Kostenloses eBook

LERNEN

wpf

Free unaffiliated eBook created from
Stack Overflow contributors.

#wpf

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: wpf

It is an unofficial and free wpf ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official wpf.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit wpf

Bemerkungen

WPF (Windows Presentation Foundation) ist die von Microsoft empfohlene Präsentationstechnologie für klassische Windows-Desktopanwendungen. WPF sollte nicht mit UWP (Universal Windows Platform) verwechselt werden, obwohl Ähnlichkeiten zwischen den beiden bestehen.

WPF unterstützt datengesteuerte Anwendungen mit einem starken Fokus auf Multimedia, Animation und Datenbindung. Schnittstellen werden mit einer Sprache erstellt, die als XAML (eXtensible Application Markup Language), eine Ableitung von XML, bezeichnet wird. XAML unterstützt WPF-Programmierer bei der Trennung von visuellem Design und Schnittstellenlogik.

Im Gegensatz zu seinem Vorgänger Windows Forms verwendet WPF ein Boxmodell, um alle Elemente der Benutzeroberfläche zu gestalten. Jedes Element hat eine Höhe, Breite und Ränder und wird relativ zum übergeordneten Element auf dem Bildschirm angezeigt.

WPF steht für Windows Presentation Foundation und ist auch unter dem Codenamen Avalon bekannt. Es ist ein grafisches Framework und Teil von Microsofts .NET Framework. WPF ist in Windows Vista, 7, 8 und 10 vorinstalliert und kann unter Windows XP und Server 2003 installiert werden.

Versionen

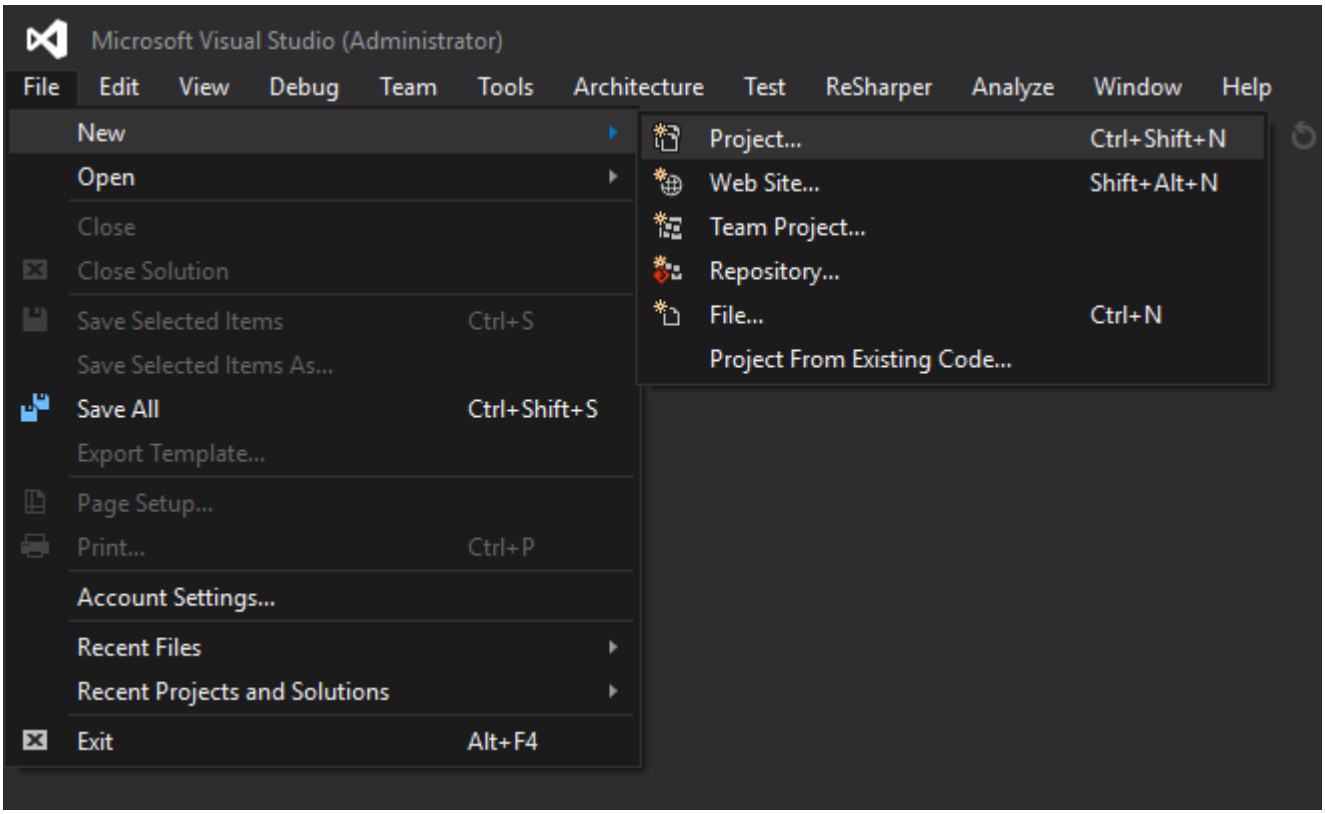
Version 4.6.1 - Dezember 2015

Examples

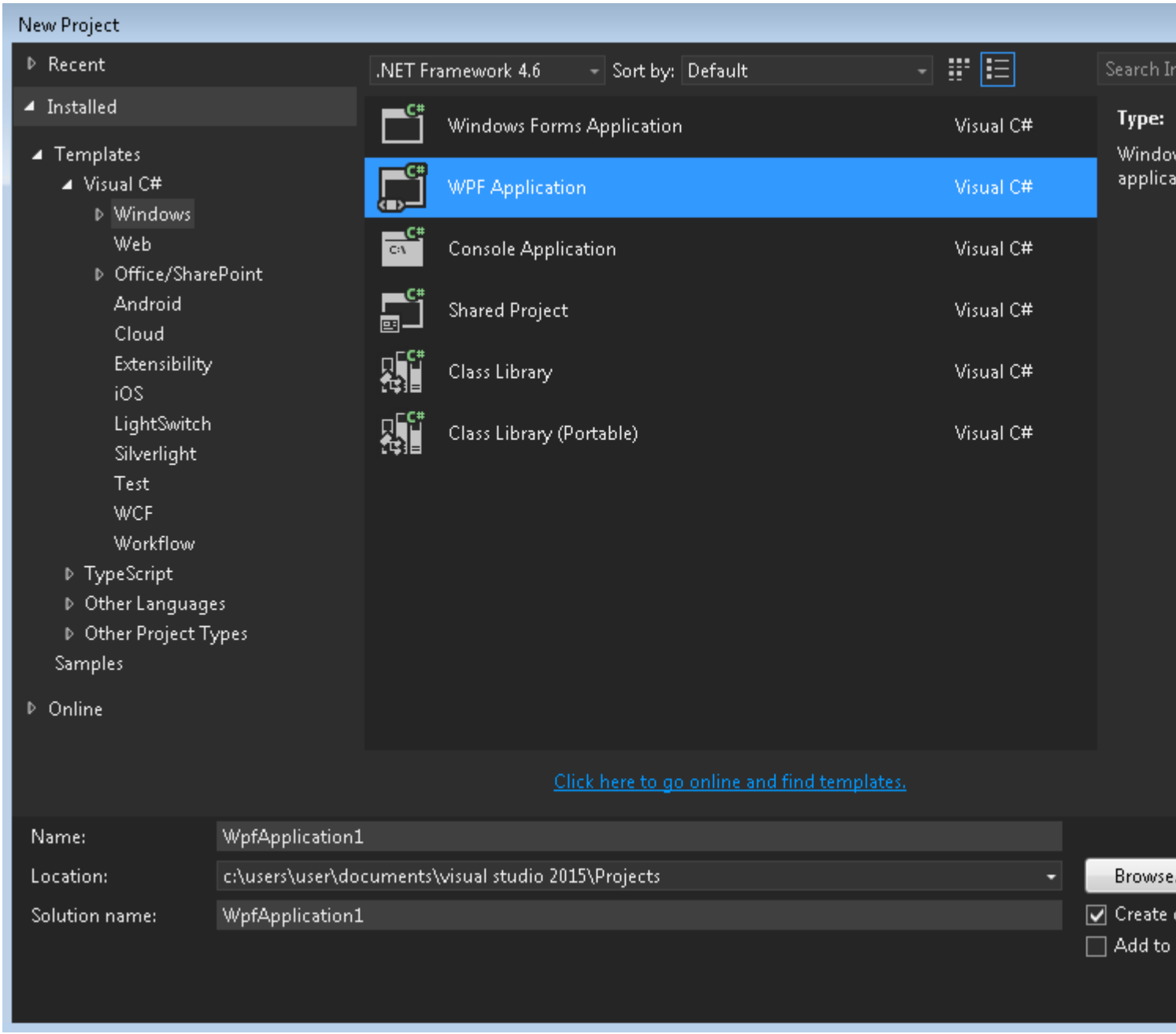
Hallo Weltanwendung

So erstellen Sie ein neues WPF-Projekt in Visual Studio und führen es aus:

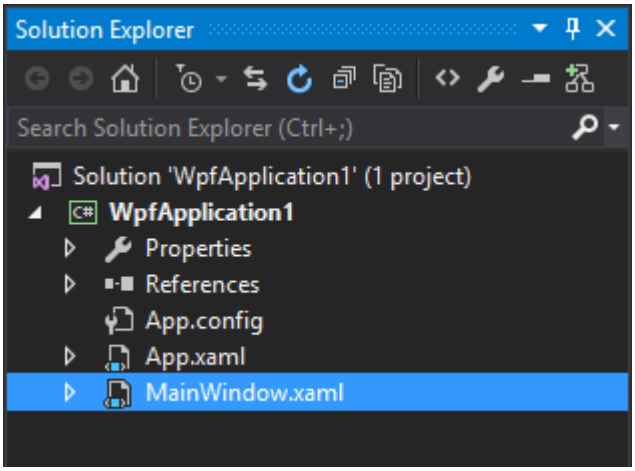
1. Klicken Sie auf **Datei → Neu → Projekt**



2. Wählen Sie die Vorlage aus, indem Sie auf **Vorlagen → Visuelle C # → Windows → WPF-Anwendung** klicken und OK drücken.



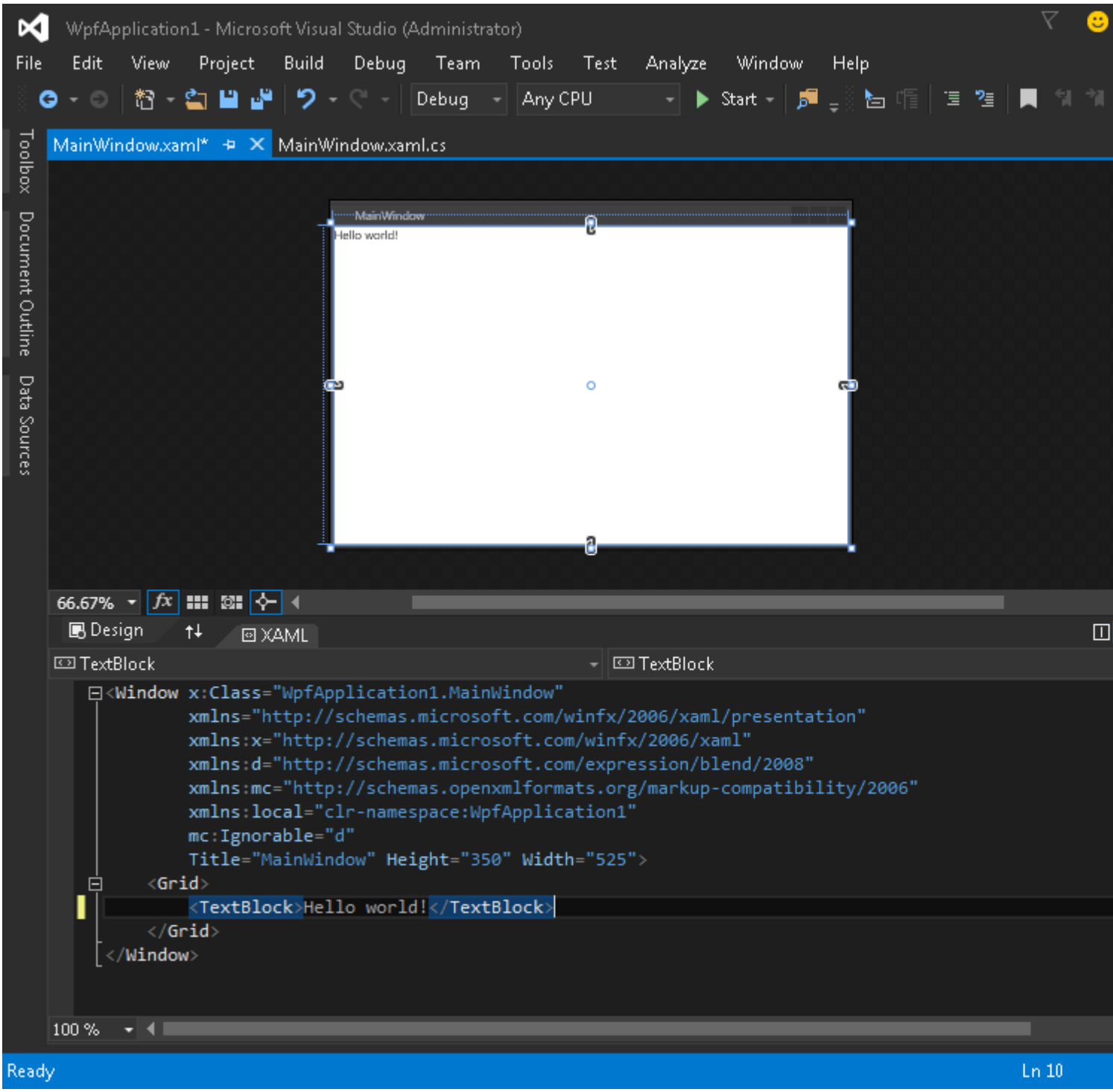
3. Öffnen **MainWindow.xaml** Datei in *Projektmappen* - *Explorer* (wenn Sie sehen *Mappen* - *Explorer* - Fenster nicht, öffnen Sie es , indem Sie **Ansicht → Mappen** - **Explorer** klicken):



4. *Fügen Sie* im *XAML*- Abschnitt (standardmäßig unter *Entwurfsabschnitt*) diesen Code hinzu

```
<TextBlock>Hello world!</TextBlock>
```

innerhalb des `Grid` Tags:

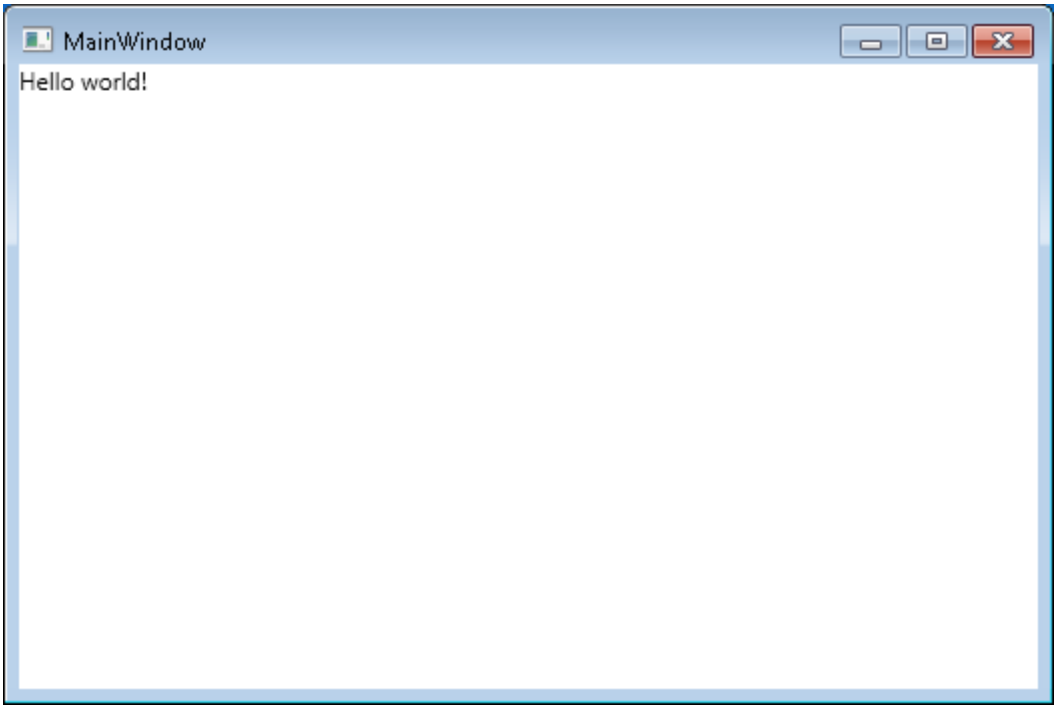


Code sollte folgendermaßen aussehen:

```
<Window x:Class="WpfApplication1.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:WpfApplication1"
        mc:Ignorable="d"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <TextBlock>Hello world!</TextBlock>
    </Grid>
</Window>
```

5. Führen Sie die Anwendung aus, indem Sie die Taste `F5` drücken oder auf das Menü **Debug → Debugging starten** klicken. Es sollte so aussehen:



Erste Schritte mit wpf online lesen: https://riptutorial.com/de/wpf/topic/820/erste-schritte-mit-wpf

Kapitel 2: "Half the Whitespace" Konstruktionsprinzip

Einführung

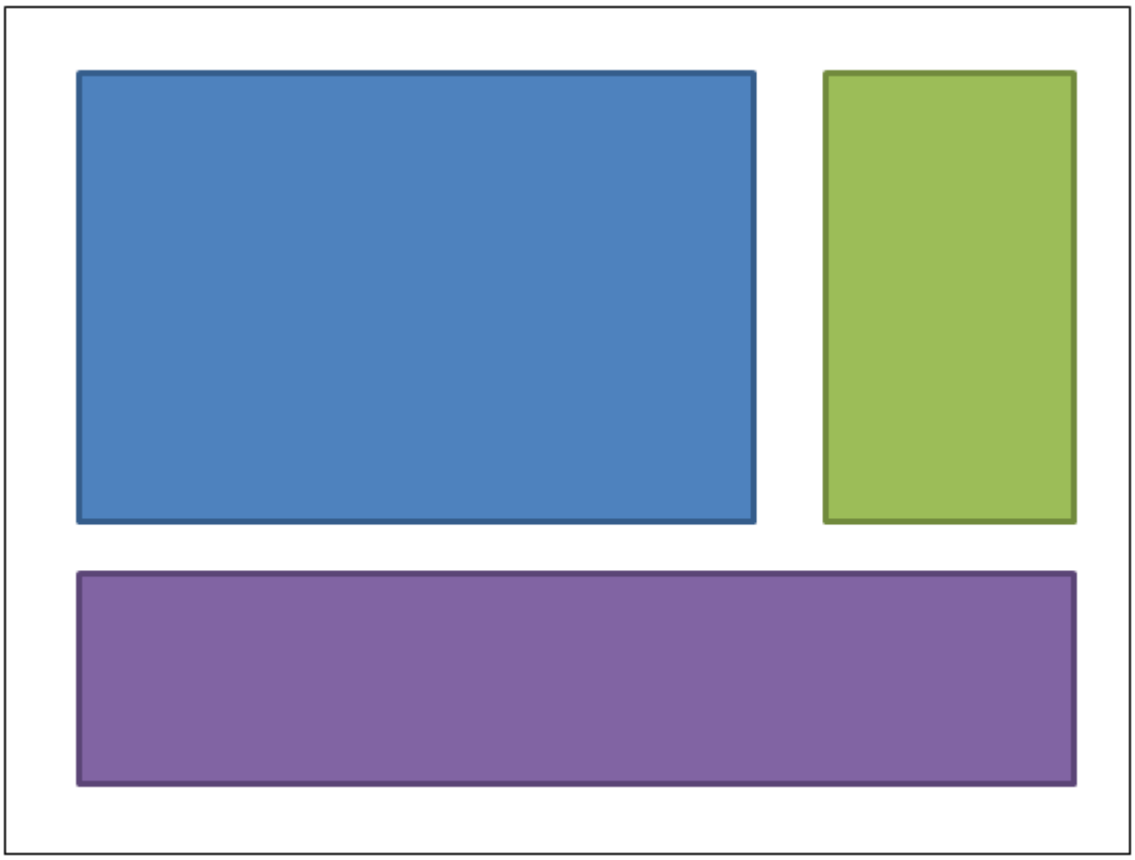
Bei der Anordnung von Steuerelementen können bestimmte Werte in den Rändern und Auffüllungen leicht hartcodiert werden, um die Einstellungen an das gewünschte Layout anzupassen. Durch das Festcodieren dieser Werte wird die Wartung jedoch erheblich teurer. Wenn sich das Layout ändert, was als trivial angesehen werden kann, muss viel Arbeit in die Korrektur dieser Werte gesteckt werden.

Dieses Konstruktionsprinzip reduziert die Kosten für die Wartung des Layouts, indem das Layout anders betrachtet wird.

