#14 TECHARTIKEL

WPF STYLING UND CONTROLTEMPLATES

05.02.2025

Lena Böhm

Inhalt

1. WPF Styling

1.1 Die Struktur eines Styles

1.2 Globale Styles und Ressourcen

1.3 Lokale Styles

1.4 Konditionales Styling

2. ControlTemplates in WPF

2.1 Die Struktur eines ControlTemplates

2.2 Der Unterschied zwischen Styles und ControlTemplates

2.3 Trigger in ControlTemplates

3. Fazit

4. Quellen

WPF Styling und ControlTemplates

Windows Presentation Foundation **(WPF)** ist eine mächtige Benutzeroberflächen-Technologie von Microsoft, die es Entwicklern ermöglicht, komplexe, skalierbare und vielfältige Windows Desktop Apps zu erstellen [1]. Ein zentrales Konzept in WPF, das für die Erstellung einer benutzerfreundlichen und ästhetisch ansprechenden Anwendung, sowie für die saubere Trennung zwischen Entwickler und Designer unerlässlich ist, ist das Styling und die Verwendung von ControlTemplates.

Diese beiden Mechanismen ermöglichen es Entwicklern, die visuelle Darstellung von Steuerelementen in ihrer Anwendung vollständig zu kontrollieren, ohne die zugrunde liegende Funktionalität der Steuerelemente zu verändern [2]. In diesem Artikel werden die grundlegenden Prinzipien des WPF Stylings und die Verwendung von ControlTemplates detailliert untersucht und gezeigt, wie sie genutzt werden können, um eine flexible und anpassbare Benutzeroberfläche zu gestalten.

1. WPF Styling

Styling in WPF ist eine Technik, mit der das Aussehen von Steuerelementen unabhängig von ihrer Funktionsweise verändert werden kann. Im Wesentlichen handelt es sich bei Styles um eine Sammlung von Eigenschaften, die auf ein bestimmtes Steuerelement angewendet werden können. Diese Eigenschaften definieren das visuelle Erscheinungsbild eines Steuerelements, wie z.B. Farben, Schriftarten, Abstände und mehr. Styles in WPF können wiederverwendet werden, was dazu beiträgt, eine einheitliche Benutzeroberfläche zu gestalten [2].

1.1 Die Struktur eines Styles

Ein WPF-Style besteht aus Settern, Triggern und Ressourcen [2], die zusammen das visuelle Erscheinungsbild eines Steuerelements bestimmen. Eine grundlegende Struktur eines Styles könnte folgendermaßen aussehen:

```
<Window x:Class="WpfApp.MainWindow" ... >
  <Window.Resources>
    <!-- Defining a Style for Buttons -->
    <Style x:Key="MyButtonStyle" TargetType="Button">
      <Setter Property="Background" Value="LightBlue"/>
      <Setter Property="FontSize" Value="16"/>
      <Setter Property="Margin" Value="10"/>
      <Setter Property="Padding" Value="15,10"/>
    </Style>
  </Window.Resources>
  <Grid>
    <Button Style="{StaticResource MyButtonStyle}" Content="Click Me" />
  </Grid>
</Window>
```

1.2 Globale Styles und Ressourcen

Im oben aufgeführten Beispiel wird ein Style für das Button-Steuerelement definiert. Die Eigenschaften wie Background, FontSize, Margin und Padding werden gesetzt, um das Aussehen explizit festzulegen. Der Style wird dann auf den Button angewendet, indem für den Button die Style-Eigenschaft auf den definierten Key (MyButtonStyle) gesetzt wird.

Um auf den Key zugreifen zu können, wird auf die unter Window deklarierte Ressource zugegriffen, wo der Style definiert ist. Eine Ressource in WPF ist „ein benamtes Datenpaket, das getrennt vom Code definiert wird und fest mit der Applikation oder darin enthaltenen Komponenten zusammenhängt“. Styles, die in global verfügbaren Ressourcen, wie z.B. in Window.Resources definiert werden, sind dementsprechend global verfügbar. Styles, die innerhalb eines UserControls definiert werden, sind folgemäßig nur innerhalb dieses UserControls verwendbar [3].

1.3 Lokale Styles

Alternativ können all die Eigenschaften, die innerhalb des Styles gesetzt werden, theoretisch auch direkt auf dem Steuerelement gesetzt werden:

```
<Button Style="{StaticResource MyButtonStyle}" Content="Click Me"
Background=”LightBlue”
FontSize=”16”
…
/>
```

Wenn Styles direkt auf dem Steuerelement gesetzt werden, können sie nicht wiederverwendet werden. Wenn das gleiche Styling auf mehreren Steuerelementen direkt angewendet wird, muss eine globale Änderung in diesem Styling (z.B. die Farbe) auf all diesen Steuerelementen einzeln angepasst werden. Hier ist die Gefahr groß, dass etwas übersehen wird. Daher ist das Styling über einen globalen Style deutlich vorteilhafter.

