Rozdział 14.
Projektowanie własnych kontrolek

Autor: Jacek Matulewski

e-mail: jacek@phys.uni.torun.pl

Fragment książki „Visual Studio 2017. Tworzenie aplikacji Windows w języku C#” (Helion, 2018) udostępniony studentom Wydziału Fizyki, Astronomii i Informatyki Stosowanej UMK w semestrze zimowym 2020/2021.

Załóżmy, że bardzo napracowaliśmy się nad wyglądem jakiejś kontrolki (dla ustalenia uwagi niech to będzie nasz przycisk z logo i nazwą uniwersytetu) i podejrzewamy, że użyjemy jej ponownie. Łatwo można z takiej kontrolki zrobić „pakiet” gotowy do wielokrotnego stosowania. Służą do tego kontrolki projektowane przez użytkownika (ang. *user control*) lub kontrolki niestandardowe (ang. *custom control*). W przypadku kontrolki, której typ jest jedną ze standardowych kontrolek (w naszym przypadku przycisk `Button`), a my jedynie ją modyfikujemy lub wzbogacamy, co przypomina zmiany wprowadzane w klasie przy dziedziczeniu, lepszym wyborem jest *Custom Control*. Przećwiczymy oba rodzaje kontrolek.

User Control

Jak już wspomniałem, to nie jest najlepszy rodzaj kontrolki w tym przypadku. Pomijając sprawy zajęcia pamięci, wadą tego rozwiązania jest to, że tracimy dostęp do atrybutów przycisku i tym samym możliwość wpływania na ten element po umieszczeniu kontrolki w kodzie XAML, chociażby na jego zawartość.

Pozostajemy przy tym samym projekcie *XamlWpf*.

1. Z menu *Project* środowiska Visual Studio wybieramy polecenie *Add User Control…*. Wybieramy nazwę pliku XAML dla przycisku, np. *PrzyciskUMK.xaml*.
2. Po kliknięciu *Add* w oknie edytora pojawi się kod kontrolki, podobny do kodu okna, z tą różnicą, że nadrzędnym elementem jest nie `Window`, lecz `UserControl`. Pozostała część kodu jest podobna do pustego okna. Widoczne są deklaracje przestrzeni nazw oraz element `Grid` organizujący zawartość projektowanej kontrolki.
3. Nasza kontrolka składa się tylko z przycisku, wobec tego usuwamy element `Grid` i zamiast niego wstawiamy kod z listingu 10.7 (listing 14.1).

Listing 14.1. Kod XAML nowej kontrolki ze wstawionym przyciskiem

```
<UserControl x:Class="XamlWpf.PrzyciskUMK"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:XamlWpf"
             mc:Ignorable="d"
```

```
             d:DesignHeight="100" d:DesignWidth="200">
        <Button HorizontalAlignment="Center" VerticalAlignment="Center"
                Width="200" Height="100">
            <Button.Foreground>White</Button.Foreground>
            <Button.Background>
                <LinearGradientBrush StartPoint="0,0.5" EndPoint="1,0.5">
                    <GradientStop Color="Blue" Offset="0.0" />
                    <GradientStop Color="Navy" Offset="1.0" />
                </LinearGradientBrush>
            </Button.Background>
            <StackPanel Orientation="Horizontal">
                <Image Width ="70" Height="70" Source="logo.gif" />
                <TextBlock FontSize="16" Margin="10,0,0,0" Foreground="White">
                    <Run Foreground="Yellow">Uniwersytet</Run><LineBreak/>
                    Mikołaja<LineBreak/>
                    Kopernika
                </TextBlock>
            </StackPanel>
        </Button>
    </UserControl>
```

4. Budujemy cały projekt. Dzięki temu po powrocie do kodu XAML okna (zakładka *MainWindow.xaml* edytora kodu) w podoknie *Toolbox* powinniśmy zobaczyć nową sekcję o nazwie *XamlWpf Controls*, a w niej kontrolkę oznaczoną jako *PrzyciskWPF (XamlWpf)*.
5. Usuwamy z okna projektowany w przycisk i zamiast niego wstawiamy utworzoną na jego wzór kontrolkę. Ustawiamy jej szerokość i wysokość:

```
<local:PrzyciskUMK Width="200" Height="100" Margin="10,10,10,10" />
```

Nie możemy tu użyć stylu `SzablonNiebieski`, bo nie zgadza się typ kontrolki.

Nawet po usunięciu siatki przycisk nie jest całą kontrolką, lecz tylko jest w niej zawarty. To oznacza, że własności i zdarzenia kontrolki nie są własnościami i zdarzeniami przycisku, nie będziemy więc mogli łatwo zmienić „z zewnątrz" np. pędzli użytych do rysowania przycisku[1]. Dalszy rozwój kontrolki, którego celem byłoby przywrócenie kontroli nad jej wyglądem, oznacza konieczność edycji kodu C#. Możemy użyć do tego zdarzeń i *code-behind* kontrolki (plik *PrzyciskUMK.xaml.cs* został dołączony do projektu podczas definiowania kontrolki) lub wzorca MVVM omówionego w trzeciej części książki.

