WPF – Binding

Petr Novák / 2021-05-26

Obsah

1 Úvod

Converter je možnost jak změnit / zpracovat datové typy poskytované programovou třídou na typy vyžadované GUI prvkem. Converter se tedy používá pro tyto hlavní účely:

- Běžná converter – Změna datového typu vlastnosti z programové třídy na datový typ vlastnosti GUI prvku (například **Bool** na **Visibility**).
- Multi converter – Pro sloučení více vlastností programové třídu na jednu vlastnost GUI prvku (například **Visibilty** je podmíněno hodnotou **Bool** a **Int** současně).

V adresáři **XUtils/Converters** jsou zdrojové kódy pro některé zde uvedené ukázky. Zejména pro jejich snadné použití. Zdrojové kódy rovněž obsahují popis použití.

2 Postup použití Converter

Aby mohl být **Converter** použit je potřeba udělat následující kroky:

- A) Na začátek **XAML** zapsat kde je **Converter** umístěn v programovém kódu (stejně jako v případě jakéhokoli GUI prvku). Například:

```
xmlns:XUtilsConverterUni01="clr-namespace:XUtils.Converters"
```

- B) Do **Resources** (například) dané stránky pojmenovat **Converter**, aby mohl být použit v **XAML** stránce. Například:

```
<Window.Resources>
      <XUtilsConverters:BoolToStringConverter x:Key="BoolToStringConverter" />
<Window.Resources>
```

- C) Použít **Converter** v **XAML** nějakého GUI prvku. Například:

```
<Label Content="{Binding …, Converter={StaticResource BoolToStringConverter}}">
```

- D) Vytvořit / definovat požadovaný **Converter**. Například:

```
// hodnota 'Bool' -> (string) „Ano“ / „Ne“
public class BoolToStringConverter : IValueConverter
{
     public object Convert(object value, Type targetType, object parameter,
            System.Globalization.CultureInfo culture)
       { return /* konverze 'Bool' do 'Color' */; }

    public object ConvertBack(object value, Type targetType, object parameter,
            System.Globalization.CultureInfo culture)
       { throw new NotImplementedException(); }
}
```

Aby mohl být **Converter** použit tak je nutno vždy vykonat tyto kroky. Zřejmě není možno **Converter** použít v **XAML** bez jeho definování v **Resources**.

3 Single Converter

Uživatelský „Converter“ je tedy potřeba vytvořit když typ hodnoty vlastnosti neodpovídá typu hodnoty akceptované položkou v GUI. (Případně rovněž pokud je potřeba hodnotu z vlastnosti třídy nějak upravit před jejím předáním položce v GUI. Například znegovat hodnotu typu „Bool“).
Požadavky na „Converter“:

- Jedná se o třídu odvozenou ze třídy „IValueConverter“.
- Musí implementovat vlastnost „`public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)`“ pro konverzi typu hodnoty z vlastnosti třídy na typ hodnoty do položky v GUI.
- Musí implementovat vlastnost „`public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)`“ pro konverzi typu hodnoty z položky v GUI na typ hodnoty do vlastnosti třídy. Tato metoda musí být implementována, i když tento směr konverze není použit (může tedy vracen „null“).
- Obě vlastnosti přijímají parametr jako „object“ (vstupní typ hodnoty pro konverzi) a rovněž vracejí parametr „object“ (výstupní typ hodnoty po konverzi).

Zde jsou uvedeny některé příklady „Converter(s)“.

```
// hodnota 'Bool' -> '!Bool'
public class BoolToNonBoolConverter : IValueConverter
{
  public object Convert(object value, Type targetType, object parameter,
  System.Globalization.CultureInfo culture)
    { return !(bool)value; }

  public object ConvertBack(object value, Type targetType, object parameter,
  System.Globalization.CultureInfo culture)
       { throw new NotImplementedException(); }
}

// hodnota 'Bool' -> (Visibility) Visible / Collapsed
public class BoolToVisibilityConverter : IValueConverter
{
  public object Convert(object value, Type targetType, object parameter,
  System.Globalization.CultureInfo culture)
```

```
  { return (bool)value ? Visibility.Visible : Visibility.Collapsed; }

  public object ConvertBack(object value, Type targetType, object parameter,
  System.Globalization.CultureInfo culture)
  { throw new NotImplementedException(); }
}

// hodnota 'Bool' -> (string) „Ano“ / „Ne“
public class BoolToStringConverter : IValueConverter
{
  public object Convert(object value, Type targetType, object parameter,
  System.Globalization.CultureInfo culture)
  { return (bool)value ? „Ano“ : „Ne“; }

  public object ConvertBack(object value, Type targetType, object parameter,
  System.Globalization.CultureInfo culture)
  { throw new NotImplementedException(); }
}
```