1.4 Konditionales Styling

Um das Styling eines Elements noch benutzerfreundlicher und intuitiver zu machen, können Properties konditional verändert werden über das Verwenden von Triggern:

```
<Style x:Key="MyButtonStyle" TargetType="Button">
      <Setter Property="Background" Value="LightBlue"/>
      <Setter Property="FontSize" Value="16"/>
      <Setter Property="Margin" Value="10"/>
      <Setter Property="Padding" Value="15,10"/>
<Style.Triggers>
   <Trigger Property="IsMouseOver" Value="True">
       <Setter Property="Background" Value="Red" />
   </Trigger>
</Style.Triggers>
</Style>
```

In diesem Beispiel wird der Button rot, sobald der Nutzer mit dem Mauszeiger darüber schwebt. Ausschlaggebend ist hier das Element *Trigger*. Darüber können für eine Vielzahl von Properties Bedingungen zu deren Styling gesetzt werden.

Zusammenfassung: Die wichtigsten Komponenten eines Styles

- **TargetType**: Der TargetType gibt an, auf welchen Steuerelemente-Typ der Style angewendet wird. Im obigen Beispiel ist das „Button“.

- **Setters**: Ein Setter ist eine Zuordnung zwischen einer Eigenschaft und einem Wert. In dem obigen Beispiel werden verschiedene Eigenschaften wie Background, FontSize und Margin gesetzt.

- **Trigger**: Ein Trigger stellt eine Bedingung für das Setzen einer Property dar. Wird ein Trigger ausgelöst, überschreibt der im Trigger hinterlegte Wert den Standardwert der entsprechenden Property.

- **Resources**: Styles werden häufig innerhalb von Ressourcen definiert. Dies ermöglicht es, den Style in der gesamten Anwendung wiederzuverwenden.

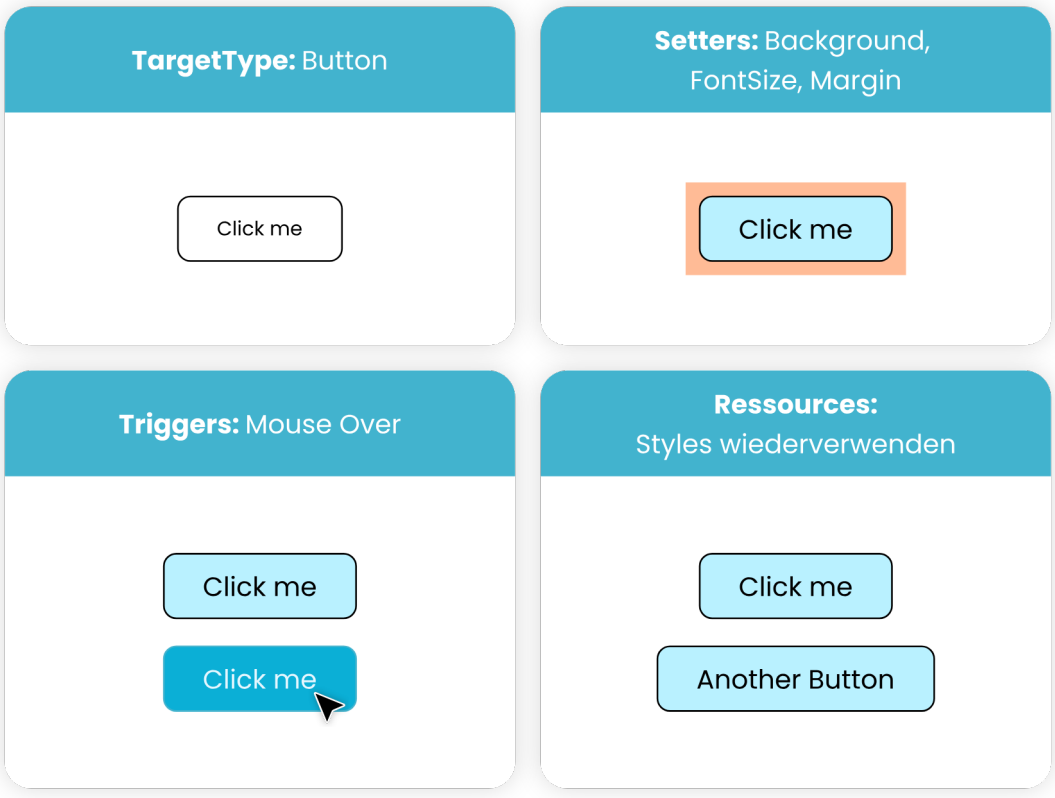

Abbildung 1: Die Komponenten eines Styles

2. ControlTemplates in WPF

Während es über Styles möglich ist, das Aussehen von Steuerelementen aus Basis des Standard-Aussehens zu *verändern*, kann über das Setzen eines ControlTemplates das gesamte Aussehen eines Steuerelements *ausgetauscht* werden [3].