Examples

Demonstration des Problems und der Lösung

Stellen Sie sich beispielsweise einen Bildschirm mit drei Abschnitten vor, die wie folgt angeordnet



sind:

Der blaue Kasten könnte einen Spielraum von 4,4,0,0 erhalten. Die grüne Box könnte einen Spielraum von 4,4,4,0 haben. Der violette Rahmenrand wäre 4,4,4,4. Hier ist die XAML: (Ich

verwende ein Raster, um das Layout zu erreichen; dieses Konstruktionsprinzip gilt jedoch unabhängig davon, wie Sie das Layout erreichen möchten):

```
<UserControl x:Class="WpfApplication5.UserControl1HardCoded"
             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"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300">
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="3*"/>
        <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="2*"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <Border Grid.Column="0" Grid.Row="0" Margin="4,4,0,0" Background="DodgerBlue"
BorderBrush="DarkBlue" BorderThickness="5"/>
    <Border Grid.Column="1" Grid.Row="0" Margin="4,4,4,0" Background="Green"
BorderBrush="DarkGreen" BorderThickness="5"/>
    <Border Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="1" Margin="4,4,4,4"
Background="MediumPurple" BorderBrush="Purple" BorderThickness="5"/>
</Grid>
</UserControl>
```

Stellen Sie sich nun vor, dass wir das Layout ändern möchten, um das grüne Feld links vom blauen Feld zu platzieren. Sollte einfach sein, oder? Außer, dass wir beim Verschieben der Box jetzt an den Rändern basteln müssen. Entweder können wir die Ränder des blauen Kästchens auf 0,4,4,0 ändern. oder wir könnten Blau auf 4,4,4,0 und Grün auf 4,4,0,0 ändern. Hier ist die XAML:

```
<UserControl x:Class="WpfApplication5.UserControl2HardCoded"
             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"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="3*"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="2*"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <Border Grid.Column="1" Grid.Row="0" Margin="4,4,4,0" Background="DodgerBlue"
BorderBrush="DarkBlue" BorderThickness="5"/>
        <Border Grid.Column="0" Grid.Row="0" Margin="4,4,0,0" Background="Green"
BorderBrush="DarkGreen" BorderThickness="5"/>
        <Border Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="1" Margin="4,4,4,4"
Background="MediumPurple" BorderBrush="Purple" BorderThickness="5"/>
    </Grid>
```

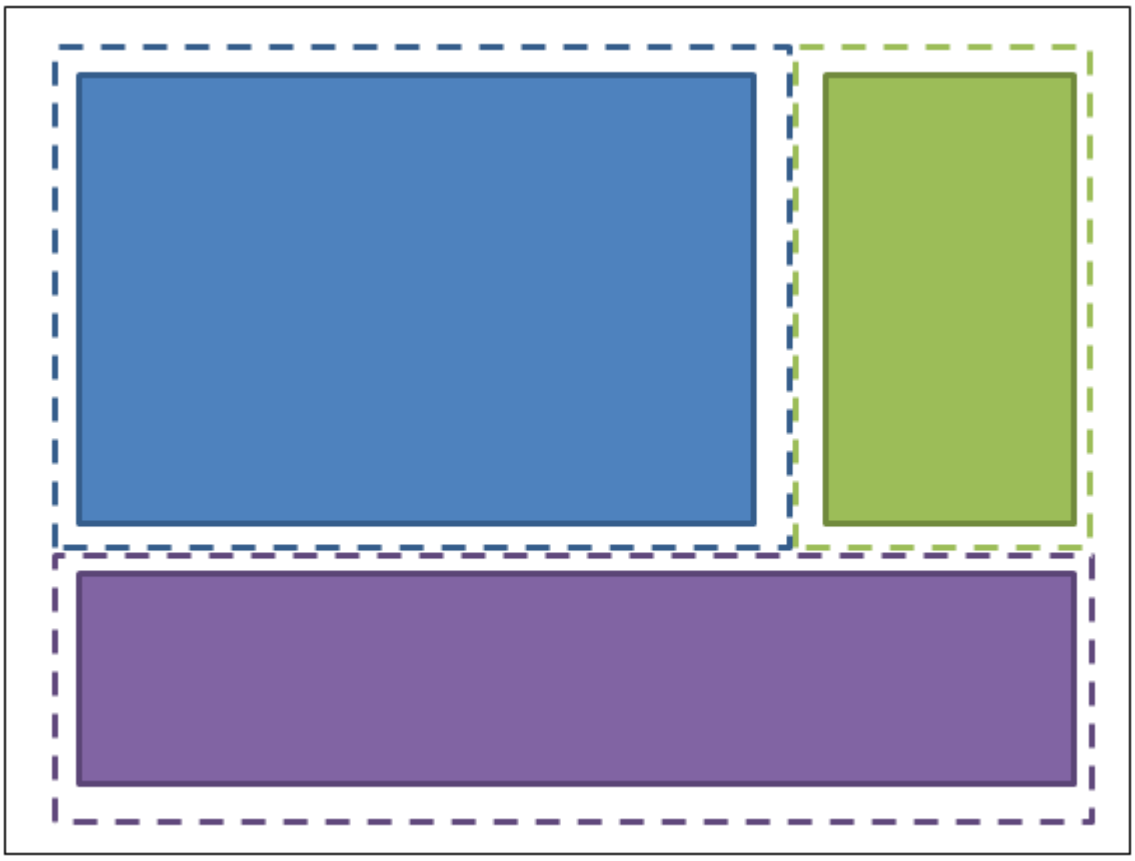
```
</UserControl>
```

Nun setzen wir die violette Box oben ein. So werden die Ränder von Blau 4,0,4,4; Grün wird 4,0,0,4.

```
<UserControl x:Class="WpfApplication5.UserControl3HardCoded"
             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"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="3*"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="*"/>
            <RowDefinition Height="2*"/>
        </Grid.RowDefinitions>

        <Border Grid.Column="1" Grid.Row="1" Margin="4,0,4,4" Background="DodgerBlue"
BorderBrush="DarkBlue" BorderThickness="5"/>
        <Border Grid.Column="0" Grid.Row="1" Margin="4,0,0,4" Background="Green"
BorderBrush="DarkGreen" BorderThickness="5"/>
        <Border Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="0" Margin="4,4,4,4"
Background="MediumPurple" BorderBrush="Purple" BorderThickness="5"/>
    </Grid>
</UserControl>
```

Wäre es nicht schön, wenn wir die Dinge verschieben könnten, so dass wir diese Werte überhaupt nicht anpassen müssen. Dies kann erreicht werden, indem der Whitespace nur auf eine andere Art und Weise betrachtet wird. Anstatt den gesamten Leerraum einem oder dem anderen Steuerelement zuzuordnen, stellen Sie sich vor, dass jeder Box die Hälfte des Whitespaces zugewiesen wird: .



Die blaue Box hat also einen Rand von 2,2,2,2; die grüne Box hat Ränder von 2,2,2,2; Die violette Box hat Ränder von 2,2,2,2. Und der Behälter, in dem sie untergebracht sind, erhält eine Polsterung (kein Rand) von 2,2,2,2. Hier ist die XAML:

```
<UserControl x:Class="WpfApplication5.UserControl1HalfTheWhitespace"
             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"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300"
             Padding="2,2,2,2">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="3*"/>
            <ColumnDefinition Width="*"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="2*"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <Border Grid.Column="0" Grid.Row="0" Margin="2,2,2,2" Background="DodgerBlue"
BorderBrush="DarkBlue" BorderThickness="5"/>
        <Border Grid.Column="1" Grid.Row="0" Margin="2,2,2,2" Background="Green"
BorderBrush="DarkGreen" BorderThickness="5"/>
        <Border Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="1" Margin="2,2,2,2"
Background="MediumPurple" BorderBrush="Purple" BorderThickness="5"/>
    </Grid>
</UserControl>
```

Versuchen wir nun, die Boxen auf die gleiche Weise wie zuvor zu verschieben ... Setzen wir die grüne Box links von der blauen Box. OK, fertig. Außerdem mussten keine Polsterungen oder Ränder geändert werden. Hier ist die XAML:

```
<UserControl x:Class="WpfApplication5.UserControl2HalfTheWhitespace"
             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"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300"
             Padding="2,2,2,2">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="3*"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="2*"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <Border Grid.Column="1" Grid.Row="0" Margin="2,2,2,2" Background="DodgerBlue"
BorderBrush="DarkBlue" BorderThickness="5"/>
        <Border Grid.Column="0" Grid.Row="0" Margin="2,2,2,2" Background="Green"
BorderBrush="DarkGreen" BorderThickness="5"/>
        <Border Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="1" Margin="2,2,2,2"
Background="MediumPurple" BorderBrush="Purple" BorderThickness="5"/>
    </Grid>
</UserControl>
```

Nun setzen wir die violette Box oben ein. OK, fertig. Außerdem mussten keine Polsterungen oder Ränder geändert werden. Hier ist die XAML:

```
<UserControl x:Class="WpfApplication5.UserControl3HalfTheWhitespace"
             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"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300"
             Padding="2,2,2,2">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="3*"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="*"/>
            <RowDefinition Height="2*"/>
        </Grid.RowDefinitions>

        <Border Grid.Column="1" Grid.Row="1" Margin="2,2,2,2" Background="DodgerBlue"
BorderBrush="DarkBlue" BorderThickness="5"/>
        <Border Grid.Column="0" Grid.Row="1" Margin="2,2,2,2" Background="Green"
BorderBrush="DarkGreen" BorderThickness="5"/>
        <Border Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="0" Margin="2,2,2,2"
Background="MediumPurple" BorderBrush="Purple" BorderThickness="5"/>
```

```
    </Grid>
</UserControl>
```

Wie man dies in echtem Code verwendet

Zu verallgemeinern , was wir oben gezeigt haben: einzelne Dinge eine feste *Marge* von „halb-the-Leerzeichen“ enthalten, und der Behälter sie gehalten werden , werden sollte eine *Polsterung* von „half-the-Leerzeichen“ haben. Sie können diese Stile in Ihrem Anwendungsressourcenwörterbuch anwenden, und Sie müssen sie nicht einmal bei den einzelnen Elementen erwähnen. So definieren Sie "HalfTheWhiteSpace":

```
<system:Double x:Key="DefaultMarginSize">2</system:Double>
<Thickness x:Key="HalfTheWhiteSpace" Left="{StaticResource DefaultMarginSize}"
Top="{StaticResource DefaultMarginSize}" Right="{StaticResource DefaultMarginSize}"
Bottom="{StaticResource DefaultMarginSize}"/>
```

Dann kann ich einen Basisstil definieren, auf den sich meine anderen Steuerelementstile beziehen: (Dieser kann auch Ihre Standard-FontFamily, FontSize usw. enthalten.)

```
<Style x:Key="BaseStyle" TargetType="{x:Type Control}">
    <Setter Property="Margin" Value="{StaticResource HalfTheWhiteSpace}"/>
</Style>
```

Dann kann ich meinen Standardstil für TextBox definieren, um diesen Rand zu verwenden:

```
<Style TargetType="TextBox" BasedOn="{StaticResource BaseStyle}"/>
```

Ich kann diese Art von Dingen für DatePicker, Labels usw. usw. (alles, was in einem Container enthalten ist) tun. Passen Sie auf, TextBlock wie folgt zu gestalten ... dieses Steuerelement wird intern von vielen Steuerelementen verwendet. Ich würde vorschlagen, dass Sie ein eigenes Steuerelement erstellen, das einfach von TextBlock abgeleitet wird. Sie können *Ihren* TextBlock so gestalten, dass der Standardrand verwendet wird. Sie sollten *Ihren* TextBlock immer dann verwenden, wenn Sie explizit einen TextBlock in Ihrer XAML verwenden.

Sie können einen ähnlichen Ansatz verwenden, um die Auffüllung auf allgemeine Container anzuwenden (z. B. ScrollViewer, Border usw.).

Wenn Sie dies getan haben, benötigen die *meisten* Steuerelemente keine Ränder und Füllungen - und Sie müssen Werte nur an Stellen angeben, an denen Sie absichtlich von diesem Konstruktionsprinzip abweichen möchten.

"Half the Whitespace" Konstruktionsprinzip online lesen: https://riptutorial.com/de/wpf/topic/9407/-half-the-whitespace--konstruktionsprinzip

Kapitel 3: Abhängigkeitseigenschaften

Einführung

Abhängigkeitseigenschaften sind ein Eigenschaftstyp, der eine CLR-Eigenschaft erweitert. Während eine CLR-Eigenschaft direkt von einem Member Ihrer Klasse gelesen wird, wird eine Dependency-Eigenschaft beim Aufrufen der GetValue () - Methode, die Ihr Objekt durch Vererbung von der Basisklasse DependencyObject erhält, dynamisch aufgelöst.

In diesem Abschnitt werden die Abhängigkeitseigenschaften aufgeschlüsselt und deren Verwendung sowohl konzeptionell als auch anhand von Codebeispielen erläutert.

Syntax

- DependencyProperty.Register (Zeichenfolgenname, Typ propertyType, Type ownerType)
- DependencyProperty.Register (Zeichenfolgenname, Typ propertyType, Type ownerType, PropertyMetadata typeMetadata)
- DependencyProperty.Register (Zeichenfolgenname, Typ propertyType, Type ownerType, PropertyMetadata typeMetadata, ValidateValueCallback validateValueCallback)
- DependencyProperty.RegisterAttached (Zeichenfolgenname, Typ propertyType, Type ownerType)
- DependencyProperty.RegisterAttached (Zeichenfolgenname, Typ propertyType, Type ownerType, PropertyMetadata typeMetadata)
- DependencyProperty.RegisterAttached (Stringname, Typ propertyType, Type ownerType, PropertyMetadata typeMetadata, ValidateValueCallback validateValueCallback)
- DependencyProperty.RegisterReadOnly (Name der Zeichenfolge, Typ propertyType, Type ownerType, PropertyMetadata typeMetadata)
- DependencyProperty.RegisterReadOnly (Zeichenfolgenname, Typ propertyType, Type ownerType, PropertyMetadata typeMetadata, ValidateValueCallback validateValueCallback)
- DependencyProperty.RegisterAttachedReadOnly (Zeichenfolgenname, Typ propertyType, Type ownerType, PropertyMetadata typeMetadata)
- DependencyProperty.RegisterAttachedReadOnly (Zeichenfolgenname, Typ propertyType, Type ownerType, PropertyMetadata typeMetadata, ValidateValueCallback validateValueCallback)

Parameter

Parameter	Einzelheiten
Name	Die `String` Darstellung des Namens der Eigenschaft
Art der Immobilie	Der `Type` der Eigenschaft, zB `typeof(int)`
ownerType	Der `Type` der Klasse, in der die Eigenschaft definiert wird, z. B. `typeof(MyControl)` oder `typeof(MyAttachedProperties)` .

Parameter	Einzelheiten
typeMetadata	Instanz von `System.Windows.PropertyMetadata` (oder einer ihrer Unterklassen), die Standardwerte definiert, die Eigenschaft geänderter Callbacks. Mit `FrameworkPropertyMetadata` können Bindungsoptionen wie `System.Windows.Data.BindingMode.TwoWay` .
validateValueCallback	Benutzerdefinierter Rückruf, der true zurückgibt, wenn der neue Wert der Eigenschaft gültig ist, andernfalls false.

Examples

Standardabhängigkeitseigenschaften

Wann verwenden?

Praktisch alle WPF-Steuerelemente nutzen die Abhängigkeitseigenschaften stark. Eine Abhängigkeitseigenschaft ermöglicht die Verwendung vieler WPF-Funktionen, die mit den Standard-CLR-Eigenschaften allein nicht möglich sind, einschließlich der Unterstützung für Stile, Animationen, Datenbindung, Wertvererbung und Änderungsbenachrichtigungen.

Die `TextBox.Text` Eigenschaft ist ein einfaches Beispiel dafür, wo eine Standardabhängigkeitseigenschaft benötigt wird. Hier wäre eine Datenbindung nicht möglich, wenn `Text` eine CLR-Standardeigenschaft war.

```
<TextBox Text="{Binding FirstName}" />
```

Wie zu definieren

Abhängigkeitseigenschaften können nur in von `DependencyObject` abgeleiteten Klassen wie `FrameworkElement` , `Control` usw. definiert werden.

Eine der schnellsten Methoden zum Erstellen einer Standardabhängigkeitseigenschaft, ohne sich an die Syntax zu erinnern, besteht darin, den "propdp" -Schnipsel zu verwenden, indem Sie `propdp` und dann die `Tabulatortaste` drücken. Ein Code-Snippet wird eingefügt, der dann an Ihre Bedürfnisse angepasst werden kann:

```
public class MyControl : Control
{
    public int MyProperty
    {
        get { return (int)GetValue(MyPropertyProperty); }
        set { SetValue(MyPropertyProperty, value); }
    }

    // Using a DependencyProperty as the backing store for MyProperty.
```

```
    // This enables animation, styling, binding, etc...
    public static readonly DependencyProperty MyPropertyProperty =
        DependencyProperty.Register("MyProperty", typeof(int), typeof(MyControl),
            new PropertyMetadata(0));
}
```

Sie sollten `Tab` durch die verschiedenen Teile des Code - Schnipsel , die notwendigen Änderungen vorzunehmen, einschließlich den Namen der Eigenschaft zu aktualisieren, Objekttyp, Klassentyp enthält, und den Standardwert.

Wichtige Konventionen

Hier sind einige wichtige Konventionen / Regeln zu beachten:

1. **Erstellen Sie eine CLR-Eigenschaft für die Abhängigkeitseigenschaft.** Diese Eigenschaft wird im Code-Behind Ihres Objekts oder von anderen Konsumenten verwendet. Es sollte `GetValue` und `SetValue` damit die Verbraucher dies nicht tun müssen.

2. **Benennen Sie die Abhängigkeitseigenschaft richtig.** Das `DependencyProperty` Feld sollte `public static readonly` . Es sollte einen Namen haben, der dem Namen der CLR-Eigenschaft entspricht und mit "Property" endet, z. B. `Text` und `TextProperty` .

3. **Fügen Sie dem Einsteller der CLR-Eigenschaft keine zusätzliche Logik hinzu.** Das Abhängigkeitseigenschaftssystem (und insbesondere XAML) verwendet die CLR-Eigenschaft nicht. Wenn Sie eine Aktion ausführen möchten, wenn sich der Wert der Eigenschaft ändert, müssen Sie einen Rückruf über `PropertyMetadata` bereitstellen:

   ```
   public static readonly DependencyProperty MyPropertyProperty =
       DependencyProperty.Register("MyProperty", typeof(int), typeof(MyControl),
           new PropertyMetadata(0, MyPropertyChangedHandler));

   private static void MyPropertyChangedHandler(DependencyObject sender,
   DependencyPropertyChangedEventArgs args)
   {
       // Use args.OldValue and args.NewValue here as needed.
       // sender is the object whose property changed.
       // Some unboxing required.
   }
   ```

Bindungsmodus

Um die Notwendigkeit der Angabe von `Mode=TwoWay` in Bindungen (ähnlich dem Verhalten von `TextBox.Text`) zu `TextBox.Text` aktualisieren Sie den Code, um `FrameworkPropertyMetadata` anstelle von `PropertyMetadata` und geben Sie das entsprechende Flag an:

```
public static readonly DependencyProperty MyPropertyProperty =
    DependencyProperty.Register("MyProperty", typeof(int), typeof(MyControl),
        new FrameworkPropertyMetadata(0,
FrameworkPropertyMetadataOptions.BindsTwoWayByDefault));
```

Wann verwenden?

Eine angefügte Eigenschaft ist eine Abhängigkeitseigenschaft, die auf jedes `DependencyObject` angewendet werden kann, um das Verhalten verschiedener Steuerelemente oder Dienste zu verbessern, die sich der Existenz der Eigenschaft bewusst sind.

Einige Anwendungsfälle für angefügte Eigenschaften umfassen:

1. Ein Elternelement iteriert durch seine Kinder und wirkt auf eine bestimmte Weise auf die Kinder ein. Das `Grid` Steuerelement verwendet beispielsweise die angefügten Eigenschaften `Grid.Row` , `Grid.Column` , `Grid.RowSpan` und `Grid.ColumnSpan` , um Elemente in Zeilen und Spalten anzuordnen.
2. Hinzufügen von Visuals zu vorhandenen Steuerelementen mit benutzerdefinierten Vorlagen, z. B. Hinzufügen von Wasserzeichen zu leeren Textfeldern, ohne dass `TextBox` Unterklasse verwendet werden `TextBox` .
3. Bereitstellung eines generischen Services oder Features für einige oder alle vorhandenen Steuerelemente, z. B. `ToolTipService` oder `FocusManager` . Diese werden im Allgemeinen als *angefügte Verhaltensweisen bezeichnet* .
4. Wenn die Vererbung nach unten erfolgt, ist der visuelle Baum erforderlich, z. B. ähnlich dem Verhalten von `DataContext` .