Kontrolki typu *User Control* mogą być umieszczane w bibliotekach. Służy do tego szablon projektu o nazwie *WPF User Control Library (.NET Framework)*.

Custom Control

Drugi rodzaj kontrolki, bardziej nadający się do tworzenia niestandardowego przycisku, to *custom control*. Najwygodniej jest dodać do rozwiązania bibliotekę takich kontrolek.

1. W oknie *Solution Explorer* zaznaczamy całe rozwiązanie (nie projekt aplikacji).

[1] Pomijam fakt, że własności są „dziedziczone" przez kontrolki-dzieci. To dziedziczenie nie działa jednak w sytuacji, w której w kontrolce-dziecku ta własność została nadpisana.

2. Z jego menu kontekstowego wybieramy *Add, New Project*. W otwartym oknie wskazujemy zakładkę *Windows Classic Desktop*, a na niej szablon *WPF Custom Control Library (.NET Framework)*. Następnie wybieramy nazwę biblioteki, np. *Przyciski.xaml*.
3. W utworzonej bibliotece zmieniamy nazwę pliku z *CustomControl1.cs* na *PrzyciskUMK.cs*. Pojawi się pytanie, czy zmienić także nazwę klasy kontrolki. Zróbmy to. Nie obawiajmy się powtórzonej nazwy kontrolek — znajdują się w różnych przestrzeniach nazw.
4. Kod C# nowej kontrolki jest bardzo krótki — zawiera jedynie deklarację klasy z konstruktorem. Zmienimy jej klasę bazową z ogólnej `Control` na bardziej konkretną `Button` (listing 14.2). Ponieważ nie będziemy zmieniali sposobu działania przycisku, a jedynie jego wygląd, ta klasa pozostanie już niezmieniona.

Listing 14.2. Kod C# kontrolki. Usunąłem deklaracje niepotrzebnych przestrzeni nazw i obszerny komentarz z instrukcją użycia kontrolki

```
using System.Windows;
using System.Windows.Controls;


namespace Przyciski
{
    public class PrzyciskUMK : Button
    {
        static PrzyciskUMK2()
        {
            DefaultStyleKeyProperty.OverrideMetadata(typeof(PrzyciskUMK),
                                new FrameworkPropertyMetadata(typeof(PrzyciskUMK)));
        }
    }
}
```

5. Przejdźmy do pliku *Generic.xaml* w podkatalogu *Themes*. To miejsce, gdzie możemy umieścić definicję stylu, który przypisze naszej kontrolce taki szablon, jaki zechcemy. Możemy wobec tego dowolnie zmienić jej wygląd, choćby używając szablonu z obracającym się tłem z poprzedniego rozdziału (pamiętaj o zdefiniowaniu używanych w nim pędzli). My jednak poprzestaniemy na prostej kontrolce, analogicznej do tej z przykładu kontrolki użytkownika w poprzednim podrozdziale (listing 14.3).

Listing 14.3. Zasoby biblioteki kontrolek custom control ze stylem przypisującym szablon kontrolce

```
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:Przyciski">
    <Style TargetType="{x:Type local:PrzyciskUMK}">
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="{x:Type local:PrzyciskUMK}">
                    <StackPanel Orientation="Horizontal"
                                HorizontalAlignment="Center" VerticalAlignment="Center">
                            <StackPanel.Background>
                                <LinearGradientBrush StartPoint="0,0.5"
                                                     EndPoint="1,0.5">
                                    <GradientStop Color="Blue" Offset="0.0" />
                                    <GradientStop Color="Navy" Offset="1.0" />
                                </LinearGradientBrush>
```

```
                        </StackPanel.Background>
                        <Image Width ="70" Height="70" Margin="10,10,10,10"
                               Source="logo.gif" />
                        <TextBlock FontSize="16" Foreground="White"
                                   Margin="0,10,10,10">
                        <Run Foreground="Yellow">Uniwersytet</Run><LineBreak/>
                        Mikołaja<LineBreak/>
                        Kopernika
                        </TextBlock>
                    </StackPanel>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
</ResourceDictionary>
```

6. W kodzie XAML jest element `Image`, który odwołuje się do pliku *logo.gif* z rysunkiem. Dodajemy go do projektu biblioteki.
7. Aby użyć przycisku w aplikacji, najpierw dodajemy referencję do jego biblioteki *Przyciski*.
8. Następnie w kodzie XAML okna (plik *MainWindow.xaml*) dodajemy przestrzeń nazw *Przyciski*, wskazując jej zespół (ang. *assembly*).
9. Teraz możemy dodać przycisk (listing 14.4).