2.1 Die Struktur eines ControlTemplates

Ein ControlTemplate definiert die Struktur des Steuerelements, das visuelle Layout und die inneren Komponenten. Dies bedeutet, dass man mit einem ControlTemplate nicht nur die Eigenschaften eines Steuerelements wie bei einem Style ändert, sondern die gesamte visuelle Struktur – einschließlich aller inneren Elemente und deren Anordnung.

Hier ein Beispiel eines ControlTemplate für einen Button:

```
<Button Click="OnButtonClickedCommand”>
    <Button.Template>
      <ControlTemplate>
        <Border x:Name="ImageButtonBorder"
BorderBrush="Transparent"
            BorderThickness="4">
          <Image Source="Image.png" Height="50"/>
        </Border>
<ControlTemplate.Triggers>
          <Trigger Property="IsMouseOver" Value="True">
            <Setter Property="BorderBrush" TargetName=" ImageButtonBorder "
                Value="Red"/>
          </Trigger>
</ControlTemplate.Triggers>
      </ControlTemplate>
    </Button.Template>
  </Button>
```

In diesem Beispiel wird die Standarddarstellung eines Buttons vollständig durch ein benutzerdefiniertes ControlTemplate ersetzt. Der Button besteht nun aus einem Border-Element, das den Hintergrund und die Umrandung des Buttons darstellt, und einem Bild, das nun den Inhalt des Buttons darstellt.

Das Template wird über das „Template“ Property des Buttons gesetzt. Im angefügten Beispiel geschieht dies lokal direkt auf dem Steuerelement. Es ist auch hier möglich ein ControlTemplate über Ressourcen zu definieren und über einen Key an den entsprechenden Steuerelementen zu setzen. Weiter ist es möglich einen Style zu definieren, der wiederum ein ControlTemplate festlegt.

2.2 Der Unterschied zwischen Styles und ControlTemplates

- **Styles**: Definieren die visuellen Eigenschaften eines Steuerelements wie Hintergrundfarbe, Schriftgröße, Rahmen und andere Eigenschaften. Sie können das Aussehen ändern, aber sie ändern nicht die Struktur des Steuerelements.
- **ControlTemplates**: Erlauben es, die komplette visuelle Struktur eines Steuerelements zu verändern. Sie definieren, wie das Steuerelement aufgebaut ist, und ersetzen damit das Standardlayout eines Steuerelements [3].

Styles	ControlTemplates
• Fokus auf visuellen Eigenschaften wie Farben, Schriftarten, Abstände • Keine Veränderung der Struktur des Steuerelements	• Veränderung der gesamten Struktur des Steuerelements, einschl. innerer Komponenten und Layouts • Erlaubt die vollständige Neugestaltung des Steuerelements, z.B. durch das Ersetzen des Standardlayouts mit benutzerdefinierten Elementen wie Borders und Bildern

Abbildung 2: Der Unterschied zwischen Styles und ControlTemplates

2.3 Trigger in ControlTemplates

Eine der mächtigsten Funktionen von ControlTemplates sind Trigger, die es ermöglichen, auf Änderungen des Zustands eines Steuerelements zu reagieren und die Darstellung dynamisch zu ändern [4].

Auch in dem Beispiel (siehe oben) wird mit einem Trigger gearbeitet: Sobald der Nutzer über den Button fährt, wird die Border rot eingefärbt. Der Button hat nun die Erscheinung eines umrahmten Bildes. Die dahinterliegende Logik wird weiterhin ausgeführt, wenn der Button angeklickt wird.

Fazit

WPF bietet große Flexibilität bei der Gestaltung von Benutzeroberflächen, und die Verwendung von Styles und ControlTemplates ist der Schlüssel, um diese Flexibilität voll auszuschöpfen. Styles ermöglichen es, das Aussehen von Steuerelementen auf einfache Weise zu ändern, während ControlTemplates eine tiefere Kontrolle über das visuelle Layout eines Steuerelements bieten und es ermöglichen, die Struktur und das Verhalten von Steuerelementen vollständig zu ersetzen. Durch den effektiven Einsatz dieser Mechanismen kann eine Anwendung nicht nur funktional, sondern auch benutzerfreundlich und ästhetisch ansprechend gestaltet werden.

Quellen

1. WPF 4.5 Unleashed, Adam Nathan, 2014
2. Essential Windows Presentation Foundation (WPF), Chris Anderson, 2007
3. Programming WPF: Building Windows UI with Windows Presentation Foundation, Chris Sells, Ian Griffiths, 2007
4. Illustrated WPF, Daniel Solis, 2010