Dies zeigt weiter, was im `Grid` Anwendungsfall passiert:

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>

    <Label Grid.Column="0" Content="Your Name:" />
    <TextBox Grid.Column="1" Text="{Binding FirstName}" />
</Grid>
```

`Grid.Column` ist keine Eigenschaft, die in `Label` oder `TextBox` . Das `Grid` Steuerelement durchsucht die untergeordneten Elemente und ordnet sie gemäß den Werten der angefügten Eigenschaften an.

Wie zu definieren

Wir werden `Grid` für dieses Beispiel verwenden. Die Definition von `Grid.Column` wird unten gezeigt, der `DependencyPropertyChangedEventHandler` ist jedoch der Kürze `Grid.Column` ausgeschlossen.

```
public static readonly DependencyProperty RowProperty =
    DependencyProperty.RegisterAttached("Row", typeof(int), typeof(Grid),
        new FrameworkPropertyMetadata(0, ...));
```

```
public static void SetRow(UIElement element, int value)
{
    if (element == null)
        throw new ArgumentNullException("element");

    element.SetValue(RowProperty, value);
}

public static int GetRow(UIElement element)
{
    if (element == null)
        throw new ArgumentNullException("element");

    return ((int)element.GetValue(RowProperty));
}
```

Da die angefügten Eigenschaften an eine Vielzahl von Elementen angehängt werden können, können sie nicht als CLR-Eigenschaften implementiert werden. Stattdessen wird ein Paar statischer Methoden eingeführt.

Im Gegensatz zu Standardabhängigkeitseigenschaften können angefügte Eigenschaften auch in Klassen definiert werden, die nicht von `DependencyObject` abgeleitet sind.

Die gleichen Namenskonventionen gelten auch für reguläre Abhängigkeitseigenschaften: Die Abhängigkeitseigenschaft `RowProperty` verfügt über die entsprechenden Methoden `GetRow` und `SetRow` .

Vorsichtsmaßnahmen

Wie auf MSDN dokumentiert :

> Obwohl die Vererbung von Eigenschaftswerten für nicht angefügte Abhängigkeitseigenschaften zu funktionieren scheint, ist das Vererbungsverhalten für eine nicht angefügte Eigenschaft über bestimmte Elementgrenzen in der Laufzeitstruktur nicht definiert. Verwenden Sie immer RegisterAttached, um Eigenschaften zu registrieren, in denen Sie Inherits in den Metadaten angeben.

Schreibgeschützte Abhängigkeitseigenschaften

Wann verwenden?

Eine schreibgeschützte Abhängigkeitseigenschaft ähnelt einer normalen Abhängigkeitseigenschaft. Sie ist jedoch so strukturiert, dass ihr Wert nicht außerhalb des Steuerelements festgelegt werden kann. Dies funktioniert gut, wenn Sie über eine Eigenschaft verfügen, die rein informativ für Verbraucher ist, z. B. `IsMouseOver` oder `IsKeyboardFocusWithin` .

Wie zu definieren

Genau wie Standardabhängigkeitseigenschaften muss eine schreibgeschützte Abhängigkeitseigenschaft für eine Klasse definiert werden, die von `DependencyObject` .

```
public class MyControl : Control
{
    private static readonly DependencyPropertyKey MyPropertyPropertyKey =
        DependencyProperty.RegisterReadOnly("MyProperty", typeof(int), typeof(MyControl),
            new FrameworkPropertyMetadata(0));

    public static readonly DependencyProperty MyPropertyProperty =
MyPropertyPropertyKey.DependencyProperty;

    public int MyProperty
    {
        get { return (int)GetValue(MyPropertyProperty); }
        private set { SetValue(MyPropertyPropertyKey, value); }
    }
}
```

Die gleichen Konventionen, die für reguläre Abhängigkeitseigenschaften gelten, gelten auch hier, jedoch mit zwei Hauptunterschieden:

1. Die `DependencyProperty` wird von einem `private DependencyPropertyKey` .
2. Der CLR-Eigenschaftssetzer ist `protected` oder `private` anstelle von `public` .

Beachten Sie, dass der Setter `MyPropertyPropertyKey` und nicht `MyPropertyProperty` an die `SetValue` Methode `SetValue` . Da die Eigenschaft als schreibgeschützt definiert wurde, muss jeder Versuch, `SetValue` für die Eigenschaft zu verwenden, mit Überladung verwendet werden, die `DependencyPropertyKey` empfängt. Andernfalls wird eine `InvalidOperationException` ausgelöst.

Abhängigkeitseigenschaften online lesen:
https://riptutorial.com/de/wpf/topic/2914/abhangigkeitseigenschaften

Kapitel 4: Benutzerdefinierte UserControls mit Datenbindung erstellen

Bemerkungen

Beachten Sie, dass ein UserControl sich sehr von einem Control unterscheidet. Einer der Hauptunterschiede besteht darin, dass ein UserControl eine XAML-Layoutdatei verwendet, um zu bestimmen, wo mehrere einzelne Steuerelemente platziert werden. Ein Control hingegen ist reiner Code - es gibt überhaupt keine Layoutdatei. Das Erstellen eines benutzerdefinierten Steuerelements kann in gewisser Weise effektiver sein als das Erstellen eines benutzerdefinierten UserControl.

Examples

ComboBox mit benutzerdefiniertem Standardtext

Dieses benutzerdefinierte UserControl wird als reguläre Combobox angezeigt. Im Gegensatz zum integrierten ComboBox-Objekt kann es dem Benutzer jedoch eine Standardzeichenfolge anzeigen, wenn er noch keine Auswahl getroffen hat.

Um dies zu erreichen, besteht unser UserControl aus zwei Controls. Natürlich benötigen wir eine echte ComboBox, aber wir werden auch einen normalen Label verwenden, um den Standardtext anzuzeigen.

CustomComboBox.xaml

```
<UserControl x:Class="UserControlDemo.CustomComboBox"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:cnvrt="clr-namespace:UserControlDemo"
             x:Name="customComboBox">
    <UserControl.Resources>
        <cnvrt:InverseNullVisibilityConverter x:Key="invNullVisibleConverter" />
    </UserControl.Resources>
    <Grid>
        <ComboBox x:Name="comboBox"
                  ItemsSource="{Binding ElementName=customComboBox, Path=MyItemsSource}"
                  SelectedItem="{Binding ElementName=customComboBox, Path=MySelectedItem}"
                  HorizontalContentAlignment="Left" VerticalContentAlignment="Center"/>

        <Label HorizontalAlignment="Left" VerticalAlignment="Center"
               Margin="0,2,20,2" IsHitTestVisible="False"
               Content="{Binding ElementName=customComboBox, Path=DefaultText}"
               Visibility="{Binding ElementName=comboBox, Path=SelectedItem,
Converter={StaticResource invNullVisibleConverter}}"/>
    </Grid>
</UserControl>
```

Wie Sie sehen, besteht dieses einzelne UserControl tatsächlich aus einer Gruppe von zwei einzelnen Controls. Dies ermöglicht uns eine gewisse Flexibilität, die nicht nur in einer ComboBox verfügbar ist.

Hier sind einige wichtige Dinge zu beachten:

- Das UserControl selbst hat einen `x:Name` Satz. Dies liegt daran, dass wir an Eigenschaften binden möchten, die sich im dahinter liegenden Code befinden, was bedeutet, dass es eine Möglichkeit gibt, sich selbst zu referenzieren.
- Jede Bindung der ComboBox hat den Namen des UserControl als `ElementName` . Dies ist so, dass das UserControl weiß, sich selbst zu suchen, um Bindungen zu finden.
- Das Label ist nicht sichtbar. Dies gibt dem Benutzer die Illusion, dass das Label Teil der ComboBox ist. Durch Festlegen von `IsHitTestVisible=false` wir, dass der Benutzer den Mauszeiger darüber bewegt oder auf das Label klickt. Alle Eingaben werden durch die unten angegebene ComboBox geleitet.
- Das Label verwendet einen `InverseNullVisibility` Konverter, um zu bestimmen, ob es sich selbst `InverseNullVisibility` soll oder nicht. Den Code dafür finden Sie am Ende dieses Beispiels.

CustomComboBox.xaml.cs

```
public partial class CustomComboBox : UserControl
{
    public CustomComboBox()
    {
        InitializeComponent();
    }

    public static DependencyProperty DefaultTextProperty =
        DependencyProperty.Register("DefaultText", typeof(string), typeof(CustomComboBox));

    public static DependencyProperty MyItemsSourceProperty =
        DependencyProperty.Register("MyItemsSource", typeof(IEnumerable),
typeof(CustomComboBox));

    public static DependencyProperty SelectedItemProperty =
        DependencyProperty.Register("SelectedItem", typeof(object), typeof(CustomComboBox));

    public string DefaultText
    {
        get { return (string)GetValue(DefaultTextProperty); }
        set { SetValue(DefaultTextProperty, value); }
    }

    public IEnumerable MyItemsSource
    {
        get { return (IEnumerable)GetValue(MyItemsSourceProperty); }
        set { SetValue(MyItemsSourceProperty, value); }
    }

    public object MySelectedItem
    {
        get { return GetValue(MySelectedItemProperty); }
        set { SetValue(MySelectedItemProperty, value); }
    }
```

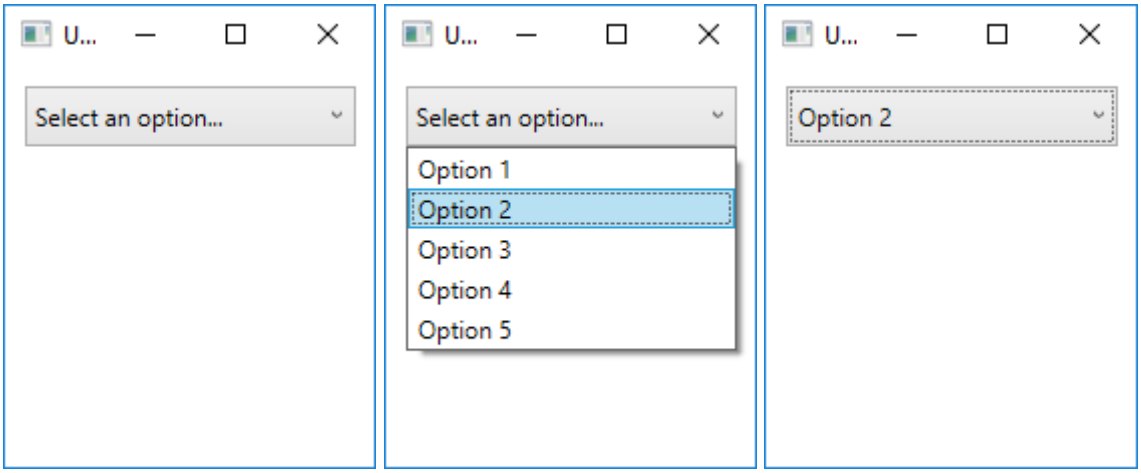
```
}
```

Im Code-behind stellen wir einfach offen, welche Eigenschaften dem Programmierer mit diesem UserControl zur Verfügung stehen sollen. Da wir keinen direkten Zugriff auf die ComboBox außerhalb dieser Klasse haben, müssen wir `MyItemsSource` doppelte Eigenschaften verfügbar `ItemsSource` (z. B. `MyItemsSource` für die ItemsSource der `ItemsSource`). Dies ist jedoch ein geringfügiger Kompromiss, wenn man bedenkt, dass wir dies jetzt ähnlich wie bei einer nativen Steuerung verwenden können.

So kann das `CustomComboBox` UserControl verwendet werden:

```
<Window x:Class="UserControlDemo.UserControlDemo"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:cntrls="clr-namespace:UserControlDemo"
        Title="UserControlDemo" Height="240" Width=200>
    <Grid>
        <cntrls:CustomComboBox HorizontalAlignment="Left" Margin="10,10,0,0"
VerticalAlignment="Top" Width="165"
                               MyItemsSource="{Binding Options}"
                               MySelectedItem="{Binding SelectedOption, Mode=TwoWay}"
                               DefaultText="Select an option..."/>
    <Grid>
</Window>
```

Und das Endergebnis:



Hier ist der InverseNullVisibilityConverter, der für das Label des UserControl benötigt wird. Dies ist nur eine geringfügige Abweichung von der Version von lll :

```
public class InverseNullVisibilityConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo
culture)
    {
        return value == null ? Visibility.Visible : Visibility.Hidden;
    }
```

```
    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo
culture)
    {
        throw new NotImplementedException();
    }
}
```

Benutzerdefinierte UserControls mit Datenbindung erstellen online lesen: https://riptutorial.com/de/wpf/topic/6508/benutzerdefinierte-usercontrols-mit-datenbindung-erstellen

Kapitel 5: Eine Einführung in WPF-Styles

Einführung

Ein Style ermöglicht die vollständige Änderung des visuellen Erscheinungsbildes eines WPF-Steuerelements. Hier einige Beispiele für ein einfaches Styling und eine Einführung in Ressourcenwörterbücher und Animationen.

Examples

Einen Button gestalten

Am einfachsten erstellen Sie einen Stil, indem Sie einen vorhandenen kopieren und bearbeiten.

Erstellen Sie ein einfaches Fenster mit zwei Schaltflächen:

```
<Window x:Class="WPF_Style_Example.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"
    mc:Ignorable="d" ResizeMode="NoResize"
    Title="MainWindow"
    Height="150" Width="200">
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <Button Margin="5" Content="Button 1"/>
    <Button Margin="5" Grid.Row="1" Content="Button 2"/>
</Grid>
```

In Visual Studio können Sie kopieren, indem Sie mit der rechten Maustaste auf die erste Schaltfläche im Editor klicken und im Menü "Vorlage bearbeiten" die Option "Kopie bearbeiten" wählen.

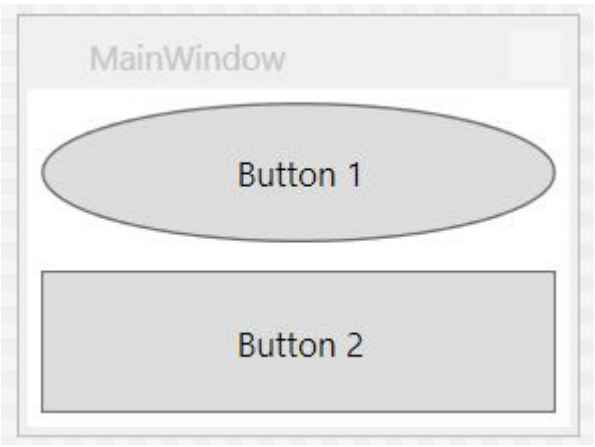
Definieren Sie in "Anwendung".

Das folgende Beispiel zeigt eine modifizierte Vorlage zum Erstellen einer Ellipsenschaltfläche:

```
<Style x:Key="ButtonStyle1" TargetType="{x:Type Button}">
        <Setter Property="FocusVisualStyle" Value="{StaticResource FocusVisual}"/>
        <Setter Property="Background" Value="{StaticResource Button.Static.Background}"/>
        <Setter Property="BorderBrush" Value="{StaticResource Button.Static.Border}"/>
        <Setter Property="Foreground" Value="{DynamicResource {x:Static
SystemColors.ControlTextBrushKey}}"/>
        <Setter Property="BorderThickness" Value="1"/>
        <Setter Property="HorizontalContentAlignment" Value="Center"/>
        <Setter Property="VerticalContentAlignment" Value="Center"/>
        <Setter Property="Padding" Value="1"/>
```

```
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="{x:Type Button}">
                    <Grid>
                        <Ellipse x:Name="ellipse" StrokeThickness="{TemplateBinding
BorderThickness}" Stroke="{TemplateBinding BorderBrush}" Fill="{TemplateBinding Background}"
SnapsToDevicePixels="true"/>
                        <ContentPresenter x:Name="contentPresenter" Focusable="False"
HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}" Margin="{TemplateBinding
Padding}" RecognizesAccessKey="True" SnapsToDevicePixels="{TemplateBinding
SnapsToDevicePixels}" VerticalAlignment="{TemplateBinding VerticalContentAlignment}"/>
                    </Grid>
                    <ControlTemplate.Triggers>
                        <Trigger Property="IsDefaulted" Value="true">
                            <Setter Property="Stroke" TargetName="ellipse"
Value="{DynamicResource {x:Static SystemColors.HighlightBrushKey}}"/>
                        </Trigger>
                        <Trigger Property="IsMouseOver" Value="true">
                            <Setter Property="Fill" TargetName="ellipse"
Value="{StaticResource Button.MouseOver.Background}"/>
                            <Setter Property="Stroke" TargetName="ellipse"
Value="{StaticResource Button.MouseOver.Border}"/>
                        </Trigger>
                        <Trigger Property="IsPressed" Value="true">
                            <Setter Property="Fill" TargetName="ellipse"
Value="{StaticResource Button.Pressed.Background}"/>
                            <Setter Property="Stroke" TargetName="ellipse"
Value="{StaticResource Button.Pressed.Border}"/>
                        </Trigger>
                        <Trigger Property="IsEnabled" Value="false">
                            <Setter Property="Fill" TargetName="ellipse"
Value="{StaticResource Button.Disabled.Background}"/>
                            <Setter Property="Stroke" TargetName="ellipse"
Value="{StaticResource Button.Disabled.Border}"/>
                            <Setter Property="TextElement.Foreground"
TargetName="contentPresenter" Value="{StaticResource Button.Disabled.Foreground}"/>
                        </Trigger>
                    </ControlTemplate.Triggers>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
```

Das Ergebnis:



Auf alle Schaltflächen angewendeter Stil

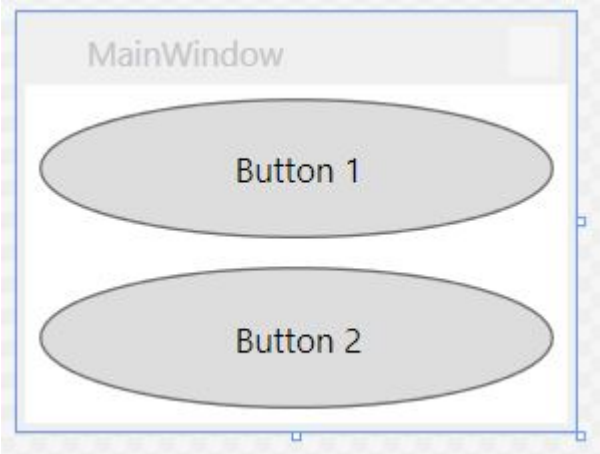
Im vorherigen Beispiel wird beim Entfernen des Elements x: Key des Stils der Stil auf alle Schaltflächen im Anwendungsbereich angewendet.

