Konwertery w .NET MAUI

Bartosz Miąskowski

Czym są konwertery?

- Konwerter to klasa, która przekształca dane podczas bindingu z UI. Jest to pewnego rodzaju pośrednik lub tłumacz między UI oraz ViewModelem.

- Kiedy używamy konwerterów?
  - jeżeli chcemy zmieniać dynamicznie typy danych (np. `bool <-> Color`),
  - chcemy sformatować jakąś wartość (np. 1234 -> 1234.00 zł),
  - chcemy aby ViewModel był możliwie zwięzły.

```
// Bez konwertera, musimy konwertować dane w VM
public Color ValidityColor => IsValid ? Colors.Green : Colors.Red;
```

```
<Label Text="Status"
       TextColor="{Binding IsValid, Converter={StaticResource BoolToColorConverter}}" />
```

Konwerter to logika transformacji danych w UI. **Kod jest czystszy**, a logika UI jest częścią UI.

Kiedy sam binding nie wystarcza?

- Zakładamy, że w VM posiadamy właściwość `bool IsValid` – chcemy na podstawie wartości wyświetlić komunikat:
  - „**Poprawne dane**” – jeżeli wartość to `true`,
  - „**Błędne dane**” – jeżeli wartość to `false`.

```
public string StatusText => IsValid ? "Poprawne dane" : "Błędne dane";
public Color StatusColor => IsValid ? Colors.Green : Colors.Red;
```

Bez konwertera, musimy dodać dwie nowe właściwości w VM – tylko do obsługi jednej kontrolki.

```
<Label Text="{Binding IsValid, Converter={StaticResource BoolToTextConverter}}" />
```

Z konwerterem, wystarczy go wskazać w kodzie XAML.

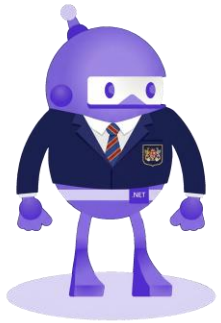
Interfejs `IValueConverter`

- Każdy konwerter w .NET MAUI implementuje interfejs `IValueConverter`.
  - `Convert` jest używany przy tłumaczeniu danych z VM do UI.
  - `ConvertBack` służy do tłumaczenia danych z UI do VM.

- Parametry tych metod oznaczają:
  - `object` **`value`** – wartość z bindingu,
  - `Type` **`targetType`** – typ docelowy (np. Color),
  - `object` **`parameter`** – dodatkowy parametr z XAML,
  - `CultureInfo` **`culture`** – kultura, np. sposób formatowania liczb.

```
public interface IValueConverter
{
    object Convert(object value, Type targetType, object parameter, CultureInfo culture);

    object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture);
}
```



Ważne!
Wartość jest typu `object`, więc za każdym razem będzie trzeba **zrobić rzutowanie** (*unboxing*).

```
var isValid = (bool)value;
```

Implementacja `IValueConverter`

- Chcemy stworzyć konwerter, który w zależności od wartości `bool` zwraca kolor zielony lub czerwony.

```
public class CustomColorConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        var isValid = (bool)value;
        return isValid ? Colors.Green : Colors.Red;
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        var color = (Color)value;
        return color == Colors.Green;

        // Jeżeli ConvertBack nie jest nam z jakiegoś powodu potrzebne, możemy zrobić:
        // throw new NotImplementedException();
    }
}
```

Konwerter dwukierunkowy – `int <-> string`

1. Tworzymy klasę konwertera.

```
using System.Globalization;

public class IntToStringConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        if (value == null)
            return string.Empty;

        return value.ToString();
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        var text = value as string;

        if (int.TryParse(text, out var result))
            return result;

        return 0; // fallback
    }
}
```

Konwerter dwukierunkowy – `int <-> string`

2. Dodajemy pole `int` do VM.

```
[ObservableProperty]
private int age;
```

3. Rejestrujemy konwerter w `ResourceDictionary` w XAML.

```
<ContentPage.Resources>
    <ResourceDictionary>
        <local:IntToStringConverter x:Key="IntToStringConverter" />
    </ResourceDictionary>
</ContentPage.Resources>
```

Konwerter dwukierunkowy – `int <-> string`

4. Podpinamy konwerter do kontrolki w XAML.

```
<Entry Text="{Binding Age,
                Converter={StaticResource IntToStringConverter}, Mode=TwoWay}" />
```

- Co się stanie, jeżeli nie użyjemy `Mode=TwoWay`?
  - Domyślnym trybem bindingu w konwerterach jest `OneWay` – wtedy będzie działało konwertowanie wartości z VM do UI, ale nie będzie możliwa ich konwersja z UI do VM.