Předpokládáme, že náš „Converter“ je umístěn v „namespace XUtils.Controls“. Aby mohl být „Converter“ v binding použít, tak je postup následující:

- Vložit na něho cestu do cest v XAML, například:
  `xmlns:XUtilsConverters="clr-namespace:XUtils.Controls"`
- Potřeba jej přidat do „Resources“ tohoto dialogu, například pomocí:
  ```
  <Window.Resources>
   <XUtilsConverters:BoolToStringConverter x:Key="BoolToStringConverter" />
  <Window.Resources>
  ```
- Nyní jej lze použít v položce GUI při binding:
  `<Label Content="{Binding …, Converter={StaticResource BoolToStringConverter}}">`

4 Multi Coverter – Text

Pokud je potřeba **Converter** pro více vstupních hodnot, je nutno použít **Mult-Converter**. Jeho použití je obdobné jako v případě pouze jedné hodnoty. Nejprve vytvoříme položky v GUI / XAML, například následovně (jde o velmi jednoduchý principiální příklad):

```
<!-- prvek v GUI bude „TextBlock“ -->
<TextBlock>
  <!-- jeho vlastnost „Text“ -->
  <TextBlock.Text>
    <!-- bude složena z několika „binding(s)“ pomoci „Converter“ -->
    <MultiBinding Converter="{StaticResource ConcatTextsConverter}">
      <Binding Path="TextA"/>
      <Binding Path="TextB"/>
    </MultiBinding>
  </TextBlock.Text>
</TextBlock>
```

V **TextBlock** do jeho vlastnosti **Text** jsou v podstatě navázány dvě vlastnosti programové třídy a to **TextA** a **TextB** (obě typu **String**). Potřebný **Converter** tedy musí přijímat dva řetězce (**TextA** a **TextB**) a na výstupu poskytovat jeden řetězec (pro **TextBlock.Text**). Jeho obsah bude následující:

```
// „Converter“ pro vice vstupnnich parametru
public class ConcatTextsConvertor : IMultiValueConverter
{
  // smer z „vlastnosti“ do „GUI“
```

```
  public object Convert(object[] values, Type targetType, object parameter,
    System.Globalization.CultureInfo culture)
  {
    // slozeni dvou vstupnich retezcu do jednoho vystupniho, oddelenych strednikem
    return String.Concat((String)values[0], ",", (String)values[1]);
  }
  // smer z „GUI“ do „vlastnosti“ (asi nema vyznam, pouze pro nazornost)
  public object[] ConvertBack(object value, Type[] targetTypes, object parameter,
    System.Globalization.CultureInfo culture)
  {
    // rozdeleni vstupniho retezce podle stredniku na vystupni pole pod-retezcu
    return (value as string).Split(',');
  }
}
```

Aby mohl být uvedený **Multi-Converte** použít, je potřeba na začátek XAML zadat umístění, tedy:

```
xmlns:XUtilsConverterUni01="clr-namespace:XUtils.Converters"
```

Dále do Resources XAML stránky vložit:

```
<XUtilsConverterUni01:ConcatTextConverter x:Key="ConcatTextConverter" />
```

Z programového hlediska tento **Convertor** nemá skoro žádný význam, protože spojení dvou řetězců lze udělat přímo ve vlastnosti programové třídy. Je ale velmi názorný pro pochopení činnosti **Multi-Converter**.