```
<Style TargetType="{x:Type Button}">
        <Setter Property="FocusVisualStyle" Value="{StaticResource FocusVisual}"/>
        <Setter Property="Background" Value="{StaticResource Button.Static.Background}"/>
        <Setter Property="BorderBrush" Value="{StaticResource Button.Static.Border}"/>
        <Setter Property="Foreground" Value="{DynamicResource {x:Static
SystemColors.ControlTextBrushKey}}"/>
        <Setter Property="BorderThickness" Value="1"/>
        <Setter Property="HorizontalContentAlignment" Value="Center"/>
        <Setter Property="VerticalContentAlignment" Value="Center"/>
        <Setter Property="Padding" Value="1"/>
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="{x:Type Button}">
                    <Grid>
                        <Ellipse x:Name="ellipse" StrokeThickness="{TemplateBinding
BorderThickness}" Stroke="{TemplateBinding BorderBrush}" Fill="{TemplateBinding Background}"
SnapsToDevicePixels="true"/>
                        <ContentPresenter x:Name="contentPresenter" Focusable="False"
HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}" Margin="{TemplateBinding
Padding}" RecognizesAccessKey="True" SnapsToDevicePixels="{TemplateBinding
SnapsToDevicePixels}" VerticalAlignment="{TemplateBinding VerticalContentAlignment}"/>
                    </Grid>
                    <ControlTemplate.Triggers>
                        <Trigger Property="IsDefaulted" Value="true">
                            <Setter Property="Stroke" TargetName="ellipse"
Value="{DynamicResource {x:Static SystemColors.HighlightBrushKey}}"/>
                        </Trigger>
                        <Trigger Property="IsMouseOver" Value="true">
                            <Setter Property="Fill" TargetName="ellipse"
Value="{StaticResource Button.MouseOver.Background}"/>
                            <Setter Property="Stroke" TargetName="ellipse"
Value="{StaticResource Button.MouseOver.Border}"/>
                        </Trigger>
                        <Trigger Property="IsPressed" Value="true">
                            <Setter Property="Fill" TargetName="ellipse"
Value="{StaticResource Button.Pressed.Background}"/>
                            <Setter Property="Stroke" TargetName="ellipse"
Value="{StaticResource Button.Pressed.Border}"/>
                        </Trigger>
                        <Trigger Property="IsEnabled" Value="false">
                            <Setter Property="Fill" TargetName="ellipse"
Value="{StaticResource Button.Disabled.Background}"/>
                            <Setter Property="Stroke" TargetName="ellipse"
Value="{StaticResource Button.Disabled.Border}"/>
                            <Setter Property="TextElement.Foreground"
TargetName="contentPresenter" Value="{StaticResource Button.Disabled.Foreground}"/>
                        </Trigger>
                    </ControlTemplate.Triggers>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
```

Beachten Sie, dass der Stil für einzelne Schaltflächen nicht mehr angegeben werden muss:

```
<Window x:Class="WPF_Style_Example.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"
    mc:Ignorable="d" ResizeMode="NoResize"
    Title="MainWindow"
    Height="150" Width="200">
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <Button Margin="5" Content="Button 1"/>
    <Button Margin="5" Grid.Row="1" Content="Button 2"/>
</Grid>
```

Beide Knöpfe sind jetzt gestylt.



Eine ComboBox gestalten

Beginnen `ComboBox` mit folgenden `ComboBox` es:

```
<Window x:Class="WPF_Style_Example.ComboBoxWindow"
    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"
    mc:Ignorable="d" ResizeMode="NoResize"
    Title="ComboBoxWindow"
    Height="100" Width="150">
<StackPanel>
    <ComboBox Margin="5" SelectedIndex="0">
        <ComboBoxItem Content="Item A"/>
        <ComboBoxItem Content="Item B"/>
        <ComboBoxItem Content="Item C"/>
    </ComboBox>
    <ComboBox IsEditable="True" Margin="5" SelectedIndex="0">
        <ComboBoxItem Content="Item 1"/>
        <ComboBoxItem Content="Item 2"/>
        <ComboBoxItem Content="Item 3"/>
    </ComboBox>
</StackPanel>
```

Klicken Sie mit der `ComboBox` auf die erste `ComboBox` im Designer und wählen Sie "Vorlage bearbeiten -> Kopie bearbeiten". Definieren Sie den Stil im Anwendungsbereich.

Es wurden 3 Stile erstellt:

```
ComboBoxToggleButton
ComboBoxEditableTextBox
ComboBoxStyle1
```

Und 2 Vorlagen:

```
ComboBoxTemplate
ComboBoxEditableTemplate
```

Ein Beispiel zum Bearbeiten des `ComboBoxToggleButton` Stils:

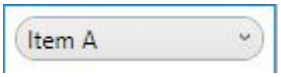
```
<SolidColorBrush x:Key="ComboBox.Static.Border" Color="#FFACACAC"/>
    <SolidColorBrush x:Key="ComboBox.Static.Editable.Background" Color="#FFFFFFFF"/>
    <SolidColorBrush x:Key="ComboBox.Static.Editable.Border" Color="#FFABADB3"/>
    <SolidColorBrush x:Key="ComboBox.Static.Editable.Button.Background" Color="Transparent"/>
    <SolidColorBrush x:Key="ComboBox.Static.Editable.Button.Border" Color="Transparent"/>
    <SolidColorBrush x:Key="ComboBox.MouseOver.Glyph" Color="#FF000000"/>
    <LinearGradientBrush x:Key="ComboBox.MouseOver.Background" EndPoint="0,1"
StartPoint="0,0">
        <GradientStop Color="Orange" Offset="0.0"/>
        <GradientStop Color="OrangeRed" Offset="1.0"/>
    </LinearGradientBrush>
    <SolidColorBrush x:Key="ComboBox.MouseOver.Border" Color="Red"/>
    <SolidColorBrush x:Key="ComboBox.MouseOver.Editable.Background" Color="#FFFFFFFF"/>
    <SolidColorBrush x:Key="ComboBox.MouseOver.Editable.Border" Color="#FF7EB4EA"/>
    <LinearGradientBrush x:Key="ComboBox.MouseOver.Editable.Button.Background" EndPoint="0,1"
StartPoint="0,0">
        <GradientStop Color="#FFEBF4FC" Offset="0.0"/>
        <GradientStop Color="#FFDCECFC" Offset="1.0"/>
    </LinearGradientBrush>
    <SolidColorBrush x:Key="ComboBox.MouseOver.Editable.Button.Border" Color="#FF7EB4EA"/>
    <SolidColorBrush x:Key="ComboBox.Pressed.Glyph" Color="#FF000000"/>
    <LinearGradientBrush x:Key="ComboBox.Pressed.Background" EndPoint="0,1" StartPoint="0,0">
        <GradientStop Color="OrangeRed" Offset="0.0"/>
        <GradientStop Color="Red" Offset="1.0"/>
    </LinearGradientBrush>
    <SolidColorBrush x:Key="ComboBox.Pressed.Border" Color="DarkRed"/>
    <SolidColorBrush x:Key="ComboBox.Pressed.Editable.Background" Color="#FFFFFFFF"/>
    <SolidColorBrush x:Key="ComboBox.Pressed.Editable.Border" Color="#FF569DE5"/>
    <LinearGradientBrush x:Key="ComboBox.Pressed.Editable.Button.Background" EndPoint="0,1"
StartPoint="0,0">
        <GradientStop Color="#FFDAEBFC" Offset="0.0"/>
        <GradientStop Color="#FFC4E0FC" Offset="1.0"/>
    </LinearGradientBrush>
    <SolidColorBrush x:Key="ComboBox.Pressed.Editable.Button.Border" Color="#FF569DE5"/>
    <SolidColorBrush x:Key="ComboBox.Disabled.Glyph" Color="#FFBFBFBF"/>
    <SolidColorBrush x:Key="ComboBox.Disabled.Background" Color="#FFF0F0F0"/>
    <SolidColorBrush x:Key="ComboBox.Disabled.Border" Color="#FFD9D9D9"/>
    <SolidColorBrush x:Key="ComboBox.Disabled.Editable.Background" Color="#FFFFFFFF"/>
    <SolidColorBrush x:Key="ComboBox.Disabled.Editable.Border" Color="#FFBFBFBF"/>
    <SolidColorBrush x:Key="ComboBox.Disabled.Editable.Button.Background"
Color="Transparent"/>
```

```
    <SolidColorBrush x:Key="ComboBox.Disabled.Editable.Button.Border" Color="Transparent"/>
    <SolidColorBrush x:Key="ComboBox.Static.Glyph" Color="#FF606060"/>
    <Style x:Key="ComboBoxToggleButton" TargetType="{x:Type ToggleButton}">
        <Setter Property="OverridesDefaultStyle" Value="true"/>
        <Setter Property="IsTabStop" Value="false"/>
        <Setter Property="Focusable" Value="false"/>
        <Setter Property="ClickMode" Value="Press"/>
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="{x:Type ToggleButton}">
                    <Border x:Name="templateRoot" CornerRadius="10"
BorderBrush="{StaticResource ComboBox.Static.Border}" BorderThickness="{TemplateBinding
BorderThickness}" Background="{StaticResource ComboBox.Static.Background}"
SnapsToDevicePixels="true">
                        <Border x:Name="splitBorder" BorderBrush="Transparent"
BorderThickness="1" HorizontalAlignment="Right" Margin="0" SnapsToDevicePixels="true"
Width="{DynamicResource {x:Static SystemParameters.VerticalScrollBarWidthKey}}">
                            <Path x:Name="arrow" Data="F1 M 0,0 L 2.667,2.66665 L 5.3334,0 L
5.3334,-1.78168 L 2.6667,0.88501 L0,-1.78168 L0,0 Z" Fill="{StaticResource
ComboBox.Static.Glyph}" HorizontalAlignment="Center" Margin="0" VerticalAlignment="Center"/>
                        </Border>
                    </Border>
                    <ControlTemplate.Triggers>
                        <MultiDataTrigger>
                            <MultiDataTrigger.Conditions>
                                <Condition Binding="{Binding IsEditable,
RelativeSource={RelativeSource AncestorType={x:Type ComboBox}}}" Value="true"/>
                                <Condition Binding="{Binding IsMouseOver,
RelativeSource={RelativeSource Self}}" Value="false"/>
                                <Condition Binding="{Binding IsPressed,
RelativeSource={RelativeSource Self}}" Value="false"/>
                                <Condition Binding="{Binding IsEnabled,
RelativeSource={RelativeSource Self}}" Value="true"/>
                            </MultiDataTrigger.Conditions>
                            <Setter Property="Background" TargetName="templateRoot"
Value="{StaticResource ComboBox.Static.Editable.Background}"/>
                            <Setter Property="BorderBrush" TargetName="templateRoot"
Value="{StaticResource ComboBox.Static.Editable.Border}"/>
                            <Setter Property="Background" TargetName="splitBorder"
Value="{StaticResource ComboBox.Static.Editable.Button.Background}"/>
                            <Setter Property="BorderBrush" TargetName="splitBorder"
Value="{StaticResource ComboBox.Static.Editable.Button.Border}"/>
                        </MultiDataTrigger>
                        <Trigger Property="IsMouseOver" Value="true">
                            <Setter Property="BorderThickness" TargetName="templateRoot"
Value="2"/>
                        </Trigger>
                        <MultiDataTrigger>
                            <MultiDataTrigger.Conditions>
                                <Condition Binding="{Binding IsMouseOver,
RelativeSource={RelativeSource Self}}" Value="true"/>
                                <Condition Binding="{Binding IsEditable,
RelativeSource={RelativeSource AncestorType={x:Type ComboBox}}}" Value="false"/>
                            </MultiDataTrigger.Conditions>
                            <Setter Property="Background" TargetName="templateRoot"
Value="{StaticResource ComboBox.MouseOver.Background}"/>
                            <Setter Property="BorderBrush" TargetName="templateRoot"
Value="{StaticResource ComboBox.MouseOver.Border}"/>
                        </MultiDataTrigger>
                        <MultiDataTrigger>
                            <MultiDataTrigger.Conditions>
```

```
                                <Condition Binding="{Binding IsMouseOver,
RelativeSource={RelativeSource Self}}" Value="true"/>
                                <Condition Binding="{Binding IsEditable,
RelativeSource={RelativeSource AncestorType={x:Type ComboBox}}}" Value="true"/>
                            </MultiDataTrigger.Conditions>
                            <Setter Property="Background" TargetName="templateRoot"
Value="{StaticResource ComboBox.MouseOver.Editable.Background}"/>
                            <Setter Property="BorderBrush" TargetName="templateRoot"
Value="{StaticResource ComboBox.MouseOver.Editable.Border}"/>
                            <Setter Property="Background" TargetName="splitBorder"
Value="{StaticResource ComboBox.MouseOver.Editable.Button.Background}"/>
                            <Setter Property="BorderBrush" TargetName="splitBorder"
Value="{StaticResource ComboBox.MouseOver.Editable.Button.Border}"/>
                        </MultiDataTrigger>
                        <Trigger Property="IsPressed" Value="true">
                            <Setter Property="Fill" TargetName="arrow" Value="{StaticResource
ComboBox.Pressed.Glyph}"/>
                        </Trigger>
                        <MultiDataTrigger>
                            <MultiDataTrigger.Conditions>
                                <Condition Binding="{Binding IsPressed,
RelativeSource={RelativeSource Self}}" Value="true"/>
                                <Condition Binding="{Binding IsEditable,
RelativeSource={RelativeSource AncestorType={x:Type ComboBox}}}" Value="false"/>
                            </MultiDataTrigger.Conditions>
                            <Setter Property="Background" TargetName="templateRoot"
Value="{StaticResource ComboBox.Pressed.Background}"/>
                            <Setter Property="BorderBrush" TargetName="templateRoot"
Value="{StaticResource ComboBox.Pressed.Border}"/>
                        </MultiDataTrigger>
                        <MultiDataTrigger>
                            <MultiDataTrigger.Conditions>
                                <Condition Binding="{Binding IsPressed,
RelativeSource={RelativeSource Self}}" Value="true"/>
                                <Condition Binding="{Binding IsEditable,
RelativeSource={RelativeSource AncestorType={x:Type ComboBox}}}" Value="true"/>
                            </MultiDataTrigger.Conditions>
                            <Setter Property="Background" TargetName="templateRoot"
Value="{StaticResource ComboBox.Pressed.Editable.Background}"/>
                            <Setter Property="BorderBrush" TargetName="templateRoot"
Value="{StaticResource ComboBox.Pressed.Editable.Border}"/>
                            <Setter Property="Background" TargetName="splitBorder"
Value="{StaticResource ComboBox.Pressed.Editable.Button.Background}"/>
                            <Setter Property="BorderBrush" TargetName="splitBorder"
Value="{StaticResource ComboBox.Pressed.Editable.Button.Border}"/>
                        </MultiDataTrigger>
                        <Trigger Property="IsEnabled" Value="false">
                            <Setter Property="Fill" TargetName="arrow" Value="{StaticResource
ComboBox.Disabled.Glyph}"/>
                        </Trigger>
                        <MultiDataTrigger>
                            <MultiDataTrigger.Conditions>
                                <Condition Binding="{Binding IsEnabled,
RelativeSource={RelativeSource Self}}" Value="false"/>
                                <Condition Binding="{Binding IsEditable,
RelativeSource={RelativeSource AncestorType={x:Type ComboBox}}}" Value="false"/>
                            </MultiDataTrigger.Conditions>
                            <Setter Property="Background" TargetName="templateRoot"
Value="{StaticResource ComboBox.Disabled.Background}"/>
                            <Setter Property="BorderBrush" TargetName="templateRoot"
Value="{StaticResource ComboBox.Disabled.Border}"/>
```

```
                        </MultiDataTrigger>
                        <MultiDataTrigger>
                            <MultiDataTrigger.Conditions>
                                <Condition Binding="{Binding IsEnabled,
RelativeSource={RelativeSource Self}}" Value="false"/>
                                <Condition Binding="{Binding IsEditable,
RelativeSource={RelativeSource AncestorType={x:Type ComboBox}}}" Value="true"/>
                            </MultiDataTrigger.Conditions>
                            <Setter Property="Background" TargetName="templateRoot"
Value="{StaticResource ComboBox.Disabled.Editable.Background}"/>
                            <Setter Property="BorderBrush" TargetName="templateRoot"
Value="{StaticResource ComboBox.Disabled.Editable.Border}"/>
                            <Setter Property="Background" TargetName="splitBorder"
Value="{StaticResource ComboBox.Disabled.Editable.Button.Background}"/>
                            <Setter Property="BorderBrush" TargetName="splitBorder"
Value="{StaticResource ComboBox.Disabled.Editable.Button.Border}"/>
                        </MultiDataTrigger>
                    </ControlTemplate.Triggers>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
```

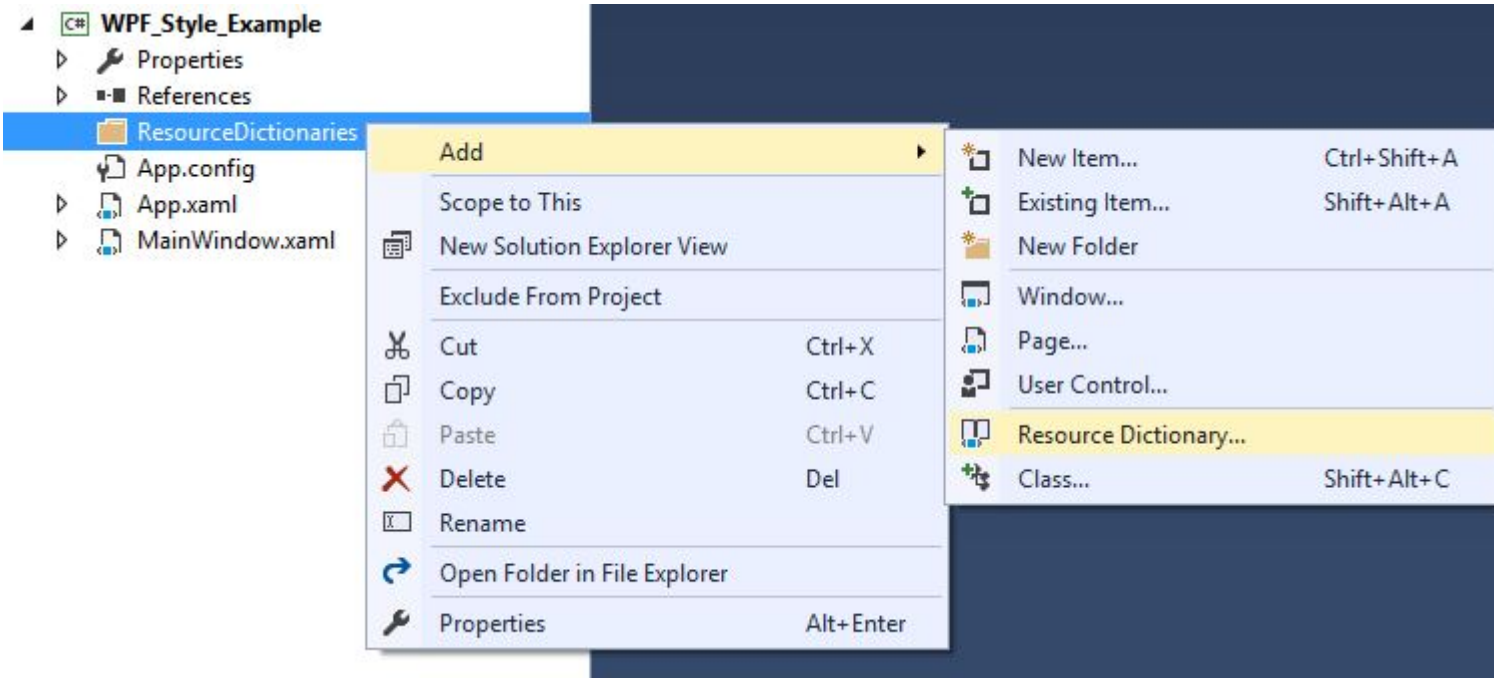
Dadurch wird eine abgerundete `ComboBox` , die bei Mauszeiger orange hervorgehoben wird und beim Drücken rot wird.



Beachten Sie, dass dadurch die Editable-Combobox darunter nicht geändert wird. Das Ändern erfordert, dass der `ComboBoxEditableTextBox` Stil oder die `ComboBoxEditableTemplate` .

Ressourcenwörterbuch erstellen

Viele Stile in App.xaml werden schnell komplex, sodass sie in separate Ressourcenwörterbücher eingefügt werden können.



Um das Wörterbuch verwenden zu können, muss es mit App.xaml zusammengeführt werden. Nach dem Erstellen des Ressourcenwörterbuchs in App.xaml:

```
<Application
         xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
         xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
         x:Class="WPF_Style_Example.App"
         StartupUri="MainWindow.xaml">
    <Application.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="ResourceDictionaries/Dictionary1.xaml"/>
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

Neue Stile können jetzt in der Dictionary1.xaml erstellt und auf sie verwiesen werden, als wären sie in App.xaml. Nach dem Erstellen des Projekts wird die Option auch in Visual Studio angezeigt, wenn Sie einen Stil kopieren, um ihn im neuen Ressourcenwörterbuch zu finden.

