

Стилізація елементів управління

Питання 4.4.

Література



Стилізація вбудованих елементів управління

- Найпростіший спосіб змінити вигляд елементу управління – налаштувати його властивості.
- Визначаючи стилі для елементів управління, можна задати їх тип за допомогою властивості TargetProperty:

```
<Button Content="Go">  
  <Button.Style>  
    <Style TargetType="{x:Type Button}">  
      <Setter Property="Foreground" Value="Green" />  
      <Setter Property="Background" Value="White" />  
    </Style>  
  </Button.Style>  
</Button>
```

Стилізація вбудованих елементів управління

- Корисна властивість класу Style – BasedOn.
- Можна розширювати стилі на базі вже існуючих.

```
<Style x:Key="TextBoxStyle" TargetType="{x:Type TextBox}">
  <Setter Property="SnapsToDevicePixels" Value="True" />
  <Setter Property="Margin" Value="0,0,0,5" />
  <Setter Property="Padding" Value="1.5,2" />
  <Setter Property="TextWrapping" Value="Wrap" />
</Style>
<Style x:Key="ReadOnlyTextBoxStyle" TargetType="{x:Type TextBox}"
  BasedOn="{StaticResource TextBoxStyle}">
  <Setter Property="IsReadOnly" Value="True" />
  <Setter Property="Cursor" Value="Arrow" />
</Style>
```

- Для розширення на основі стилю за замовчуванням:

```
<Style x:Key="ExtendedTextBoxStyle" TargetType="{x:Type TextBox}"
  BasedOn="{StaticResource {x:Type TextBox}}">
  ...
</Style>
```

Стилізація та ресурси

- Найчастіше стилі оголошуються у словниках ресурсів.
 - Властивість Resources має тип ResourceDictionary та оголошена в класі FrameworkElement.

```
<Application.Resources>
  <ResourceDictionary>
    <!-- Add resources here -->
  </ResourceDictionary>
</Application.Resources>
```

=

```
<Application.Resources>
  <!-- Add Resources here -->
</Application.Resources>
```

- При описі стилю в секції Resources можна задати значення TargetType – стиль неявно застосується до елементів коректного типу.

```
<Resources>
  <Style TargetType="{x:Type Button}">
    <Setter Property="Foreground" Value="Green" />
    <Setter Property="Background" Value="White" />
  </Style>
</Resources>
```

Стилізація та ресурси

- Неявне застосування стилю: шрифтів тощо.

```
<Resources>
  <Style TargetType="{x:Type Button}">
    <Setter Property="Foreground" Value="Green" />
    <Setter Property="Background" Value="White" />
  </Style>
</Resources>
```


- Явне застосування стилю:

```
<Resources>
  <Style x:Key="ButtonStyle">
    <Setter Property="Button.Foreground" Value="Green" />
    <Setter Property="Button.Background" Value="White" />
  </Style>
</Resources>

...
<Button Style="{StaticResource ButtonStyle}" Content="Go" />
```

Використання StaticResource та DynamicResource

- Процес пошуку для StaticResource відбувається один раз при ініціалізації.
 - При посиланні на один ресурс, який буде використано всередині іншого ресурсу, потрібний для використання ресурс необхідно оголосити заздалегідь.



```
<Style TargetType="{x:Type Button}">
  <Setter Property="Foreground" Value="{StaticResource RedBrush}" />
</Style>
<SolidColorBrush x:Key="RedBrush" Color="Red" />
```

- Інколи використання StaticResource недоречне.
 - Наприклад, потрібно оновлювати стилі під час виконання програми у відповідь на взаємодію з користувачем чи іншим кодом (зокрема, зміною теми Windows).

Використання StaticResource та DynamicResource

- Оновлення стилю при змінах у ресурсах потребує використання DynamicResource.

```
<Style TargetType="{x:Type Button}">
  <Setter Property="Foreground" Value="{DynamicResource RedBrush}" />
</Style>
<SolidColorBrush x:Key="RedBrush" Color="Red" />
```

- Не буде помилки компіляції, оскільки пошук ресурсу почнеться тільки з моменту його потреби та при кожному запиті ресурсу.
- DynamicResource суттєво погіршує продуктивність коду.

Злиття ресурсів

- Створимо кілька додаткових ресурсів

```
<Application.Resources>
  <ResourceDictionary>
    <!-- Add Resources here... -->
    <ResourceDictionary.MergedDictionaries>
      <ResourceDictionary Source="Default Styles.xaml" />
      <ResourceDictionary Source="Default Templates.xaml" />
    </ResourceDictionary.MergedDictionaries>
    <!-- ... or add resources here, but not in both locations -->
  </ResourceDictionary>
</Application.Resources>
```

- Можна задати значення `ResourceDictionary.MergedDictionaries` або до, або після локально оголошених ресурсів всередині `ResourceDictionary`.
 - Within this property, we can declare another `ResourceDictionary` element for each external resource file that we want to merge and specify its location using a Uniform Resource Identifier (URI) in the `Source` property.

Злиття ресурсів

- If our external resource files reside in our startup project with our App.xaml file, we can reference them with relative paths.
- Otherwise, we will need to use the Pack URI notation.

```
<ResourceDictionary  
  Source="pack://application:,,,/CompanyName.ApplicationName.Resources;  
  component/Styles/Control Styles.xaml" />
```

- When merging resource files, it is important to understand how naming conflicts will be resolved.
- Although the x:Key directives that we set on our resources must each be unique within their declared resource dictionary, it is perfectly legal to have duplicated key values within separate resource files.
- As such, there is an order of priority that will be followed in these cases.