Listing 14.4. Przykład użycia kontrolki

```
<Window x:Class="XamlWpf.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:XamlWpf"
        xmlns:p="clr-namespace:Przyciski;assembly=Przyciski"
        mc:Ignorable="d"
        Title="MainWindow" Height="350" Width="525">
    <StackPanel Orientation="Vertical">
        <local:PrzyciskUMK Width="200" Height="100" Margin="10,10,10,10" />
        <p:PrzyciskUMK Width="200" Height="100" Margin="0,0,0,10" />
        <Button Width="100" Height="50" Margin="0,0,0,10"
                Template="{StaticResource SzablonNiebieski}" />
        <Button Width="100" Height="50" Margin="0,0,0,10"
                Template="{StaticResource SzablonNiebieski}" />
    </StackPanel>
</Window>
```

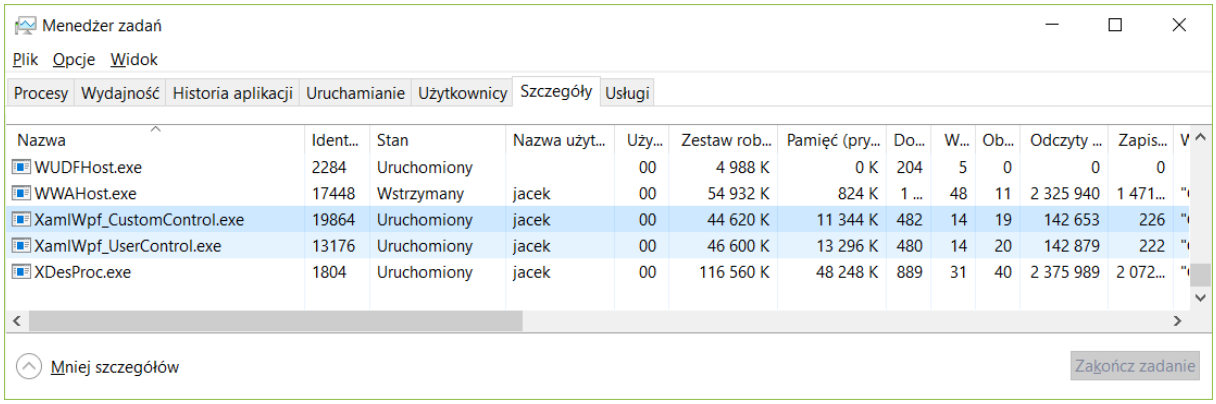
Zwróć uwagę, że nawet gdybyśmy do elementu `p:PrzyciskUMK` dodali atrybut `Background="Red"` lub `Background="{StaticResource FioletowyGradient}"`, nie spowodowałby on zmiany tła przycisku. Tło jest bowiem nadpisywane w szablonie przypisywanym wewnątrz kontrolki. Usunięcie podelementu `StackPanel.Background` z szablonu kontrolki (plik *Generic.xaml*) też nie przyniesie spodziewanego efektu — kontrolka w ogóle będzie pozbawiona tła. Żeby w szablonie użyć własności przypisywanej kontrolce, należy wykorzystać konstrukcję `TemplateBinding` (listing 14.5).

Listing 14.5. Korzystanie w szablonie z wartości przypisanych poprzez atrybuty

```
<ControlTemplate TargetType="{x:Type local:PrzyciskUMK}">
    <StackPanel Orientation="Horizontal"
                HorizontalAlignment="Center" VerticalAlignment="Center"
                Background="{TemplateBinding Background}">
        <StackPanel.Background>
            <LinearGradientBrush StartPoint="0,0.5" EndPoint="1,0.5">
                <GradientStop Color="Blue" Offset="0.0" />
                <GradientStop Color="Navy" Offset="1.0" />
            </LinearGradientBrush>
        </StackPanel.Background>
        <Image Width ="70" Height="70" Margin="10,10,10,10" Source="logo.gif" />
        <TextBlock FontSize="16" Foreground="White" Margin="0,10,10,10">
            <Run Foreground="Yellow">Uniwersytet</Run><LineBreak/>
            Mikołaja<LineBreak/>
            Kopernika
        </TextBlock>
    </StackPanel>
</ControlTemplate>
```

* * *

Często komentowanym zagadnieniem jest ilość pamięci wykorzystywana przez oba typy kontrolek. Ponieważ utworzyliśmy kontrolki o bardzo podobnej zawartości, możemy to łatwo sprawdzić. Umieściłem w oknie dziesięć kontrolek typu *user control* i sprawdziłem rozmiar aplikacji w pamięci (skompilowana w trybie *Release*). Następnie zmieniłem je na kontrolki *custom control* (wystarczy zmienić przestrzeń nazw). Okazuje się, że różnica rzeczywiście się pojawia. Mniej pamięci zajmuje drugi typ kontrolki, co jest widoczne nawet przy tak prostej zawartości, jakiej użyliśmy do testów (rysunek 14.1).



Rysunek 14.1. Menedżer zadań z widocznymi procesami dwóch wersji aplikacji testowej