Schaltflächenstil DoubleAnimation

Das folgende `Window` wurde erstellt:

```
<Window x:Class="WPF_Style_Example.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"
    mc:Ignorable="d" ResizeMode="NoResize"
    Title="MainWindow"
    Height="150" Width="250">
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <Button Margin="5" Content="Button 1" Width="200"/>
    <Button Margin="5" Grid.Row="1" Content="Button 2" Width="200"/>
</Grid>
```

Ein Stil (erstellt in App.xaml) wurde auf die Schaltflächen angewendet, der die Breite von 200 bis 100 animiert, wenn die Maus in das Steuerelement eintritt, und von 100 bis 200, wenn sie verlassen wird:

```
<Style TargetType="{x:Type Button}">
    <Setter Property="FocusVisualStyle" Value="{StaticResource FocusVisual}"/>
    <Setter Property="Background" Value="{StaticResource Button.Static.Background}"/>
    <Setter Property="BorderBrush" Value="{StaticResource Button.Static.Border}"/>
    <Setter Property="Foreground" Value="{DynamicResource {x:Static
SystemColors.ControlTextBrushKey}}"/>
    <Setter Property="BorderThickness" Value="1"/>
    <Setter Property="HorizontalContentAlignment" Value="Center"/>
    <Setter Property="VerticalContentAlignment" Value="Center"/>
```

```
    <Setter Property="Padding" Value="1"/>
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type Button}">
                <Grid Background="White">
                    <Border x:Name="border" BorderBrush="{TemplateBinding BorderBrush}"
BorderThickness="{TemplateBinding BorderThickness}" Background="{TemplateBinding Background}"
SnapsToDevicePixels="true">
                        <ContentPresenter x:Name="contentPresenter" Focusable="False"
HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}" Margin="{TemplateBinding
Padding}" RecognizesAccessKey="True" SnapsToDevicePixels="{TemplateBinding
SnapsToDevicePixels}" VerticalAlignment="{TemplateBinding VerticalContentAlignment}"/>
                    </Border>
                </Grid>
                <ControlTemplate.Triggers>
                    <EventTrigger RoutedEvent="MouseEnter">
                        <BeginStoryboard>
                            <Storyboard>
                                <DoubleAnimation To="100" From="200"
Storyboard.TargetProperty="Width" Storyboard.TargetName="border" Duration="0:0:0.25"/>
                            </Storyboard>
                        </BeginStoryboard>
                    </EventTrigger>
                    <EventTrigger RoutedEvent="MouseLeave">
                        <BeginStoryboard>
                            <Storyboard>
                                <DoubleAnimation To="200" From="100"
Storyboard.TargetProperty="Width" Storyboard.TargetName="border" Duration="0:0:0.25"/>
                            </Storyboard>
                        </BeginStoryboard>
                    </EventTrigger>
                    <Trigger Property="IsDefaulted" Value="true">
                        <Setter Property="BorderBrush" TargetName="border"
Value="{DynamicResource {x:Static SystemColors.HighlightBrushKey}}"/>
                    </Trigger>
                    <Trigger Property="IsMouseOver" Value="true">
                        <Setter Property="Background" TargetName="border"
Value="{StaticResource Button.MouseOver.Background}"/>
                        <Setter Property="BorderBrush" TargetName="border"
Value="{StaticResource Button.MouseOver.Border}"/>
                    </Trigger>
                    <Trigger Property="IsPressed" Value="true">
                        <Setter Property="Background" TargetName="border"
Value="{StaticResource Button.Pressed.Background}"/>
                        <Setter Property="BorderBrush" TargetName="border"
Value="{StaticResource Button.Pressed.Border}"/>
                    </Trigger>
                    <Trigger Property="IsEnabled" Value="false">
                        <Setter Property="Background" TargetName="border"
Value="{StaticResource Button.Disabled.Background}"/>
                        <Setter Property="BorderBrush" TargetName="border"
Value="{StaticResource Button.Disabled.Border}"/>
                        <Setter Property="TextElement.Foreground"
TargetName="contentPresenter" Value="{StaticResource Button.Disabled.Foreground}"/>
                    </Trigger>
                </ControlTemplate.Triggers>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

Eine Einführung in WPF-Styles online lesen: https://riptutorial.com/de/wpf/topic/9670/eine-einfuhrung-in-wpf-styles

Kapitel 6: Einführung in die WPF-Datenbindung

Syntax

- {Binding PropertyName} *entspricht* {Bindungspfad = PropertyName}
- {Bindungspfad = SomeProperty.SomeOtherProperty.YetAnotherProperty}
- {Bindungspfad = SomeListProperty [1]}

Parameter

Parameter	Einzelheiten
Pfad	Gibt den Pfad an, an den gebunden werden soll. Wenn nicht angegeben, bindet an den DataContext selbst.
UpdateSourceTrigger	Gibt an, wann der Wert der Bindungsquelle aktualisiert wurde. `LostFocus` . Der am häufigsten verwendete Wert ist `PropertyChanged` .
Modus	Normalerweise `OneWay` oder `TwoWay` . Wenn die Bindung dies nicht `TwoWay` , wird standardmäßig `OneWay` sofern das Bindungsziel nicht `TwoWay` . Ein Fehler tritt auf, wenn `TwoWay` verwendet wird , um eine Nur - Lese - Eigenschaft zu binden, zB `OneWay` explizit festgelegt werden muß , wenn eine Nur - Lese - String - Eigenschaft zur Bindung `TextBox.Text` .
Quelle	Ermöglicht die Verwendung einer `StaticResource` als Bindungsquelle anstelle des aktuellen DataContext.
RelativeSource	Ermöglicht die Verwendung eines anderen XAML-Elements als Bindungsquelle anstelle des aktuellen DataContext.
ElementName	Ermöglicht die Verwendung eines benannten XAML-Elements als Bindungsquelle anstelle des aktuellen DataContext.
FallbackValue	Wenn die Bindung fehlschlägt, wird dieser Wert dem Bindungsziel bereitgestellt.
TargetNullValue	Wenn die Bindungsquelle Wert ist `null` , wird dieser Wert an das Bindungsziel bereitgestellt.
Konverter	Gibt die Konverter- `StaticResource` , die zum Konvertieren des Bindungswerts verwendet wird, z. B. Konvertieren eines Boolean in ein `Visibility` .

Parameter	Einzelheiten
ConverterParameter	Gibt einen optionalen Parameter an, der dem Konverter zur Verfügung gestellt werden soll. Dieser Wert muss statisch sein und kann nicht gebunden werden.
StringFormat	Gibt eine Formatzeichenfolge an, die beim Anzeigen des gebundenen Werts verwendet werden soll.
Verzögern	(WPF 4.5+) Gibt eine Verzögerung in `milliseconds` für die Bindung zum Aktualisieren der `BindingSource` im `ViewModel` . Dies muss mit `Mode=TwoWay` und `UpdateSourceTrigger=PropertyChanged` , um wirksam zu werden.

Bemerkungen

UpdateSourceTrigger

Standardmäßig aktualisiert WPF die Bindungsquelle, wenn das Steuerelement den Fokus verliert. Wenn es jedoch nur ein Steuerelement gibt, das den Fokus erhält - was in Beispielen üblich ist - müssen Sie `UpdateSourceTrigger=PropertyChanged` angeben, `UpdateSourceTrigger=PropertyChanged` die Aktualisierungen funktionieren.

Sie sollten `PropertyChanged` als Auslöser für viele bidirektionale Bindungen verwenden, es sei denn, das Aktualisieren der Bindungsquelle bei jedem Tastendruck ist kostspielig oder die Live-Datenvalidierung ist unerwünscht.

Die Verwendung von `LostFocus` hat einen unglücklichen Nebeneffekt: Wenn Sie die `LostFocus` drücken, um ein Formular mit einer mit `IsDefault` markierten Schaltfläche zu `IsDefault` wird die Eigenschaft, die Ihre Bindung unterstützt, nicht aktualisiert. Glücklicherweise gibt es einige Problemumgehungen .

Bitte beachten Sie auch, dass WPF (4.5+) im Gegensatz zu UWP auch die `Delay` Eigenschaft in Bindings hat, die für einige Bindings mit lokalen Einstellungen oder einfachen geringfügigen Intelligenzeinstellungen wie einigen `TextBox` Validierungen gerade ausreicht.

Examples

Konvertieren Sie einen Booleschen Wert in einen Sichtbarkeitswert

In diesem Beispiel wird das rote Feld (Rahmen) `IValueConverter` wenn das Kontrollkästchen nicht mithilfe eines `IValueConverter` .

Hinweis: Der im folgenden Beispiel verwendete `BooleanToVisibilityConverter` ist ein integrierter Wertkonverter, der sich im Namespace System.Windows.Controls befindet.

XAML:

```
<Window x:Class="StackOverflowDataBindingExample.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Window.Resources>
        <BooleanToVisibilityConverter x:Key="VisibleIfTrueConverter" />
    </Window.Resources>
    <StackPanel>
        <CheckBox x:Name="MyCheckBox"
                  IsChecked="True" />
        <Border Background="Red" Width="20" Height="20"
                Visibility="{Binding Path=IsChecked,ElementName=MyCheckBox,
Converter={StaticResource VisibleIfTrueConverter}}" />
    </StackPanel>
</Window>
```

DataContext definieren

Um mit Bindungen in WPF arbeiten zu können, müssen Sie einen **DataContext** definieren. Der DataContext teilt Bindungen mit, woher ihre Daten standardmäßig abgerufen werden.

```
<Window x:Class="StackOverflowDataBindingExample.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:StackOverflowDataBindingExample"
        xmlns:vm="clr-namespace:StackOverflowDataBindingExample.ViewModels"
        mc:Ignorable="d"
        Title="MainWindow" Height="350" Width="525">
    <Window.DataContext>
        <vm:HelloWorldViewModel />
    </Window.DataContext>
    ...
</Window>
```

Sie können DataContext auch über Code-Behind einstellen. Beachten Sie jedoch, dass XAML IntelliSense etwas wählerisch ist: In XAML muss ein stark typisierter DataContext festgelegt werden, damit IntelliSense Eigenschaften für die Bindung vorschlagen kann.

```
/// <summary>
/// Interaction logic for MainWindow.xaml
/// </summary>
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        DataContext = new HelloWorldViewModel();
    }
}
```

Zwar gibt es Frameworks, mit denen Sie DataContext flexibler definieren können (z. B. MVVM Light hat einen Viewmodel-Locator, der die Umkehrung der Steuerung verwendet). In diesem Lernprogramm verwenden wir jedoch die schnelle und unreine Methode.

Sie können einen DataContext für nahezu jedes visuelle Element in WPF definieren. Der DataContext wird in der Regel von Vorfahren im visuellen Baum geerbt, sofern er nicht explizit überschrieben wurde, z. B. in einem ContentPresenter.

INotifyPropertyChanged implementieren

`INotifyPropertyChanged` ist eine Schnittstelle, die von Bindungsquellen (dh dem DataContext) verwendet wird, um die Benutzeroberfläche oder andere Komponenten `INotifyPropertyChanged` , dass eine Eigenschaft geändert wurde. WPF aktualisiert die Benutzeroberfläche automatisch für Sie, wenn das `PropertyChanged` Ereignis ausgelöst wird. Es ist wünschenswert, dass diese Schnittstelle in einer Basisklasse implementiert ist, von der alle Viewmodels erben können.

In C # 6 ist dies alles, was Sie brauchen:

```
public abstract class ViewModelBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    protected void NotifyPropertyChanged([CallerMemberName] string name = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
    }
}
```

Auf diese Weise können Sie `NotifyPropertyChanged` auf zwei verschiedene Arten aufrufen:

1. `NotifyPropertyChanged()` , wodurch das Ereignis für den Setter `NotifyPropertyChanged()` wird, der es aufgrund des Attributs CallerMemberName aufruft .
2. `NotifyPropertyChanged(nameof(SomeOtherProperty))` , wodurch das Ereignis für SomeOtherProperty ausgelöst wird.

Für .NET 4.5 und höher mit C # 5.0 kann dies stattdessen verwendet werden:

```
public abstract class ViewModelBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    protected void NotifyPropertyChanged([CallerMemberName] string name = null)
    {
        var handler = PropertyChanged;
        if (handler != null)
        {
            handler(this, new PropertyChangedEventArgs(name));
        }
    }
}
```

In .NET-Versionen vor 4.5 müssen Sie Eigenschaftsnamen als Zeichenfolgekonstanten oder eine Lösung mit Ausdrücken festlegen .

Anmerkung: Es ist möglich, an eine Eigenschaft eines "plain old C # object" (POCO) zu binden,

das `INotifyPropertyChanged` nicht implementiert, und zu beobachten, dass die Bindungen besser funktionieren als erwartet. Dies ist eine versteckte Funktion in .NET und sollte wahrscheinlich vermieden werden. `OneTime` insbesondere für Speicherlecks, wenn der `Mode` der Bindung nicht `OneTime` (siehe hier).

Warum wird die Bindung ohne Implementierung von INotifyPropertyChanged aktualisiert?

Binden an die Eigenschaft eines anderen benannten Elements

Sie können an eine Eigenschaft eines benannten Elements binden, das benannte Element muss jedoch im Gültigkeitsbereich liegen.

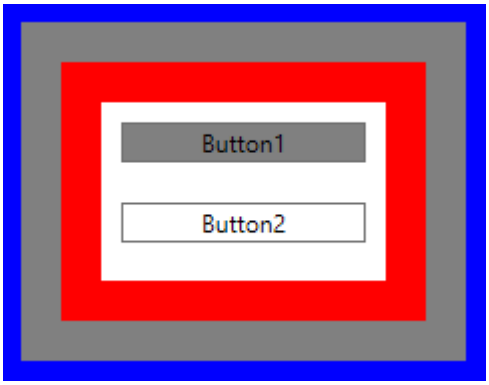
```
<StackPanel>
    <CheckBox x:Name="MyCheckBox" IsChecked="True" />
    <TextBlock Text="{Binding IsChecked, ElementName=MyCheckBox}" />
</StackPanel>
```

Binden an Eigentum eines Vorfahren

Sie können an eine Eigenschaft eines Vorfahren in der visuellen Baumstruktur binden, indem Sie eine `RelativeSource` Bindung verwenden. Das nächstgelegene Steuerelement oberhalb des visuellen Baums, das denselben Typ hat oder von dem von Ihnen angegebenen Typ abgeleitet ist, wird als Quelle der Bindung verwendet:

```
<Grid Background="Blue">
    <Grid Background="Gray" Margin="10">
        <Border Background="Red" Margin="20">
            <StackPanel Background="White" Margin="20">
                <Button Margin="10" Content="Button1" Background="{Binding Background,
RelativeSource={RelativeSource Mode=FindAncestor, AncestorType={x:Type Grid}}}" />
                <Button Margin="10" Content="Button2" Background="{Binding Background,
RelativeSource={RelativeSource Mode=FindAncestor, AncestorType={x:Type FrameworkElement}}}" />
            </StackPanel>
        </Border>
    </Grid>
</Grid>
```

In diesem Beispiel hat *Button1* einen grauen Hintergrund, da der nächste Vorläufer des `Grid` einen grauen Hintergrund hat. *Button2* hat einen weißen Hintergrund, da der nächstgelegene Vorfahr von `FrameworkElement` das weiße `StackPanel` .



Mehrere Werte mit einer MultiBinding binden

Das MultiBinding ermöglicht das Binden mehrerer Werte an dieselbe Eigenschaft. Im folgenden Beispiel werden mehrere Werte an die Text-Eigenschaft eines Textfelds gebunden und mit der StringFormat-Eigenschaft formatiert.

```
<TextBlock>
    <TextBlock.Text>
        <MultiBinding StringFormat="{}{0} {1}">
            <Binding Path="User.Forename"/>
            <Binding Path="User.Surname"/>
        </MultiBinding>
    </TextBlock.Text>
</TextBlock>
```

Neben `StringFormat` kann ein `IMultiValueConverter` auch verwendet werden, um die Werte der Bindungen in einen Wert für das MultiBinding-Ziel zu konvertieren.

MultiBindings können jedoch nicht geschachtelt werden.

Einführung in die WPF-Datenbindung online lesen:
https://riptutorial.com/de/wpf/topic/2236/einfuhrung-in-die-wpf-datenbindung

Kapitel 7: Löst aus

Einführung

Erörterung der verschiedenen in WPF verfügbaren `DataTrigger` , einschließlich `Trigger` , `DataTrigger` , `MultiTrigger` , `MultiDataTrigger` und `EventTrigger` .

Auslöser ermöglichen jeder Klasse, die von `FrameworkElement` oder `FrameworkContentElement` , ihre Eigenschaften basierend auf bestimmten im Auslöser definierten Bedingungen festzulegen oder zu ändern. Wenn ein Element formatiert werden kann, kann es auch ausgelöst werden.

Bemerkungen

- Alle Trigger außer `EventTrigger` müssen in einem `<Style>` -Element definiert werden. Ein `EventTrigger` kann entweder in einem `<Style>` -Element oder in der `Triggers` Eigenschaft eines Steuerelements definiert werden.
- `<Trigger>` -Elemente können eine beliebige Anzahl von `<Setter>` -Elementen enthalten. Diese Elemente sind für das Festlegen von Eigenschaften für das enthaltende Element verantwortlich, wenn die Bedingung des `<Trigger>` -Elements erfüllt ist.
- Wenn eine Eigenschaft im Wurzelelement-Markup definiert ist, wird die im Element `<Setter>` definierte Eigenschaftsänderung nicht wirksam, auch wenn die Triggerbedingung erfüllt ist. Betrachten Sie das Markup `<TextBlock Text="Sample">` . Die `Text` Eigenschaft des Vorgangscodes ändert sich niemals basierend auf einem Trigger, da Root-Eigenschaftendefinitionen Vorrang vor Eigenschaften haben, die in Stilen definiert sind.
- Sobald ein Trigger verwendet wurde, kann er nicht mehr geändert werden.

Examples

Auslösen

Der einfachste der fünf Triggertypen, der `Trigger` ist dafür verantwortlich, Eigenschaften basierend auf anderen Eigenschaften **innerhalb desselben Steuerelements festzulegen** .

```
<TextBlock>
    <TextBlock.Style>
        <Style TargetType="{x:Type TextBlock}">
            <Style.Triggers>
                <Trigger Property="Text" Value="Pass">
                    <Setter Property="Foreground" Value="Green"/>
                </Trigger>
            </Style.Triggers>
        </Style>
    </TextBlock.Style>
</TextBlock>
```

In diesem Beispiel wird die Vordergrundfarbe des `TextBlock` grün, wenn die Eigenschaft `Text` der

Zeichenfolge `"Pass"` .

MultiTrigger

Ein `MultiTrigger` ähnelt einem Standard- `Trigger` , da er nur für Eigenschaften **innerhalb desselben Steuerelements gilt** . Der Unterschied ist, dass ein `MultiTrigger` mehrere Bedingungen hat, die erfüllt sein müssen, bevor der Trigger ausgelöst wird. Bedingungen werden mit dem `<Condition>` -Tag definiert.

```
<TextBlock x:Name="_txtBlock" IsEnabled="False">
    <TextBlock.Style>
        <Style TargetType="{x:Type TextBlock}">
            <Style.Triggers>
                <MultiTrigger>
                    <MultiTrigger.Conditions>
                        <Condition Property="Text" Value="Pass"/>
                        <Condition Property="IsEnabled" Value="True"/>
                    </MultiTrigger.Conditions>
                    <Setter Property="Foreground" Value="Green"/>
                </MultiTrigger>
            </Style.Triggers>
        </Style>
    </TextBlock.Style>
</TextBlock>
```

Beachten Sie, dass der `MultiTrigger` erst aktiviert wird, wenn beide Bedingungen erfüllt sind.

DataTrigger

Ein `DataTrigger` kann an jede Eigenschaft angehängt werden, sei es auf einem eigenen Steuerelement, einem anderen Steuerelement oder sogar auf einer Eigenschaft in einer Nicht-UI-Klasse. Betrachten Sie die folgende einfache Klasse.

```
public class Cheese
{
    public string Name { get; set; }
    public double Age { get; set; }
    public int StinkLevel { get; set; }
}
```

Welche werden wir als anhängen `DataContext` in der folgenden `TextBlock` .

```
<TextBlock Text="{Binding Name}">
    <TextBlock.DataContext>
        <local:Cheese Age="12" StinkLevel="100" Name="Limburger"/>
    </TextBlock.DataContext>
    <TextBlock.Style>
        <Style TargetType="{x:Type TextBlock}">
            <Style.Triggers>
                <DataTrigger Binding="{Binding StinkLevel}" Value="100">
                    <Setter Property="Foreground" Value="Green"/>
                </DataTrigger>
            </Style.Triggers>
        </Style>
```

```
    </TextBlock.Style>
</TextBlock>
```

Im `TextBlock.Foreground` Code ist die Eigenschaft `TextBlock.Foreground` grün. Wenn wir die `StinkLevel` Eigenschaft in unserer XAML auf etwas anderes als 100 `Text.Foreground` , wird die `Text.Foreground` Eigenschaft auf ihren Standardwert zurückgesetzt.

Löst aus online lesen: https://riptutorial.com/de/wpf/topic/9624/lost-aus

Kapitel 8: Markup-Erweiterungen

Parameter

Methode	Beschreibung
ProvideValue	Die MarkupExtension-Klasse hat nur eine Methode, die überschrieben werden sollte. XAML-Parser verwendet dann den von dieser Methode bereitgestellten Wert, um das Ergebnis der Markup-Erweiterung auszuwerten.

Bemerkungen

Eine Markup-Erweiterung kann implementiert werden, um Werte für Eigenschaften in einer Attributverwendung, Eigenschaften in einer Eigenschaftselementverwendung oder beides bereitzustellen.

Bei der Angabe eines Attributwerts ist die Syntax, die eine Markup-Erweiterungssequenz für einen XAML-Prozessor auszeichnet, das Vorhandensein der öffnenden und schließenden geschweiften Klammern ({und}). Der Typ der Markup-Erweiterung wird dann durch das Zeichenfolge-Token identifiziert, das unmittelbar auf die öffnende geschweifte Klammer folgt.

Bei Verwendung in der Eigenschaftselementsyntax ist eine Markuperweiterung visuell dieselbe wie jedes andere Element, das zum Bereitstellen eines Eigenschaftselementwerts verwendet wird: Eine XAML-Elementdeklaration, die auf die Markuperweiterungsklasse als Element verweist, in spitzen Klammern (<>) eingeschlossen.

Weitere Informationen finden Sie unter https://msdn.microsoft.com/en-us/library/ms747254(v=vs.110).aspx

Examples

Markup-Erweiterung, die mit IValueConverter verwendet wird

Eine der besten Anwendungen für Markup-Erweiterungen ist die einfachere Verwendung von IValueConverter. Im folgenden Beispiel ist BoolToVisibilityConverter ein Wertkonverter. Da er jedoch instanzunabhängig ist, kann er ohne die normalen Hashwerte eines Wertkonverters mithilfe der Markup-Erweiterung verwendet werden. In XAML einfach verwenden

```
Visibility="{Binding [BoolProperty], Converter={[xmlns]:BoolToVisibilityConverter}}"
```

und Sie können die Elementsichtbarkeit auf den bool-Wert einstellen.