5 Parametr pro Converter

Někdy je potřeba použít na několika místech skoro stejný **Converter** avšak lišící se jen nějakým malým parametrem, například výstupním textem, barvou a atd. Aby nebylo nutno pro každý tento případ vytvářet samostatný **Convertor**, zle vytvořit pouze jeden a jeho chování upravit parametrem při jeho lokálním použití u GUI prvku. Parametr pro **Convertor** se uvádí v **XAML** v **GUI** prvku do položky `ConverterParameter`. Příklad principu zápisu je následující:

```
Foreground ="{Binding ValueBool,
    Converter={StaticResource BoolToColorConverter}, ConverterParameter=Green}"
```

V programové třídě pro **Convertor** se předaný parametr vyzvedne v metodě **Convert** a to v jejím parametru **parameter**. Hodnota předaná do **parameter** je typu **Object** a je ji tedy nutno přetypovat na požadovaný datový typ (většinou asi text).

```
public object Convert(object value, Type targetType, object parameter,
       System.Globalization.CultureInfo culture)
    { return /* zde lze pouzit zadaný parametr 'parameter' */; }
```

Tímto stylem lze velmi snadno vytvořit určitý typ dostatečně univerzálních **Converter** a poté pouze pomocí zadaného parametru pro příslušný GUI prvek upravovat jeho chování / výstup.

6 Multi Coverter – Obecný

Zde je uveden příklad pro celkem univerzální **Multi-Converter** převádějící hodnotu **Bool** na **Brush/Color** podle aktuálního nastavení / zadání v XAML. Jeho činnost a použití je následující. Na začátek XAML zadat umístění, tedy:

```
xmlns:XUtilsConverterUni01="clr-namespace:XUtils.Converters"
```

Dále do Resources XAML stránky vložit:

```
<XUtilsConverterUni01:BoolToColorConverterUni01 x:Key="BoolToColorConverterUni01" />
```

K příslušnému GUI prvku doplnit například:

```
<!-- 'ValueBool=0' text je 'Black' / 'ValueBool=1' text je 'Green' -->
Foreground ="{Binding ValueBool,
  Converter={StaticResource BoolToColorConverterUni01}, ConverterParameter=Green}"
<!-- 'ValueBool=0' text je 'red' / 'ValueBool=1' text je 'Green' -->
Foreground ="{Binding ValueBool,
  Converter={StaticResource BoolToColorConverterUni01}, ConverterParameter=Red|Green}"
```

Uvedený **Multi-Converter** převádí hodnotu **Bool** na **Brush/Color**. Výstupní barvy však nejsou pevně zadány v **Multi-Converter**, ale lze je nastavit podle potřeby a to pomocí `ConverterParameter` :

- Při zadání pouze jedná barvy je pro vstupní hodnotu **Bool=0** vytvořena výstupní barva **Black** a pro hodnotu **Bool=1** vytvořena výstupní barva zadaná v hodnotě `ConverterParameter` .
- Při zadání dvou barev je pro vstupní hodnotu **Bool=0** vytvořena výstupní barva odpovídající prvnímu parametru v `ConverterParameter` a pro hodnotu **Bool=1** vytvořena výstupní barva zadaná jako druhá parametr v hodnotě `ConverterParameter` . Parametry jsou odděleny pomocí `|`.

```
public class BoolToColorConverterUni01 : IValueConverter
{
  public object Convert(object value, Type targetType, object parameter,
       System.Globalization.CultureInfo culture)
  {
    // predvyplni se dve zakladni / cerne barvy
    string[] colorsStr = { "Black", "Black" };
    // pokud je zadana pouze jedna barva, tak je povazovana za druhou
    if (((string)parameter).Contains("|") == false) { colorsStr[1] =
       (string)parameter; }
    // pokud jsou zadany dve barvy (oddelene pomoci '|', tak se pouziji obe)
    else { colorsStr = ((string)parameter).Split("|"); }
    // vystupem je prvni nebo druha barva
    return new SolidColorBrush((Color)ColorConverter.ConvertFromString(
       colorsStr[((bool)value== false) ? 0 : 1]));
  }
  public object ConvertBack(object value, Type targetType, object parameter,
       System.Globalization.CultureInfo culture)
       { throw new NotImplementedException(); }
}
```