

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`.
 - Всередині цієї властивості можна оголосити інший елемент `ResourceDictionary` для кожного файлу зовнішніх ресурсів, які бажано поєднати (merge) та задати місце розташування за допомогою Uniform Resource Identifier (URI) у властивості `Source`.

Злиття ресурсів

- Якщо файли зовнішніх ресурсів описуються разом з App.xaml, отримати доступ до них можна за допомогою відносних шляхів.
- Інакше необхідно використовувати Pack URI notation.

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

- При злитті ресурсів важливо розуміти, як вирішувати конфлікти.
- Абсолютно легально мати дублікати ключів у різних ресурсних файлах.
- Присутні пріоритети, відповідно до яких обирається ключ з дублікатів.

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

- Нехай матимемо такий XAML-код для одного з 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>
```

- Також передбачаємо такий код у локальних ресурсах цього файлу:

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

Конфлікти з назвами ключів

- При запуску додатку текст кнопки буде кольору cyan: ресурсом з найвищим пріоритетом є найбільш локально оголошений ресурс.
 - Якщо видалити чи закомментувати локальне оголошення пензля, текст кнопки стане фіолетовим (purple) при наступному запуску програми.
 - Якщо видалити фіолетовий локальний пензель із секції Resources елемента управління, ресурси додатку будуть шукатись далі.
- Наступне загальне правило: останній оголошений ресурс will be resolved.
 - Текст кнопки стане зеленим, оскільки локально оголошений ресурс в App.xaml переозначить значення з об'єднаних словників.
 - Після видалення зеленого пензля колір стане помаранчевим (orange) відповідно до ресурсу з файлу Default Styles 2.xaml.

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 триггеру:
 - Відповідна властивість повинна бути властивістю залежності.
 - На відміну від класу EventTrigger, інші тригери задають не дії для застосування при спрацюванні, а через сеттери властивостей.

```
<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>
```

Клас DataTrigger

- Та ж прив'язка
 - Потрібно задати значення властивості Binding
 - Для досягнення аналогічної функціональності, як і property trigger, також потрібно присвоїти джерелу прив'язки елемента управління член перелічення RelativeSource.

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

Клас DataTrigger

- Рекомендація: при можливості використання простого property-тригера, який використовує властивість контрола-хоста в умові, краще використовувати клас Trigger.
- За потреби використання властивості іншого контрола або об'єкта даних в умові тригера краще брати DataTrigger.

```
<Style x:Key="TextBoxStyle" TargetType="{x:Type TextBox}">
  <Style.Triggers>
    <DataTrigger Binding="{Binding DataContext.IsEditable,
      RelativeSource={RelativeSource AncestorType={x:Type UserControl}}},
      FallbackValue=True}" Value="False">
      <Setter Property="IsReadOnly" Value="True" />
    </DataTrigger>
  </Style.Triggers>
</Style>
```

Розглянуті тригери використовували одну умову

- В одному випадку бажано використовувати один стиль, інакше може бути потрібним інший.

```
<Style x:Key="ButtonStyle" TargetType="{x:Type Button}">
  <Setter Property="Foreground" Value="Black" />
  <Style.Triggers>
    <Trigger Property="IsMouseOver" Value="True">
      <Setter Property="Foreground" Value="Red" />
    </Trigger>
    <MultiTrigger>
      <MultiTrigger.Conditions>
        <Condition Property="IsFocused" Value="True" />
        <Condition Property="IsMouseOver" Value="True" />
      </MultiTrigger.Conditions>
      <Setter Property="Foreground" Value="Green" />
    </MultiTrigger>
  </Style.Triggers>
</Style>
```

Класи MultiTrigger та MultiDataTrigger

- Можемо задавати багато Condition-елементів у колекції Conditions та багато сеттерів всередині елементу MultiTrigger.
 - Проте кожна умова повинна повертати true, щоб застосовувались сеттери чи інші тригери.
- Те ж характерно і для MultiDataTrigger.
 - Відмінності аналогічні, як і між property-тригером та data-тригером.
 - Data- та multi-data-тригери мають набагато ширший діапазон target sources, а тригери та мультитригери працюють тільки з властивостями локального елементу управління.


```

<StackPanel>
  <CheckBox Name="ShowErrors" Content="Show Errors" Margin="0,0,0,10" />
  <TextBlock>
    <TextBlock.Style>
      <Style TargetType="{x:Type TextBlock}">
        <Setter Property="Text" Value="No Errors" />
        <Style.Triggers>
          <MultiDataTrigger>
            <MultiDataTrigger.Conditions>
              <Condition Binding="{Binding IsValid}" Value="False" />
              <Condition Binding="{Binding IsChecked,
                ElementName=ShowErrors}" Value="True" />
            </MultiDataTrigger.Conditions>
            <MultiDataTrigger.Setters>
              <Setter Property="Text" Value="{Binding ErrorList}" />
            </MultiDataTrigger.Setters>
          </MultiDataTrigger>
        </Style.Triggers>
      </Style>
    </TextBlock.Style>
  </TextBlock>
  ...
</StackPanel>

```

- Явно оголошуємо властивість-колекцію `Setters` та визначаємо сеттер в ній.
- Це не обов'язково, можна неявно додавати сеттер у ту ж колекцію без її оголошення.

Властивості EnterActions та ExitActions класу TriggerBase

```
<TextBox Width="200" Height="28">
  <TextBox.Style>
    <Style TargetType="{x:Type TextBox}">
      <Setter Property="Opacity" Value="0.25" />
      <Style.Triggers>
        <Trigger Property="IsMouseOver" Value="True">
          <Trigger.EnterActions>
            <BeginStoryboard>
              <Storyboard Storyboard.TargetProperty="Opacity">
                <DoubleAnimation Duration="0:0:0.25" To="1.0" />
              </Storyboard>
            </BeginStoryboard>
          </Trigger.EnterActions>
          <Trigger.ExitActions>
            <BeginStoryboard>
              <Storyboard Storyboard.TargetProperty="Opacity">
                <DoubleAnimation Duration="0:0:0.25" To="0.25" />
              </Storyboard>
            </BeginStoryboard>
          </Trigger.ExitActions>
        </Trigger>
      </Style.Triggers>
    </Style>
  </TextBox.Style>
</TextBox>
```

Дозволяють задати один або кілька об'єктів TriggerActions, що будуть застосовані при активації/деактивації тригера відповідно.

- Умова Trigger пов'язана з властивістю IsMouseOver елементу управління TextBox.
- Оголошення анімацій у властивостях EnterActions та ExitActions за використання властивості IsMouseOver = 2 елементам EventTrigger (для MouseEnter та MouseLeave).

Дякую за увагу!