```
public class BoolToVisibilityConverter : MarkupExtension, IValueConverter
```

```
    {
        public object Convert(object value, Type targetType, object parameter, CultureInfo
culture)
        {
            if (value is bool)
                return (bool)value ? Visibility.Visible : Visibility.Collapsed;
            else
                return Visibility.Collapsed;
        }

        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo
culture)
        {
            if (value is Visibility)
            {
                if ((Visibility)value == Visibility.Visible)
                    return true;
                else
                    return false;
            }
            else
                return false;
        }

        public override object ProvideValue(IServiceProvider serviceProvider)
        {
            return this;
        }
    }
```

XAML-definierte Markup-Erweiterungen

Es gibt vier vordefinierte Markup-Erweiterungen in XAML:

`x:Type` liefert das Type-Objekt für den genannten Typ. Diese Funktion wird am häufigsten in Stilen und Vorlagen verwendet.

```
<object property="{x:Type prefix:typeNameValue}" .../>
```

`x:Static` erzeugt statische Werte. Die Werte stammen von Werttyp-Code-Entitäten, die nicht direkt dem Wert des Zielwerts entsprechen, sondern für diesen Typ ausgewertet werden können.

```
<object property="{x:Static prefix:typeName.staticMemberName}" .../>
```

`x:Null` gibt null als Wert für eine Eigenschaft an und kann entweder für Attribute oder Eigenschaftselementwerte verwendet werden.

```
<object property="{x:Null}" .../>
```

`x:Array` bietet Unterstützung für die Erstellung allgemeiner Arrays in XAML-Syntax für Fälle, in denen die von WPF-Basiselementen und Kontrollmodellen bereitgestellte Sammlungsunterstützung absichtlich nicht verwendet wird.

```
<x:Array Type="typeName">
  arrayContents
</x:Array>
```

Markup-Erweiterungen online lesen: https://riptutorial.com/de/wpf/topic/6619/markup-erweiterungen

Kapitel 9: MVVM in WPF

Bemerkungen

Modelle und Ansichtsmodelle

Die Definition eines Modells wird häufig heiß diskutiert, und die Grenze zwischen einem Modell und einem Ansichtsmodell kann verschwimmen. Manche bevorzugen nicht zu „verunreinigen“ ihre Modelle mit der `INotifyPropertyChanged` - Schnittstelle, und stattdessen die Modell - Eigenschaften in der Ansicht-Modell duplizieren, die diese Schnittstelle *nicht* implementiert. Wie viele Dinge in der Softwareentwicklung gibt es keine richtige oder falsche Antwort. Seien Sie pragmatisch und tun Sie, was sich richtig anfühlt.

Separation anzeigen

Die Absicht von MVVM besteht darin, diese drei unterschiedlichen Bereiche - Modell, Ansichtsmodell und Ansicht - voneinander zu trennen. Während es für die Ansicht zulässig ist, auf das Ansichtsmodell (VM) und (indirekt) auf das Modell zuzugreifen, besteht die wichtigste Regel in MVVM darin, dass die VM keinen Zugriff auf die Ansicht oder deren Steuerelemente haben sollte. Die VM sollte über öffentliche Eigenschaften alles bereitstellen, was die Ansicht benötigt. Die VM sollte UI-Steuerelemente wie `TextBox` , `Button` usw. nicht direkt `TextBox` oder `TextBox` .

In manchen Fällen kann es schwierig sein, mit dieser strikten Trennung zu arbeiten, insbesondere wenn Sie einige komplexe UI-Funktionen in Betrieb nehmen müssen. Hier ist es durchaus akzeptabel, auf Ereignisse und Ereignishandler in der "Code-Behind" -Datei der Ansicht zurückzugreifen. Wenn es sich um eine reine UI-Funktionalität handelt, werden auf jeden Fall Ereignisse in der Ansicht verwendet. Es ist auch akzeptabel, dass diese Ereignishandler öffentliche Methoden für die VM-Instanz aufrufen. Vergeben Sie die Verweise nicht an die Steuerelemente der Benutzeroberfläche oder ähnliches.

RelayCommand

Leider ist die in diesem Beispiel verwendete `RelayCommand` Klasse nicht Teil des WPF-Frameworks (sollte es auch sein!), Aber Sie finden sie in fast jedem Werkzeugkasten des WPF-Entwicklers. Bei einer schnellen Online-Suche werden zahlreiche Code-Snippets angezeigt, die Sie anheben können, um eigene zu erstellen.

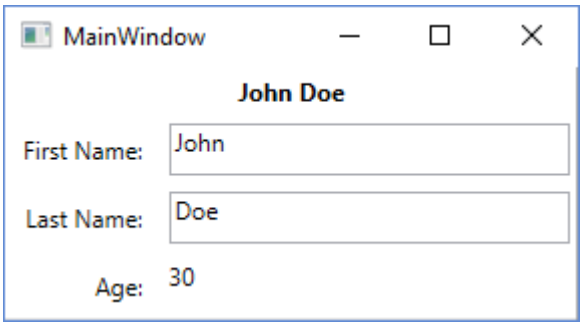
Eine nützliche Alternative zu `RelayCommand` ist `ActionCommand` das als Teil von `Microsoft.Expression.Interactivity.Core` bereitgestellt wird und vergleichbare Funktionen bietet.

Examples

Grundlegendes MVVM-Beispiel mit WPF und C

Dies ist ein einfaches Beispiel für die Verwendung des MVVM-Modells in einer Windows-Desktopanwendung mit WPF und C #. Der Beispielcode implementiert einen einfachen

"Benutzerinfo" -Dialog.



Die Aussicht

Die XAML

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
         <RowDefinition Height="Auto"/>
         <RowDefinition Height="Auto"/>
     </Grid.RowDefinitions>

    <TextBlock Grid.Column="0" Grid.Row="0" Grid.ColumnSpan="2" Margin="4" Text="{Binding
FullName}" HorizontalAlignment="Center" FontWeight="Bold"/>

    <Label Grid.Column="0" Grid.Row="1" Margin="4" Content="First Name:"
HorizontalAlignment="Right"/>
    <!-- UpdateSourceTrigger=PropertyChanged makes sure that changes in the TextBoxes are
immediately applied to the model. -->
    <TextBox Grid.Column="1" Grid.Row="1" Margin="4" Text="{Binding FirstName,
UpdateSourceTrigger=PropertyChanged}" HorizontalAlignment="Left" Width="200"/>

    <Label Grid.Column="0" Grid.Row="2" Margin="4" Content="Last Name:"
HorizontalAlignment="Right"/>
    <TextBox Grid.Column="1" Grid.Row="2" Margin="4" Text="{Binding LastName,
UpdateSourceTrigger=PropertyChanged}" HorizontalAlignment="Left"  Width="200"/>

    <Label Grid.Column="0" Grid.Row="3" Margin="4" Content="Age:"
HorizontalAlignment="Right"/>
    <TextBlock Grid.Column="1" Grid.Row="3" Margin="4" Text="{Binding Age}"
HorizontalAlignment="Left"/>

</Grid>
```

und der Code dahinter

```
public partial class MainWindow : Window
{
    private readonly MyViewModel _viewModel;

    public MainWindow() {
        InitializeComponent();
```

```
        _viewModel = new MyViewModel();
        // The DataContext serves as the starting point of Binding Paths
        DataContext = _viewModel;
    }
}
```

Das Ansichtsmodell

```
// INotifyPropertyChanged notifies the View of property changes, so that Bindings are updated.
sealed class MyViewModel : INotifyPropertyChanged
{
    private User user;

    public string FirstName {
        get {return user.FirstName;}
        set {
            if(user.FirstName != value) {
                user.FirstName = value;
                OnPropertyChange("FirstName");
                // If the first name has changed, the FullName property needs to be udpated as
well.
                OnPropertyChange("FullName");
            }
        }
    }

    public string LastName {
        get { return user.LastName; }
        set {
            if (user.LastName != value) {
                user.LastName = value;
                OnPropertyChange("LastName");
                // If the first name has changed, the FullName property needs to be udpated as
well.
                OnPropertyChange("FullName");
            }
        }
    }

    // This property is an example of how model properties can be presented differently to the
View.
    // In this case, we transform the birth date to the user's age, which is read only.
    public int Age {
        get {
            DateTime today = DateTime.Today;
            int age = today.Year - user.BirthDate.Year;
            if (user.BirthDate > today.AddYears(-age)) age--;
            return age;
        }
    }

    // This property is just for display purposes and is a composition of existing data.
    public string FullName {
        get { return FirstName + " " + LastName; }
    }

    public MyViewModel() {
        user = new User {
            FirstName = "John",
            LastName = "Doe",
```

```
            BirthDate = DateTime.Now.AddYears(-30)
        };
    }

    public event PropertyChangedEventHandler PropertyChanged;

    protected void OnPropertyChange(string propertyName) {
        if(PropertyChanged != null) {
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

Das Model

```
sealed class User
{
    public string FirstName { get; set; }

    public string LastName { get; set; }

    public DateTime BirthDate { get; set; }
}
```

Das Ansichtsmodell

Das Ansichtsmodell ist die "VM" in MV **VM** . Hierbei handelt es sich um eine Klasse, die als Vermittler fungiert, die Modelle der Benutzeroberfläche (Ansicht) zur Verfügung stellt und Anforderungen aus der Ansicht verarbeitet, z. B. Befehle, die durch Klicken mit der Schaltfläche ausgelöst werden. Hier ist ein grundlegendes Ansichtsmodell:

```
public class CustomerEditViewModel
{
    /// <summary>
    /// The customer to edit.
    /// </summary>
    public Customer CustomerToEdit { get; set; }

    /// <summary>
    /// The "apply changes" command
    /// </summary>
    public ICommand ApplyChangesCommand { get; private set; }

    /// <summary>
    /// Constructor
    /// </summary>
    public CustomerEditViewModel()
    {
        CustomerToEdit = new Customer
                         {
                             Forename = "John",
                             Surname = "Smith"
                         };

        ApplyChangesCommand = new RelayCommand(
            o => ExecuteApplyChangesCommand(),
            o => CustomerToEdit.IsValid);
```

```
    }

    /// <summary>
    /// Executes the "apply changes" command.
    /// </summary>
    private void ExecuteApplyChangesCommand()
    {
        // E.g. save your customer to database
    }
}
```

Der Konstruktor erstellt ein `Customer` Modellobjekt und weist es der `CustomerToEdit` Eigenschaft zu, sodass es für die Ansicht sichtbar ist.

Der Konstruktor erstellt auch ein `RelayCommand` Objekt und ordnet es der `ApplyChangesCommand` Eigenschaft zu. `ApplyChangesCommand` wird es erneut für die Ansicht sichtbar. WPF-Befehle werden verwendet, um Anforderungen aus der Ansicht zu bearbeiten, z. B. das Klicken von Schaltflächen oder Menüs.

Der `RelayCommand` benötigt zwei Parameter - der erste ist der Delegat, der aufgerufen wird, wenn der Befehl ausgeführt wird (z. B. als Reaktion auf einen Tastenklick). Der zweite Parameter ist ein Delegat, der einen booleschen Wert zurückgibt, der angibt, ob der Befehl ausgeführt werden kann. In diesem Beispiel ist es mit der `IsValid` Eigenschaft des `IsValid` . Wenn dies false zurückgibt, wird die Schaltfläche oder das Menüelement deaktiviert, das an diesen Befehl gebunden ist (andere Steuerelemente verhalten sich möglicherweise anders). Dies ist eine einfache, aber effektive Funktion, die das Schreiben von Code zum Aktivieren oder Deaktivieren von Steuerelementen aufgrund verschiedener Bedingungen vermeidet.

Wenn Sie dieses Beispiel in Betrieb nehmen, versuchen Sie, eine der `TextBox` zu `TextBox` (um das `Customer` in einen ungültigen Zustand zu `TextBox`). Wenn Sie das `TextBox` , sollten Sie feststellen, dass die Schaltfläche "Übernehmen" deaktiviert ist.

Bemerkung zur Kundenerstellung

Das Ansichtsmodell implementiert `INotifyPropertyChanged` (INPC) nicht. Das bedeutet, wenn der `CustomerToEdit` Eigenschaft ein anderes `Customer` Objekt zugewiesen werden sollte, würden sich die Steuerelemente der Ansicht nicht ändern, um das neue Objekt `TextBox` Die `TextBox` würde immer noch den `TextBox` und Nachnamen des vorherigen Kunden enthalten.

Der Beispielcode funktioniert, da der `Customer` im Konstruktor des Ansichtsmodells erstellt wird, bevor er dem `DataContext` der Ansicht zugewiesen wird (an diesem Punkt werden die Bindungen verdrahtet). In einer realen Anwendung können Sie Kunden aus einer Datenbank mit anderen Methoden als dem Konstruktor abrufen. Um dies zu unterstützen, sollte die VM INPC implementieren, und die `CustomerToEdit` Eigenschaft sollte geändert werden, um das "erweiterte" Getter- und Setter-Muster zu verwenden, das Sie im Beispielcode für das Modell sehen. Dadurch wird das `PropertyChanged` Ereignis im Setter ausgelöst.

Der `ApplyChangesCommand` des `ApplyChangesCommand` muss INPC nicht implementieren, da es unwahrscheinlich ist, dass sich der Befehl ändert. Sie *müssten* dieses Muster implementieren , wenn Sie den Befehl an einer anderen Stelle zu schaffen wurden als Konstruktor zum Beispiel

eine Art von `Initialize()` Methode.

Die allgemeine Regel lautet: Implementieren Sie INPC, wenn die Eigenschaft an beliebige Ansichtssteuerelemente gebunden ist *und* der Wert der Eigenschaft an einer anderen Stelle als im Konstruktor geändert werden kann. Sie müssen INPC nicht implementieren, wenn der Eigenschaftswert immer nur im Konstruktor zugewiesen wird (und Sie sich etwas Tipparbeit ersparen).

Das Model

Das Modell ist das erste "M" in **M** VVM. Das Modell ist normalerweise eine Klasse, die die Daten enthält, die Sie über eine Art Benutzeroberfläche anzeigen möchten.

Hier ist eine sehr einfache Modellklasse, die einige Eigenschaften zeigt: -

```
public class Customer : INotifyPropertyChanged
{
    private string _forename;
    private string _surname;
    private bool _isValid;

    public event PropertyChangedEventHandler PropertyChanged;

    /// <summary>
    /// Customer forename.
    /// </summary>
    public string Forename
    {
        get
        {
            return _forename;
        }
        set
        {
            if (_forename != value)
            {
                _forename = value;
                OnPropertyChanged();
                SetIsValid();
            }
        }
    }

    /// <summary>
    /// Customer surname.
    /// </summary>
    public string Surname
    {
        get
        {
            return _surname;
        }
        set
        {
            if (_surname != value)
            {
                _surname = value;
```

```
                OnPropertyChanged();
                SetIsValid();
            }
        }
    }

    /// <summary>
    /// Indicates whether the model is in a valid state or not.
    /// </summary>
    public bool IsValid
    {
        get
        {
            return _isValid;
        }
        set
        {
            if (_isValid != value)
            {
                _isValid = value;
                OnPropertyChanged();
            }
        }
    }

    /// <summary>
    /// Sets the value of the IsValid property.
    /// </summary>
    private void SetIsValid()
    {
        IsValid = !string.IsNullOrEmpty(Forename) && !string.IsNullOrEmpty(Surname);
    }

    /// <summary>
    /// Raises the PropertyChanged event.
    /// </summary>
    /// <param name="propertyName">Name of the property.</param>
    private void OnPropertyChanged([CallerMemberName] string propertyName = "")
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

Diese Klasse implementiert die `INotifyPropertyChanged` Schnittstelle, die ein `PropertyChanged` Ereignis `INotifyPropertyChanged` macht. Dieses Ereignis sollte ausgelöst werden, wenn sich einer der Eigenschaftswerte ändert. Sie können dies in Aktion im obigen Code sehen. Das `PropertyChanged` Ereignis ist ein Schlüsselelement in den WPF-Datenbindungsmechanismen, da die Benutzeroberfläche ohne sie die Änderungen des Werts einer Eigenschaft nicht widerspiegeln kann.

Das Modell enthält auch eine sehr einfache Validierungsroutine, die von den Eigenschaftssetzern aufgerufen wird. Es legt eine öffentliche Eigenschaft fest, die angibt, ob das Modell in einem gültigen Status ist oder nicht. Ich habe diese Funktionalität hinzugefügt, um eine "spezielle" Funktion von WPF- *Befehlen* zu demonstrieren, die Sie in Kürze sehen werden. *Das WPF-Framework bietet eine Reihe komplexerer Ansätze für die Validierung, die jedoch nicht in den Rahmen dieses Artikels fallen* .

Die Aussicht

Die Ansicht ist das "V" in M **V** VM. Dies ist Ihre Benutzeroberfläche. Sie können den Visual Studio-Drag-and-Drop-Designer verwenden, aber die meisten Entwickler müssen schließlich die reine XAML-Datei kodieren - eine Erfahrung, die dem Schreiben von HTML ähnelt.

Hier ist die XAML einer einfachen Ansicht, um das Bearbeiten eines `Customer` zu ermöglichen. Anstatt eine neue Ansicht zu erstellen, kann diese einfach in die `MainWindow.xaml` Datei eines WPF-Projekts `MainWindow.xaml` werden, zwischen den Tags `<Window ...>` und `</Window>` : -

```
<StackPanel Orientation="Vertical"
            VerticalAlignment="Top"
            Margin="20">
    <Label Content="Forename"/>
    <TextBox Text="{Binding CustomerToEdit.Forename}"/>

    <Label Content="Surname"/>
    <TextBox Text="{Binding CustomerToEdit.Surname}"/>

    <Button Content="Apply Changes"
            Command="{Binding ApplyChangesCommand}" />
</StackPanel>
```

Dieser Code erstellt ein einfaches Dateneingabeformular, das aus zwei `TextBox` - einer für den Vornamen des Kunden und eine für den Nachnamen. Es gibt ein `Label` über jedem `TextBox` und ein „ Übernehmen“ `Button` am unteren Rand des Formulars.

`TextBox` Sie das erste `TextBox` und sehen Sie seine `Text` Eigenschaft an:

```
Text="{Binding CustomerToEdit.Forename}"
```

Anstatt den Text der `TextBox` auf einen festen Wert zu setzen, bindet diese spezielle geschweifte Klammer-Syntax den Text stattdessen an den "Pfad" `CustomerToEdit.Forename` . Was ist dieser Weg relativ? Es ist der "Datenkontext" der Ansicht - in diesem Fall unser Ansichtsmodell. Der Bindungspfad ist, wie Sie möglicherweise herausfinden können, die `CustomerToEdit` Eigenschaft des `Forename` . `Forename` Eigenschaft ist vom Typ `Customer` , der wiederum eine Eigenschaft namens `Forename` - daher die "gepunktete" `Forename` .

Und falls Sie auf der Suche `Button` ,s XAML, hat es einen `Command` , der an die gebunden ist `ApplyChangesCommand` Eigenschaft des View-Modell. Das ist alles, was Sie benötigen, um eine Schaltfläche mit dem Befehl der VM zu verbinden.

Der DataContext

Wie stellen Sie also das Ansichtsmodell als Datenkontext der Ansicht ein? Eine Möglichkeit besteht darin, sie in den "Code-Behind" der Ansicht zu setzen. Drücken Sie F7, um diese Codedatei anzuzeigen, und fügen Sie dem vorhandenen Konstruktor eine Zeile hinzu, um eine Instanz des Ansichtsmodells zu erstellen und sie der `DataContext` Eigenschaft des Fensters zuzuweisen. Es sollte am Ende so aussehen:

```
    public MainWindow()
    {
        InitializeComponent();

        // Our new line:-
        DataContext = new CustomerEditViewModel();
    }
```

In realen Systemen werden häufig andere Ansätze verwendet, um das Ansichtsmodell zu erstellen, z. B. Abhängigkeitsinjektion oder MVVM-Frameworks.

Befehlen in MVVM

Befehle werden zur Behandlung von `Events` in WPF unter Berücksichtigung des MVVM-Musters verwendet.

Ein normaler `EventHandler` würde so aussehen (in `Code-Behind`):

```
public MainWindow()
{
    _dataGrid.CollectionChanged += DataGrid_CollectionChanged;
}

private void DataGrid_CollectionChanged(object sender,
System.Collections.Specialized.NotifyCollectionChangedEventArgs e)
{
        //Do what ever
}
```

Nein, um das gleiche in MVVM zu tun, verwenden wir `Commands` :

```
 <Button Command="{Binding Path=CmdStartExecution}" Content="Start" />
```

> Ich empfehle, eine Art Präfix (`Cmd`) für Ihre Befehlseigenschaften zu verwenden, da Sie sie hauptsächlich in xaml benötigen - auf diese Weise sind sie leichter zu erkennen.

Da es sich um MVVM handelt, möchten Sie diesen Befehl (für `Button` "eq" `Button_Click`) in Ihrem `ViewModel` .