W tym przypadku, jeżeli użytkownik wpisze coś, co nie jest liczbą – konwerter zwróci do VM wartość 0.

Parametr konwertera

- Dzięki parametrowi możemy przekazać do konwertera dodatkową wartość. Na poniższym przykładzie przekazujemy do konwertera tekst, który ma się wyświetlić w różnych scenariuszach.

```
public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
{
    var val = (bool)value;

    var param = parameter as string;
    var parts = param?.Split('|');

    var trueText = parts?[0] ?? "True";
    var falseText = parts?[1] ?? "False";

    return val ? trueText : falseText;
}
```

```
<Label Text="{Binding IsValid,
                      Converter={StaticResource BoolToTextConverter},
                      ConverterParameter='OK|Błąd'}" />
```

Gotowe konwertery w CommunityToolkit.Maui

- Biblioteka CommunityToolkit.Maui dostarcza już szereg konwerterów gotowych do użycia – podobnie jak to miało miejsce z Behaviors.

```
<Label IsVisible="{Binding IsBusy, Converter={StaticResource InvertedBoolConverter}}" />
```

`InvertedBoolConverter`

Odwraca wartość bool – robi negację.

Gotowe konwertery w CommunityToolkit.Maui


```
<Label IsVisible="{Binding ProfilePicture, Converter={StaticResource IsNotNullConverter}}" />
```

`IsNotNullConverter`

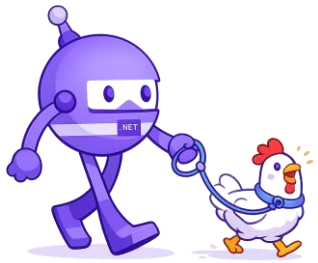
Jeżeli dana właściwość w VM jest nullem, zwraca wartość `false`.

Gotowe konwertery w CommunityToolkit.Maui

```
<Label Text="{Binding IsValid,
                      Converter={StaticResource BoolToObjectConverter},
                      ConverterParameter='OK|Błąd'}" />
```

`BoolToObjectConverter`

W zależności od wartości logicznej, pozwala wyświetlić dany tekst.

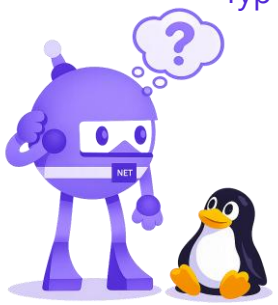


Obsługa <RadioButton />

- Załóżmy sytuację, w której chcielibyśmy utworzyć kilka pól typu radio oraz otrzymywać wartość enum, przypisaną do aktualnie zaznaczonego przycisku.
- Dzięki konwerterom możemy to zrobić w bardzo prosty sposób, bez pisania skomplikowanej logiki w VM.

```
public enum OddsNotation
{
    Decimal,
    Fractional,
    American,
    Shares
}
```

Typ wyliczeniowy, który chcemy pozyskać.



```
private OddsNotation _inputNotation = OddsNotation.Decimal;
public OddsNotation InputNotation
{
    get => _inputNotation;
    set
    {
        var prev = _inputNotation;
        if (SetProperty(ref _inputNotation, value))
        {
            // Handle any additional logic when the input notation changes, if necessary
            if (value == OutputNotation)
            {
                OutputNotation = prev;
            }
        }
    }
}
```

Właściwość w ViewModelu

Obsługa <RadioButton />

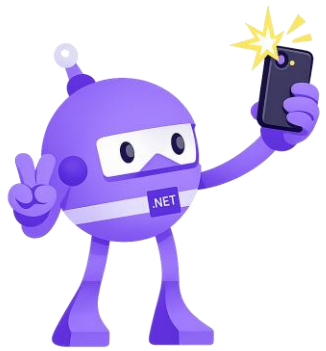
- Celem zbindowania wartości z wielu elementów radio, do jednej właściwości w VM, zbudujemy konwerter, który na podstawie tego czy dany przycisk jest zaznaczony, przypisze odpowiednią wartość enum do właściwości w VM.

```
public class EnumToBoolConverter : IValueConverter
{
    public object? Convert(object? value, Type targetType, object? parameter, CultureInfo culture)
    {
        return value?.ToString() == parameter?.ToString();
    }

    public object? ConvertBack(object? value, Type targetType, object? parameter, CultureInfo culture)
    {
        if (value is bool isChecked && isChecked && parameter != null)
            return Enum.Parse(targetType, parameter.ToString());

        // Coś musimy tutaj zwrócić, ale nie ma sensu zwracać żadnej wartości.
        return Binding.DoNothing;
    }
}
```