Конфлікти з назвами ключів

- Один ресурс в окремому проєкті (червоний колір, обов'язково імпортувати збірку System.Xaml , інші – в локальних файлах Default Styles.xaml (синій колір) та Default Styles 2.xaml (помаранчевий колір).

```
<Application.Resources>
  <ResourceDictionary>
    <ResourceDictionary.MergedDictionaries>
      <ResourceDictionary Source="Default Styles.xaml" />
      <ResourceDictionary Source="Default Styles 2.xaml" />
      <ResourceDictionary Source="pack://application:,,,/
        CompanyName.ApplicationName.Resources;
        component/Styles/Control Styles.xaml" />
    </ResourceDictionary.MergedDictionaries>
    <SolidColorBrush x:Key="Brush" Color="Green" />
    ...
  </ResourceDictionary>
</Application.Resources>
```

Конфлікти з назвами ключів

- let's imagine that we have this in the XAML of one of our Views:

```
<Button Content="Go">
  <Button.Resources>
    <SolidColorBrush x:Key="Brush" Color="Cyan" />
  </Button.Resources>
  <Button.Style>
    <Style TargetType="{x:Type Button}">
      <Setter Property="Foreground" Value="{StaticResource Brush}" />
    </Style>
  </Button.Style>
</Button>
```

- Also, let's assume that we have this in the local resources of that file:

```
<UserControl.Resources>
  <SolidColorBrush x:Key="Brush" Color="Purple" />
</UserControl.Resources>
```

Конфлікти з назвами ключів

- При запуску додатку текст кнопки буде кольору суан, оскільки головне правило області видимості ресурсу: the highest priority resource that will be used will always be the most locally declared resource.
 - Якщо видалити чи закоментувати local brush declaration, the button text would then become purple when the application was next run.
 - Якщо видалити local purple brush resource from the control's Resources section, the application resources would be searched next in an attempt to resolve the Brush resource key.
- Наступне загальне правило: the latest declared resource will be resolved.
 - In this way, the button text would then become green, because of the locally declared resource in the App.xaml file, which would override the values from the merged dictionaries.
 - However, if this green brush resource was now removed, an interesting thing will happen.
 - Given the recently stated rules, we might expect that the button text would then be set to red by the resource file from the referenced assembly.
 - Instead, it will be set to orange by the resource in the Default Styles 2.xaml file.

Реакція на зміни

- У WPF є кілька класів `Trigger`, які дозволяють (зазвичай тимчасово) вносити зміни в елементи управління.
 - Вони породжені від класу `TriggerBase` та успадковують властивості `EnterActions` та `ExitActions` (присвоюють об'єкти `TriggerAction`).
 - Більшість тригерів містять властивість `Setters`, проте не клас `EventTrigger`.
 - Замість неї використовується властивість `Actions`, яка дозволяє задати кілька об'єктів `TriggerAction` при активації тригера.
- Також клас `EventTrigger`, на відміну від інших тригерів, не має поняття `state termination`.
 - Відмінити дію, коли вже не діє умова спрацювання тригера, неможливо.
 - Такими умовами є маршрутизовані події (об'єкти `RoutedEvent`)

Простий приклад використання EventTrigger

```
<Rectangle Width="300" Height="300" Fill="Orange">
  <Rectangle.Triggers>
    <EventTrigger RoutedEvent="Loaded">
      <BeginStoryboard>
        <Storyboard Storyboard.TargetProperty="Width">
          <DoubleAnimation Duration="0:0:1" To="50" AutoReverse="True"
            RepeatBehavior="Forever" />
        </Storyboard>
      </BeginStoryboard>
    </EventTrigger>
  </Rectangle.Triggers>
</Rectangle>
```

- Умова тригера виконується, коли спрацьовує подія FrameworkElement.Loaded.
 - Застосована дія (action) запускає оголошену анімацію.
 - Клас BeginStoryboard розширяє клас TriggerAction
 - Дія неявно додається в TriggerActionCollection об'єкта EventTrigger
 - Для явного додавання в EventTrigger ще вкладається EventTrigger.Actions

Інші тригери

- Крім `EventTrigger`, доступні класи `Trigger`, `DataTrigger`, `MultiTrigger` та `MultiDataTrigger`.
 - Дозволяють встановлювати значення властивостей або анімацій елемента управління, коли виконуються одна або кілька умов.
 - На відміну від `EventTrigger`, мають деякі обмеження на використання в колекціях тригерів.
- Кожен елемент управління, породжений від `FrameworkElement`, має властивість `Triggers` типу `TriggerCollection`.
 - При спробі оголосити тригер дозволяються тільки `EventTrigger`-и.
- При визначенні `ControlTemplate` маємо доступ до колекції `ControlTemplate.Triggers`.
 - Для інших вимог можемо оголосити інші тригери в колекції `Style.Triggers`.
 - Тригери зі стилів мають вищий пріоритет за тригери з шаблонів.

Тригер на базі класу Trigger

- Вимоги до використання такого тригера:
 - Відповідна властивість повинна бути властивістю залежності. Unlike the EventTrigger class, the other triggers do not specify actions to be applied when the trigger condition is met, but property setters instead.

```
<Button Content="Go">
  <Button.Style>
    <Style TargetType="{x:Type Button}">
      <Setter Property="Foreground" Value="Black" />
      <Style.Triggers>
        <Trigger Property="IsMouseOver" Value="True">
          <Setter Property="Foreground" Value="Red" />
        </Trigger>
      </Style.Triggers>
    </Style>
  </Button.Style>
</Button>
```

