Dafür brauchen wir grundsätzlich zwei Dinge:

1. System.Windows.Input.ICommand
2. RelayCommand (zum Beispiel hier genommen) .

Ein einfaches **Beispiel** könnte so aussehen:

```
private RelayCommand _commandStart;
public ICommand CmdStartExecution
{
   get
   {
```

```
      if(_commandStart == null)
      {
          _commandStart = new RelayCommand(param => Start(), param => CanStart());
      }
      return _commandStart;
   }
}

public void Start()
{
   //Do what ever
}

public bool CanStart()
{
    return (DateTime.Now.DayOfWeek == DayOfWeek.Monday); //Can only click that button on
mondays.
}
```

Was macht das also im Detail:

Der `ICommand` das `Control` in xaml. Der `RelayCommand` leitet Ihren Befehl an eine `Action` (dh, eine `Method` aufzurufen). Die Nullprüfung stellt lediglich sicher, dass jeder `Command` nur einmal initialisiert wird (aufgrund von Leistungsproblemen). Wenn Sie den Link für `RelayCommand` oben gelesen haben, haben Sie möglicherweise bemerkt, dass `RelayCommand` zwei Überladungen für den Konstruktor hat. `(Action<object> execute)` und `(Action<object> execute, Predicate<object> canExecute)` .

Das heißt , Sie können (Aditionally) eine zweite hinzufügen `Method` Zurückgeben eines `bool` zu sagen , dass `Control` wheather das „Ereignis“ abfeuern kann oder nicht.

Das Gute daran ist, dass `Button` zum Beispiel `Enabled="false"` wenn die `Method false`

Befehlsparameter

```
<DataGrid x:Name="TicketsDataGrid">
    <DataGrid.InputBindings>
        <MouseBinding Gesture="LeftDoubleClick"
                      Command="{Binding CmdTicketClick}"
                      CommandParameter="{Binding ElementName=TicketsDataGrid,
                                                 Path=SelectedItem}" />
    </DataGrid.InputBindings>
<DataGrid />
```

In diesem Beispiel möchte ich das `DataGrid.SelectedItem` an den Click_Command in meinem ViewModel übergeben.

Ihre Methode sollte so aussehen, während die ICommand-Implementierung selbst so bleibt.

```
private RelayCommand _commandTicketClick;

public ICommand CmdTicketClick
{
   get
   {
```

```
        if(_commandTicketClick == null)
        {
            _commandTicketClick = new RelayCommand(param => HandleUserClick(param));
        }
        return _commandTicketClick;
    }
}

private void HandleUserClick(object item)
{
    MyModelClass selectedItem = item as MyModelClass;
    if (selectedItem != null)
    {
        //Do sth. with that item
    }
}
```

MVVM in WPF online lesen: https://riptutorial.com/de/wpf/topic/2134/mvvm-in-wpf

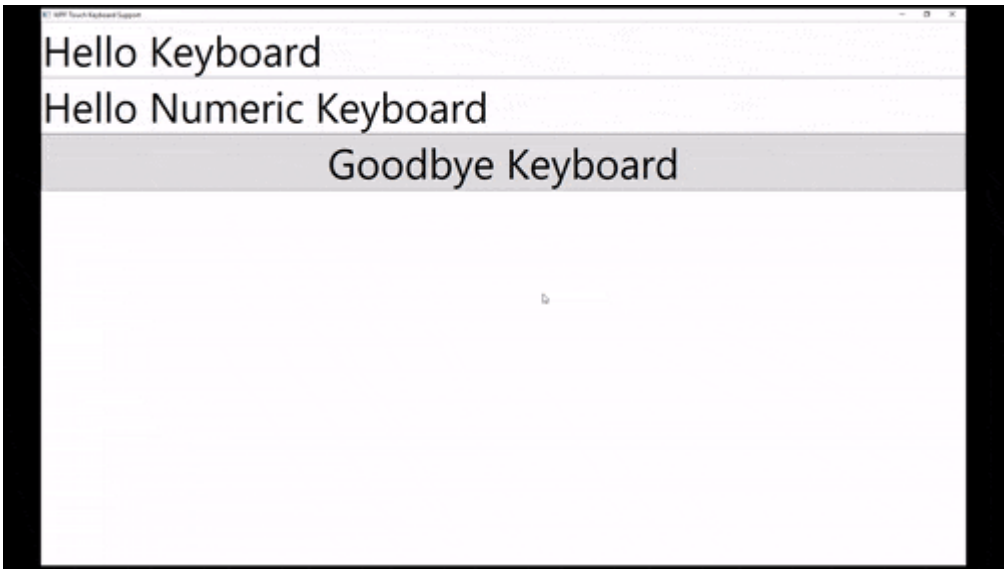
Kapitel 10: Optimierung für die Touch-Interaktion

Examples

Zeigen Sie die Touch-Tastatur unter Windows 8 und Windows 10 an

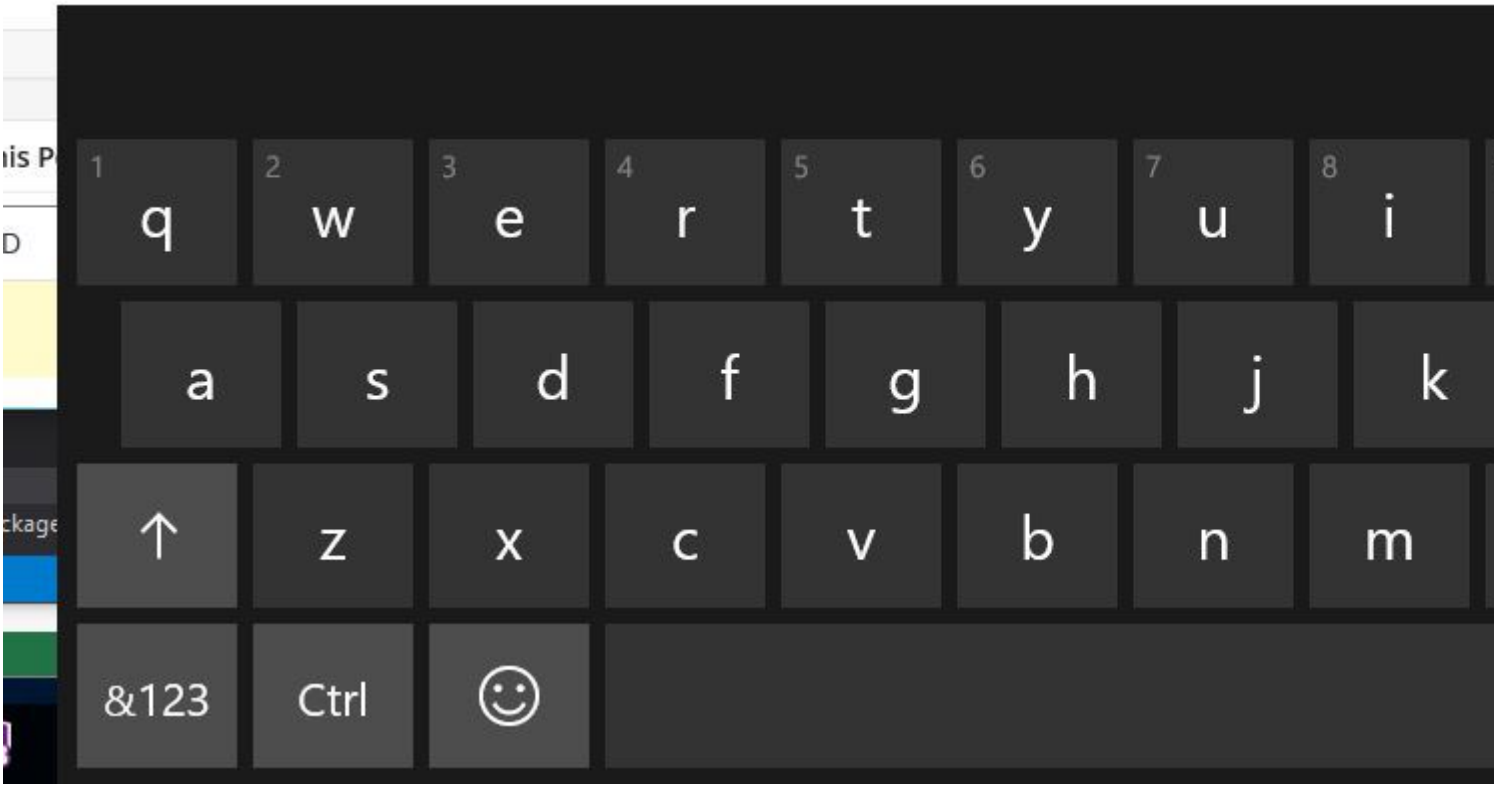
WPF-Apps für .NET Framework 4.6.2 und höher

Bei WPF-Apps, die auf .NET Framework 4.6.2 (und höher) abzielen, wird die Softtastatur automatisch aufgerufen und ohne zusätzliche Schritte verworfen.



WPF-Apps für .NET Framework 4.6.1 und früher

WPF ist nicht in erster Linie berührungsaktiviert. Wenn der Benutzer mit einer WPF-Anwendung auf dem Desktop interagiert, zeigt die App **die Touch-Tastatur nicht automatisch an,** wenn die `TextBox` Steuerelemente den Fokus erhalten. Dies ist für Benutzer von Tablets ein unbequemes Verhalten, das sie dazu zwingt, die Touch-Tastatur manuell über die System-Taskleiste zu öffnen.



Problemumgehung

Die Touch-Tastatur ist eigentlich eine klassische `exe` Anwendung, die auf jedem Windows 8- und Windows 10-PC unter dem folgenden Pfad zu finden ist: `C:\Program Files\Common Files\Microsoft Shared\Ink\TabTip.exe` .

Basierend auf diesem Wissen können Sie ein benutzerdefiniertes Steuerelement `TextBox` , das von `TextBox` abgeleitet `TextBox` , das das `GotTouchCapture` Ereignis `GotTouchCapture` (dieses Ereignis wird aufgerufen, wenn das Steuerelement durch Berührung den Fokus erhält) und **den Prozess der Berührungstastatur startet** .

```
public class TouchEnabledTextBox : TextBox
{
    public TouchEnabledTextBox()
    {
        this.GotTouchCapture += TouchEnabledTextBox_GotTouchCapture;
    }

    private void TouchEnabledTextBox_GotTouchCapture(
       object sender,
       System.Windows.Input.TouchEventArgs e )
    {
        string touchKeyboardPath =
           @"C:\Program Files\Common Files\Microsoft Shared\Ink\TabTip.exe";
        Process.Start( touchKeyboardPath );
    }
}
```

Sie können dies noch verbessern, indem Sie den erstellten Prozess zwischenspeichern und anschließend beenden, nachdem das Steuerelement den Fokus verliert:

```
//added field
private Process _touchKeyboardProcess = null;

//replace Process.Start line from the previous listing with
_touchKeyboardProcess = Process.Start( touchKeyboardPath );
```

Jetzt können Sie das `LostFocus` Ereignis verbinden:

```
//add this at the end of TouchEnabledTextBox's constructor
this.LostFocus += TouchEnabledTextBox_LostFocus;

//add this method as a member method of the class
private void TouchEnabledTextBox_LostFocus( object sender, RoutedEventArgs eventArgs ){
   if ( _touchKeyboardProcess != null )
   {
      _touchKeyboardProcess.Kill();
      //nullify the instance pointing to the now-invalid process
      _touchKeyboardProcess = null;
   }
}
```

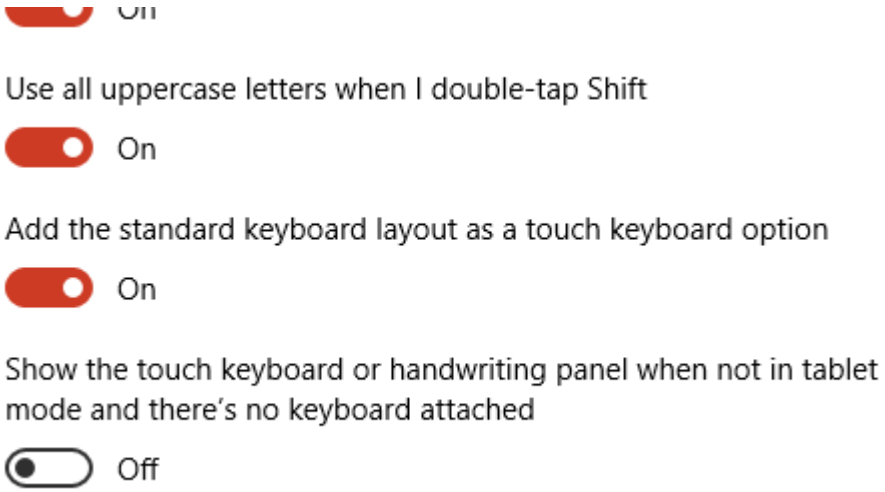
Hinweis zum Tablet-Modus in Windows 10

Windows 10 hat einen **Tablet-Modus eingeführt** , der die Interaktion mit dem System vereinfacht, wenn der PC mit der Touch-first-Methode verwendet wird. Dieser Modus stellt neben anderen Verbesserungen sicher, dass die **Touch-Tastatur** auch für klassische Desktop-Apps einschließlich WPF-Apps **automatisch angezeigt wird** .

Ansatz für Windows 10-Einstellungen

Zusätzlich zum Tablet-Modus kann Windows 10 die Touch-Tastatur für klassische Apps auch außerhalb des Tablet-Modus automatisch anzeigen. Dieses Verhalten, das standardmäßig deaktiviert ist, kann in der App Einstellungen aktiviert werden.

Wechseln Sie in der App **Einstellungen** zu der Kategorie **Geräte** und wählen Sie **Eingabe** . Wenn Sie ganz nach unten scrollen, finden Sie die Einstellung "Touch-Tastatur oder Handschriftfeld anzeigen, wenn sich der Tablet-Modus nicht befindet und keine Tastatur angeschlossen ist", die Sie aktivieren können.



Es ist erwähnenswert, dass diese Einstellung nur auf Geräten mit Touch-Funktion sichtbar ist.

Optimierung für die Touch-Interaktion online lesen:
https://riptutorial.com/de/wpf/topic/6799/optimierung-fur-die-touch-interaktion

Kapitel 11: Rastersteuerung

Examples

Ein einfaches Gitter

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <TextBlock Grid.Column="1" Text="abc"/>
    <TextBlock Grid.Row="1" Grid.Column="1" Text="def"/>
</Grid>
```

Zeilen und Spalten werden definiert, indem den entsprechenden Sammlungen `RowDefinition` und `ColumnDefinition` Elemente `ColumnDefinition` werden.

Es können beliebig viele Kinder im `Grid` . Um anzugeben, welche Zeile oder Spalte ein `Grid.Row` `Grid.Column` in den angefügten Eigenschaften `Grid.Row` und `Grid.Column` werden diese verwendet. Zeilen- und Spaltennummern basieren auf Null. Wenn keine Zeile oder Spalte gesetzt ist, ist der Standardwert `0` .

In derselben Zeile und Spalte platzierte Kinder werden in der Definitionsreihenfolge gezeichnet. Das zuletzt definierte Kind wird also über das zuvor definierte Kind gezeichnet.

Gitterkinder, die sich über mehrere Zeilen / Spalten erstrecken

Durch die Verwendung der `Grid.RowSpan` und `Grid.ColumnSpan` angefügten Eigenschaften, Kinder eines `Grid` können mehrere Zeilen oder Spalten erstrecken. In dem folgenden Beispiel wird die zweite `TextBlock` wird die zweite und dritte Spalte der Spannweite `Grid` .

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <TextBlock Grid.Column="2" Text="abc"/>
    <TextBlock Grid.Column="1" Grid.ColumnSpan="2" Text="def"/>
</Grid>
```

Zeilen oder Spalten mehrerer Raster synchronisieren

Die Zeilenhöhen oder Spaltenbreiten mehrerer `Grid` können synchronisiert werden, indem eine

gemeinsame `SharedSizeGroup` für die zu synchronisierenden Zeilen oder Spalten festgelegt wird. Dann muss ein übergeordnetes Steuerelement oben in der Baumstruktur oberhalb der `Grid` die angefügte Eigenschaft `Grid.IsSharedSizeScope` auf `True` .

```
<StackPanel Grid.IsSharedSizeScope="True">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" SharedSizeGroup="MyGroup"/>
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        [...]
    </Grid>
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" SharedSizeGroup="MyGroup"/>
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        [...]
    </Grid>
</StackPanel>
```

In diesem Beispiel hat die erste Spalte beider `Grid` immer die gleiche Breite, auch wenn eine der beiden durch den Inhalt angepasst wird.

Rastersteuerung online lesen: https://riptutorial.com/de/wpf/topic/6483/rastersteuerung

Kapitel 12: Slider-Bindung: Aktualisierung nur bei gezogenem Ziehen

Parameter

Parameter	Detail
Wert (Float)	Die an diese Abhängigkeitseigenschaft gebundene Eigenschaft wird aktualisiert, wenn der Benutzer den Schieberegler nicht mehr zieht

Bemerkungen

- Stellen Sie sicher, dass Sie auf die *System.Windows.Interactivity-* Assembly verweisen, damit der XAML-Parser die *xmlns: i-* Deklaration erkennt.
- Beachten Sie, dass die Anweisung *xmlns: b* mit dem Namespace übereinstimmt, in dem sich die Verhaltensimplementierung befindet
- Das Beispiel setzt Kenntnisse in Bezug auf Bindungsausdrücke und XAML voraus.

Examples

Verhaltensimplementierung

```
using System.Windows;
using System.Windows.Controls;
using System.Windows.Controls.Primitives;
using System.Windows.Interactivity;

namespace MyBehaviorAssembly
{

public class SliderDragEndValueBehavior : Behavior<Slider>
{

    public static readonly DependencyProperty ValueProperty = DependencyProperty.Register(
        "Value", typeof (float), typeof (SliderDragEndValueBehavior), new
PropertyMetadata(default(float)));

    public float Value
    {
        get { return (float) GetValue(ValueProperty); }
        set { SetValue(ValueProperty, value); }
    }

    protected override void OnAttached()
    {
        RoutedEventHandler handler = AssociatedObject_DragCompleted;
        AssociatedObject.AddHandler(Thumb.DragCompletedEvent, handler);
    }
```

```
    private void AssociatedObject_DragCompleted(object sender, RoutedEventArgs e)
    {
        Value = (float) AssociatedObject.Value;
    }

    protected override void OnDetaching()
    {
        RoutedEventHandler handler = AssociatedObject_DragCompleted;
        AssociatedObject.RemoveHandler(Thumb.DragCompletedEvent, handler);
    }
}
}
```

XAML-Nutzung

```
<UserControl x:Class="Example.View"
         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:i="http://schemas.microsoft.com/expression/2010/interactivity"

         xmlns:b="MyBehaviorAssembly;assembly=MyBehaviorAssembly"

         mc:Ignorable="d"
         d:DesignHeight="200" d:DesignWidth="200"

         >
           <Slider>
            <i:Interaction.Behaviors>
                <b:SliderDragEndValueBehavior

                        Value="{Binding Value, Mode=OneWayToSource,
UpdateSourceTrigger=PropertyChanged}"

                        />
            </i:Interaction.Behaviors>
          </Slider>

</UserControl>
```

Slider-Bindung: Aktualisierung nur bei gezogenem Ziehen online lesen:
https://riptutorial.com/de/wpf/topic/6339/slider-bindung--aktualisierung-nur-bei-gezogenem-ziehen

Kapitel 13: Sprachsynthese

Einführung

In der Assembly `System.Speech` hat Microsoft die **Sprachsynthese** hinzugefügt, mit der Text in gesprochene Wörter umgewandelt werden kann.