Kod konwertera do obsługi <RadioButton />



Obsługa <RadioButton />

- Kiedy mamy już konwerter, możemy dodać obsługę przypisania wartości enum do właściwości w VM w XAMLu.

```
<VerticalStackLayout>
    <RadioButton Content="Decimal odds"
                 IsChecked="{Binding InputNotation, Converter={StaticResource EnumToBoolConverter},
                    ConverterParameter={x:Static data:OddsNotation.Decimal}}" />
    <RadioButton Content="Fraction odds"
                 IsChecked="{Binding InputNotation, Converter={StaticResource EnumToBoolConverter},
                    ConverterParameter={x:Static data:OddsNotation.Fractional}}" />
    <RadioButton Content="American odds"
                 IsChecked="{Binding InputNotation, Converter={StaticResource EnumToBoolConverter},
                    ConverterParameter={x:Static data:OddsNotation.American}}" />
    <RadioButton Content="Shares"
                 IsChecked="{Binding InputNotation, Converter={StaticResource EnumToBoolConverter},
                    ConverterParameter={x:Static data:OddsNotation.Shares}}" />
</VerticalStackLayout>
```

Podłączenie konwertera do `<RadioButton />`

Ważne!
Aby działało **silne typowanie typów wyliczeniowych**, należy najpierw je **zaimportować w XAMLu**.
`xmlns:data="clr-namespace:OddsConverter.Data"`

Konwertery – dobre praktyki

- Jeden konwerter powinien mieć prostą logikę – konwersja pojedynczego typu na inny.
- Zawsze zwracaj uwagę na bezpieczeństwo typów – przed rzutowaniem, sprawdzaj, czy w object na pewno znajduje się typ, którego oczekujesz.
- Jeżeli często używasz danego konwertera, możesz umieszczać go w `App.xaml` lub osobnym przeznaczonym do tego pliku, aby każdorazowo nie powielać tego samego kodu w `ContentPage`.
- Konwertery wpływają na responsywność UI – staraj się unikać umieszczania tam ciężkich i wymagających operacji.

MultiBinding

- Do tej pory korzystaliśmy wyłącznie z bindingu jednej wartości w VM do jednej właściwości w kontrolce XAML. Czasami jednak potrzebujemy zbindować więcej niż jedną wartość.
- Wyobraźmy sobie sytuację, w której znamy dwie drużyny, które będą między sobą rozgrywały mecz:
  - home,
  - away.
- W UI chcemy wyświetlić nazwę meczu, czyli „Arsenal v Manchester City".
- Chcemy to osiągnąć, bez tworzenia nowej właściwości w VM typu `EventName`.

```
public string HomeTeam { get; set; } // Arsenal
public string AwayTeam { get; set; } // Manchester City
```

Właściwości w ViewModelu

Konwerter `IMultiValueConverter`

- Możemy bez problemu zbindować do właściwości kontrolki więcej niż jedną wartość, musimy w tym celu skorzystać z interfejsu `IMultiValueConverter`.

```
using System.Globalization;

public class MatchConverter : IMultiValueConverter
{
    public object Convert(object[] values, Type targetType, object parameter, CultureInfo culture)
    {
        var home = values[0] as string;
        var away = values[1] as string;

        return $"{home} vs {away}";
    }

    public object[] ConvertBack(object value, Type[] targetTypes, object parameter, CultureIn
        => throw new NotImplementedException();
}
```

Konwerter implementujący `IMultiValueConverter`

```
<Label>
    <Label.Text>
        <MultiBinding Converter="{StaticResource MatchConverter}">
            <Binding Path="HomeTeam" />
            <Binding Path="AwayTeam" />
        </MultiBinding>
    </Label.Text>
</Label>
```

Korzystanie z MultiBindingu w XAML

Pułapki w konwerterach wielowartościowych

- `ConvertBack` prawie nigdy nie ma sensu (*bardzo trudne do sensownego użycia oraz rzadko potrzebne*).
- Kolejność parametrów ma znaczenie.

```
<Binding Path="HomeTeam" />
<Binding Path="AwayTeam" />
```

```
values[0] → HomeTeam
values[1] → AwayTeam
```

- Brak silnego typowania, wszystko jest `object` (*brak bezpieczeństwa typów, łatwo popełnić błąd*).

```
var home = values[0] as string;
```

- Wydajność – multibinding może wywoływać się często. Starajmy się unikać umieszczania tam operacji ciężkich obliczeniowo.