Mode - způsob synchronizace hodnot

<i>Mode = TwoWay*</i>	hodnoty jsou synchronizovány obousměrně, tzn. jak při změně hodnoty ve zdroji tak při změně hodnoty nabindované vlastnosti
<i>Mode = OneWay</i>	hodnoty jsou synchronizované pouze ze zdroje do nabindované vlastnosti, změny v nabindované vlastnosti jsou zdrojem ignorovány
<i>Mode = OneTime</i>	hodnoty jsou synchronizované pouze po startu, poté jsou všechny změny ignorovány

UpdateSourceTrigger - tímto atributem určíme kdy se aktualizuje zdrojová hodnota

<i>UpdateSourceTrigger = Explicit</i>	k synchronizaci dojde pouze pokud zavoláme metodu <i>UpdateSource</i>
<i>UpdateSourceTrigger = LostFocus*</i>	zdrojová hodnota se aktualizuje jakmile nabindovaná kontrola přestane být "aktivní" (<i>LostFocus</i>)
<i>UpdateSourceTrigger = PropertyChanged</i>	k aktualizaci zdrojové hodnoty dojde okamžitě jakmile je nabindovaná hodnota změněna

(*) defaultně

Bindování uživatelských objektů

Příklad:

Máme třídu s vlastnostmi *Jmeno* a *Prijmeni* (obě typu *string*). Vytvořte aplikaci se dvěma *TextBoxy* pro zobrazení a editaci hodnot těchto vlastností a dále tlačítko, které po kliknutí nastaví hodnoty

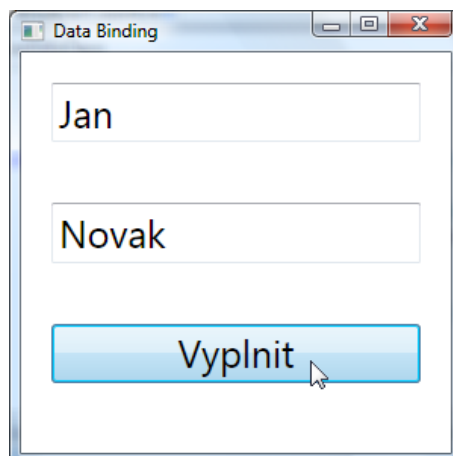
```
objekt.Jmeno = "Jan"  
objekt.Prijmeni = "Novak"
```

Třída

```
public class UserInfoClass  
{  
    private string jmeno = string.Empty;  
    private string prijmeni = string.Empty;  
    public string Jmeno  
    {  
        get { return jmeno; }  
        set { jmeno = value; }  
    }  
  
    public string Prijmeni  
    {  
        get { return prijmeni; }  
        set { prijmeni = value; }  
    }  
}
```

Abychom v XAMLu mohli bindovat na vlastnosti uživatelského objektu, musí být třída objektu poděděna od interface *INotifyPropertyChanged* - to umožní avizovat změny v hodnotách vlastností.

Jakým způsobem budeme v XAMLu přistupovat k objektu? Nejjednodušší způsob je vložit tento objekt do vlastnosti *DataContext* některého z rodičovských elementů. Vlastnost *DataContext* slouží pro určení defaultního zdroje při bindování dat.



Window1.xaml

```
<StackPanel Name="myStackPanel">
  <TextBox Text="{Binding Path=Jmeno}" />
  <TextBox Text="{Binding Path=Prijmeni}" />
  <Button Click="Fill">Vyplnit</Button>
</StackPanel>
```

Window1.xaml.cs

```
using System;
using System.Windows;
using System.ComponentModel;
```

```
namespace DataBinding
{
    public partial class Window1 : System.Windows.Window
    {
```

```
        UserInfoClass user;
```

```
        public Window1()
```

```
        {
            InitializeComponent();
```

```
            user = new UserInfoClass();
            myStackPanel.DataContext = user;
        }
```

konstruktor

```
        void Fill(object sender, RoutedEventArgs e)
```

```
        {
            user.Jmeno = "Jan";
            user.Prijmeni = "Novak";
        }
```

metoda Fill

```
        public class UserInfoClass : INotifyPropertyChanged
```

```
        {
            public event PropertyChangedEventHandler PropertyChanged;
            private void NotifyPropertyChanged(string info)
            {
                if (PropertyChanged != null)
                    PropertyChanged(this, new PropertyChangedEventArgs(info));
            }
        }
```

třída
UserInfoClass

```
        private string jmeno = "jméno";
        private string prijmeni = "příjmení";
```

```
        public string Jmeno
```

```
        {
            get { return jmeno; }
            set { jmeno = value; NotifyPropertyChanged("Jmeno"); }
        }
```

```
        public string Prijmeni
```

```
        {
            get { return prijmeni; }
            set { prijmeni = value; NotifyPropertyChanged("Prijmeni"); }
        }
```

```
    }
```

```
}
```

1. vložili jsme *System.ComponentModel* namespace, ve kterém se nachází interface *INotifyPropertyChanged*
2. abychom mohli vlastnosti objektu bindovat, podědili jsme třídu *UserInfoClass* jsme od rozhraní *INotifyPropertyChanged*
3. vytvořili jsme událost *PropertyChanged* a metodu *NotifyPropertyChanged*, která tuto událost volá
4. metodu *NotifyPropertyChanged* voláme vždy, když se změní hodnota některé z vlastností (jako parametr je uveden název vlastnosti)
5. po kliknutí na tlačítko "Vyplnit" se zavolá metoda *Fill*, která změní hodnoty objektu *user*

Jak se aplikace chová?

- změna hodnoty v objektu ⇒ změna se projeví v *TextBoxech*
- změna hodnoty v *TextBoxu* ⇒ hodnota se změní v objektu
- kliknutí na tlačítko "Vyplnit" ⇒ změní se hodnoty vlastností objektu ⇒ změní se hodnoty v *TextBoxech*

Závěr

Data Binding je "svázání" vlastností, jakmile se změní hodnota zdrojové vlastnosti, projeví se zároveň i v provázaných vlastnostech a naopak, pokud v provázaných vlastnostech dojde ke změně, tato změna se projeví zároveň ve zdroji.

Bez *Data Bindingu* bychom se obešli, mohli bychom řešit tyto situace programově přes událostmi volané metody. Tento postup nemůžeme ale v XAMLu použít a proto zde máme k dispozici *Data Binding*.

Více informací o WPF naleznete na www.unitedstatesof.net/wpf

Aleš Šturala, 4. 2. 2007

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