Syntax

1. SpeechSynthesizer SpeechSynthesizerObject = new SpeechSynthesizer ();

   speechSynthesizerObject.Speak ("Text zum Sprechen");

Examples

Beispiel für die Sprachsynthese - Hello World

```
using System;
using System.Speech.Synthesis;
using System.Windows;

namespace Stackoverflow.SpeechSynthesisExample
{
        public partial class SpeechSynthesisSample : Window
        {
                public SpeechSynthesisSample()
                {
                        InitializeComponent();
                        SpeechSynthesizer speechSynthesizer = new SpeechSynthesizer();
                        speechSynthesizer.Speak("Hello, world!");
                }
        }
}
```

Sprachsynthese online lesen: https://riptutorial.com/de/wpf/topic/8368/sprachsynthese

Kapitel 14: Startbild in WPF erstellen

Einführung

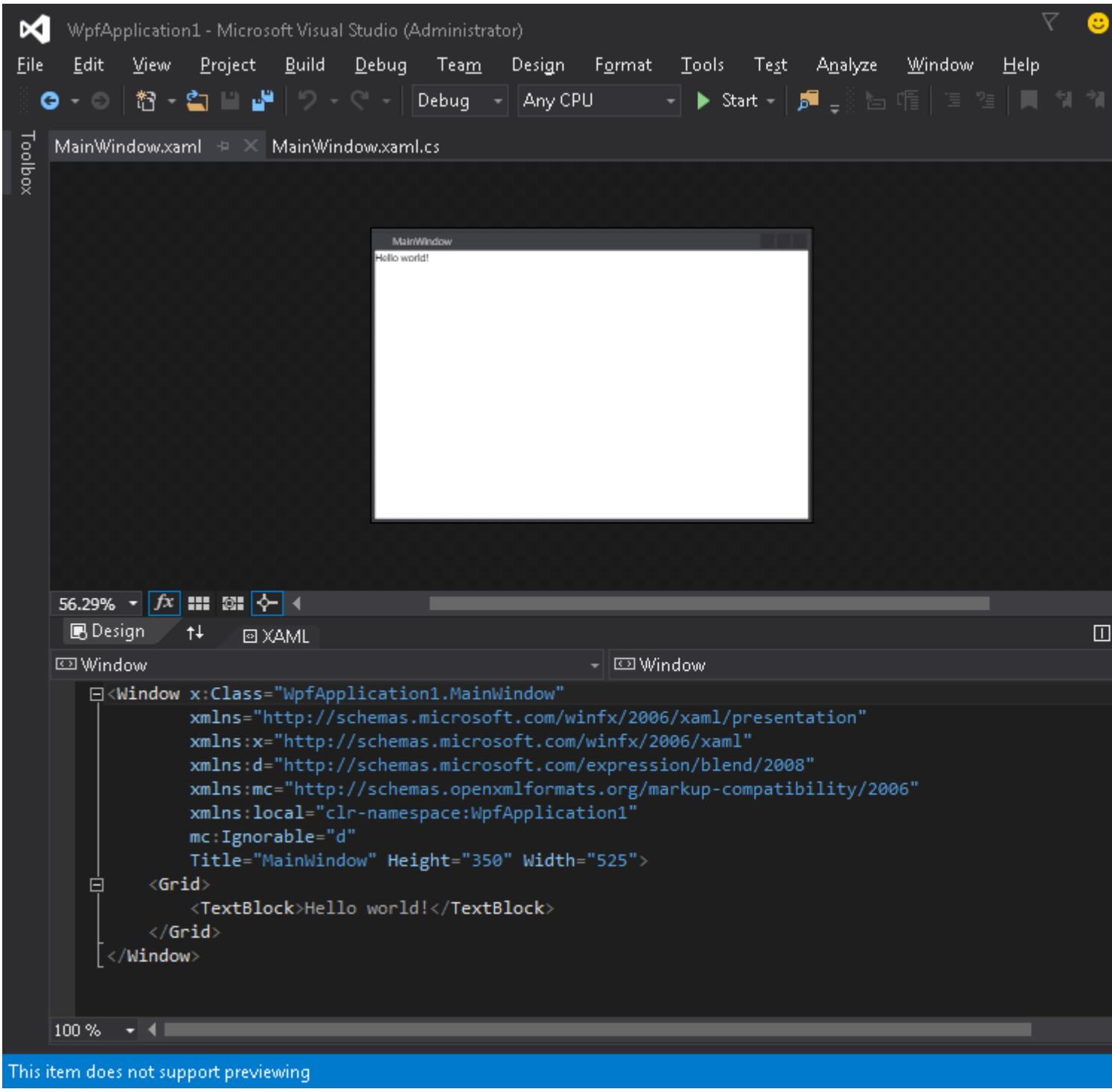
Beim Start der WPF-Anwendung kann es eine Weile dauern, bis eine aktuelle Sprachlaufzeit (CLR) für .NET Framework initialisiert ist. Daher kann das erste Anwendungsfenster einige Zeit nach dem Start der Anwendung angezeigt werden, abhängig von der Anwendungskomplexität. Der Begrüßungsbildschirm in WPF ermöglicht es der Anwendung, während der Initialisierung entweder statisches Bild oder benutzerdefinierten dynamischen Inhalt anzuzeigen, bevor das erste Fenster angezeigt wird.

Examples

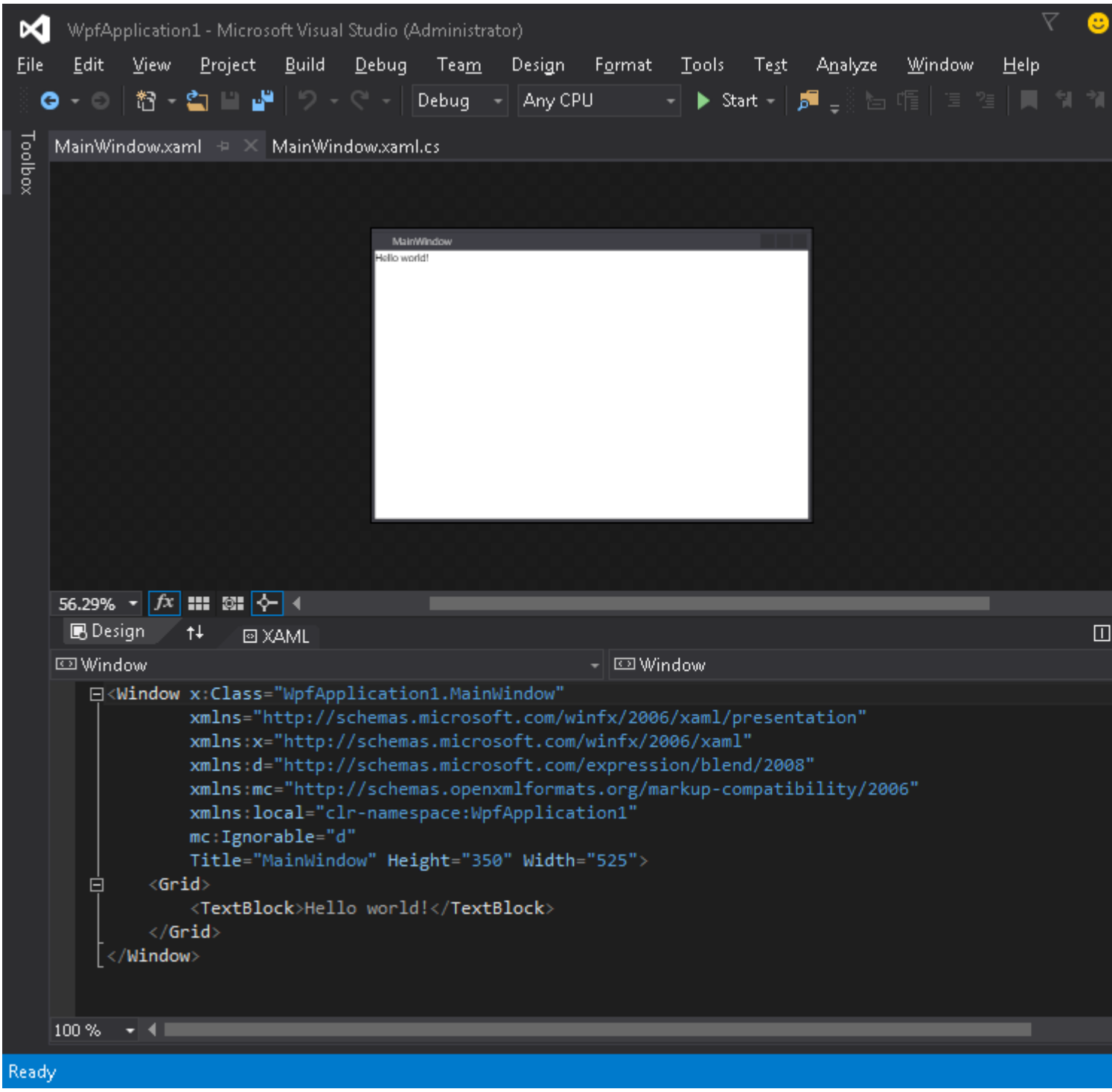
Einfacher Begrüßungsbildschirm hinzufügen

Führen Sie die folgenden Schritte aus, um der WPF-Anwendung in Visual Studio einen Begrüßungsbildschirm hinzuzufügen:

1. Erstellen oder erhalten Sie ein Bild und fügen Sie es Ihrem Projekt hinzu (z. B. im Ordner *Bilder*):



2. Öffnen Sie das Eigenschaftenfenster für dieses Bild (**Ansicht → Eigenschaftenfenster**) und ändern Sie die **Build-Aktion-** Einstellung in den **SplashScreen-** Wert:



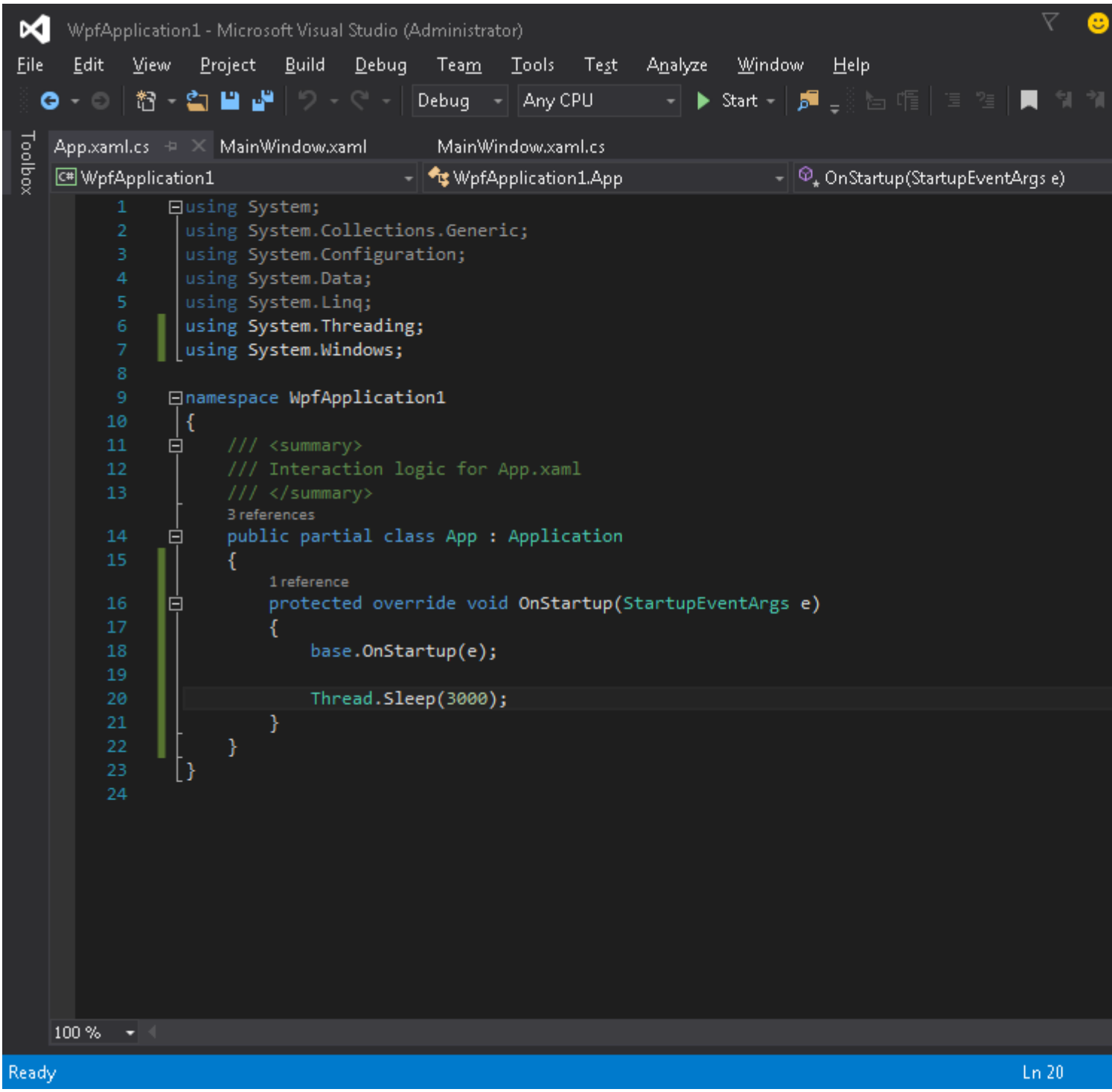
3. Führen Sie die Anwendung aus. Das Startbild wird in der Mitte des Bildschirms angezeigt, bevor das Anwendungsfenster angezeigt wird (nachdem das Fenster angezeigt wird, wird das Startbild innerhalb von 300 Millisekunden ausgeblendet).

Startbildschirm testen

Wenn Ihre Anwendung leicht und einfach ist, wird sie sehr schnell gestartet und mit ähnlicher Geschwindigkeit wird der Startbildschirm angezeigt und ausgeblendet.

Sobald der Begrüßungsbildschirm nach Abschluss der `Application.Startup` Methode verschwindet, können Sie die Anwendungsverzögerung der Anwendung simulieren, indem Sie die folgenden Schritte ausführen:

1. Öffnen Sie die **App.xaml.cs-** Datei
2. Fügen Sie *using* Namespace `using System.Threading;`
3. `OnStartup` Methode und fügen Sie `Thread.Sleep(3000);` im Inneren:



Code sollte folgendermaßen aussehen:

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Threading;
using System.Windows;
```

```
namespace WpfApplication1
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    public partial class App : Application
    {
        protected override void OnStartup(StartupEventArgs e)
        {
            base.OnStartup(e);

            Thread.Sleep(3000);
        }
    }
}
```

4. Führen Sie die Anwendung aus. Jetzt wird es etwa 3 Sekunden länger gestartet, sodass Sie mehr Zeit haben, um Ihren Begrüßungsbildschirm zu testen.

Erstellen eines benutzerdefinierten Begrüßungsbildschirmfensters

WPF unterstützt nicht die Anzeige eines anderen Bildes als Startbildschirm als Startbildschirm. Daher müssen wir ein `Window` erstellen, das als Startbildschirm dient. Wir gehen davon aus, dass wir bereits ein Projekt mit der `MainWindow` Klasse erstellt haben, das das `MainWindow` der Anwendung sein soll.

Zunächst fügen wir unserem Projekt ein `SplashScreenWindow` Fenster hinzu:

```
<Window x:Class="SplashScreenExample.SplashScreenWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        WindowStartupLocation="CenterScreen"
        WindowStyle="None"
        AllowsTransparency="True"
        Height="30"
        Width="200">
    <Grid>
        <ProgressBar IsIndeterminate="True" />
        <TextBlock HorizontalAlignment="Center"
                   VerticalAlignment="Center">Loading...</TextBlock>
    </Grid>
</Window>
```

Dann überschreiben wir die `Application.OnStartup` Methode, um den Begrüßungsbildschirm anzuzeigen, arbeiten und schließlich das Hauptfenster (***App.xaml.cs***) ***anzeigen*** :

```
public partial class App
{
    protected override void OnStartup(StartupEventArgs e)
    {
        base.OnStartup(e);

        //initialize the splash screen and set it as the application main window
        var splashScreen = new SplashScreenWindow();
```

```
        this.MainWindow = splashScreen;
        splashScreen.Show();

        //in order to ensure the UI stays responsive, we need to
        //do the work on a different thread
        Task.Factory.StartNew(() =>
        {
            //simulate some work being done
            System.Threading.Thread.Sleep(3000);

            //since we're not on the UI thread
            //once we're done we need to use the Dispatcher
            //to create and show the main window
            this.Dispatcher.Invoke(() =>
            {
                //initialize the main window, set it as the application main window
                //and close the splash screen
                var mainWindow = new MainWindow();
                this.MainWindow = mainWindow;
                mainWindow.Show();
                splashScreen.Close();
            });
        });
    }
}
```

Schließlich müssen wir uns um den Standardmechanismus kümmern, der das `MainWindow` beim Start der Anwendung `MainWindow` . Alles, was wir tun müssen, ist das `StartupUri="MainWindow.xaml"` -Attribut aus dem Root- `Application` Tag in der ***App.xaml-*** Datei zu entfernen.

Erstellen eines Startbildschirmfensters mit Fortschrittsbericht

WPF unterstützt nicht die Anzeige eines anderen Bildes als Startbildschirm als Startbildschirm. Daher müssen wir ein `Window` erstellen, das als Startbildschirm dient. Wir gehen davon aus, dass wir bereits ein Projekt mit der `MainWindow` Klasse erstellt haben, das das `MainWindow` der Anwendung sein soll.

Zunächst fügen wir unserem Projekt ein `SplashScreenWindow` Fenster hinzu:

```
<Window x:Class="SplashScreenExample.SplashScreenWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        WindowStartupLocation="CenterScreen"
        WindowStyle="None"
        AllowsTransparency="True"
        Height="30"
        Width="200">
    <Grid>
        <ProgressBar x:Name="progressBar" />
        <TextBlock HorizontalAlignment="Center"
                   VerticalAlignment="Center">Loading...</TextBlock>
    </Grid>
</Window>
```

Dann machen wir eine Eigenschaft in der `SplashScreenWindow` Klasse `SplashScreenWindow` , sodass der aktuelle Fortschrittswert (***SplashScreenWindow.xaml.cs***) problemlos aktualisiert werden

kann:

```
public partial class SplashScreenWindow : Window
{
    public SplashScreenWindow()
    {
        InitializeComponent();
    }

    public double Progress
    {
        get { return progressBar.Value; }
        set { progressBar.Value = value; }
    }
}
```

Als Nächstes überschreiben wir die `Application.OnStartup` Methode, um den Begrüßungsbildschirm anzuzeigen, einige Arbeiten ***auszuführen*** und schließlich das Hauptfenster (***App.xaml.cs***) ***anzuzeigen*** :

```
public partial class App : Application
{
    protected override void OnStartup(StartupEventArgs e)
    {
        base.OnStartup(e);

        //initialize the splash screen and set it as the application main window
        var splashScreen = new SplashScreenWindow();
        this.MainWindow = splashScreen;
        splashScreen.Show();

        //in order to ensure the UI stays responsive, we need to
        //do the work on a different thread
        Task.Factory.StartNew(() =>
        {
            //we need to do the work in batches so that we can report progress
            for (int i = 1; i <= 100; i++)
            {
                //simulate a part of work being done
                System.Threading.Thread.Sleep(30);

                //because we're not on the UI thread, we need to use the Dispatcher
                //associated with the splash screen to update the progress bar
                splashScreen.Dispatcher.Invoke(() => splashScreen.Progress = i);
            }

            //once we're done we need to use the Dispatcher
            //to create and show the main window
            this.Dispatcher.Invoke(() =>
            {
                //initialize the main window, set it as the application main window
                //and close the splash screen
                var mainWindow = new MainWindow();
                this.MainWindow = mainWindow;
                mainWindow.Show();
                splashScreen.Close();
            });
        });
    }
```

```
}
```

Schließlich müssen wir uns um den Standardmechanismus kümmern, der das `MainWindow` beim Start der Anwendung `MainWindow` . Alles, was wir tun müssen, ist das `StartupUri="MainWindow.xaml"` - Attribut aus dem Root- `Application` Tag in der ***App.xaml-*** Datei zu entfernen.

Startbild in WPF erstellen online lesen: https://riptutorial.com/de/wpf/topic/3948/startbild-in-wpf-erstellen

Kapitel 15: Styles in WPF

Bemerkungen

Einleitende Bemerkungen

In WPF definiert ein **Stil** die Werte einer oder mehrerer Abhängigkeitseigenschaften für ein bestimmtes visuelles Element. In der gesamten Anwendung werden Stile verwendet, um die Benutzeroberfläche konsistenter zu gestalten (z. B. für alle Dialogschaltflächen eine einheitliche Größe zu erhalten), und um Massenänderungen zu erleichtern (z. B. das Ändern der Breite aller Schaltflächen).

Stile werden in der Regel in einem `ResourceDictionary` auf hoher Ebene in der Anwendung definiert (z. B. in *App.xaml* oder in einem *Design*), sodass sie app-weit verfügbar sind. Sie können jedoch auch für ein einzelnes Element und dessen *untergeordnete* Elemente definiert werden, z Stil für alle `TextBlock` Elemente in einem `StackPanel` .

```
<StackPanel>
    <StackPanel.Resources>
        <Style TargetType="TextBlock">
            <Setter Property="Margin" Value="5,5,5,0"/>
            <Setter Property="Background" Value="#FFF0F0F0"/>
            <Setter Property="Padding" Value="5"/>
        </Style>
    </StackPanel.Resources>

    <TextBlock Text="First Child"/>
    <TextBlock Text="Second Child"/>
    <TextBlock Text="Third Child"/>
</StackPanel>
```

Wichtige Notizen

- Der Ort, an dem der Stil definiert ist, beeinflusst, wo er verfügbar ist.
- Weiterleitungsreferenzen können nicht von `StaticResource` aufgelöst werden. Wenn Sie also einen Stil definieren, der von einem anderen Stil oder einer anderen Ressource in einem Ressourcenwörterbuch abhängt, muss er hinter / unter der Ressource definiert werden, von der er abhängt.
- `StaticResource` ist die empfohlene Methode zum Referenzieren von Stilen und anderen Ressourcen (aus Leistungs- und Verhaltensgründen), sofern Sie nicht ausdrücklich die Verwendung von `DynamicResource` erfordern, z. B. für Designs, die zur Laufzeit geändert werden können.

Ressourcen

MSDN enthält ausführliche Artikel zu Stilen und Ressourcen, die mehr Tiefe bieten, als hier angegeben werden kann.

- Ressourcenübersicht
- Styling und Templating
- Control Authoring Übersicht

Examples

Definieren eines benannten Stils

Für einen benannten Stil muss die `x:Key` Eigenschaft festgelegt werden und gilt nur für Elemente, die explizit mit ihrem Namen darauf verweisen:

```
<StackPanel>
    <StackPanel.Resources>
        <Style x:Key="MyTextBlockStyle" TargetType="TextBlock">
            <Setter Property="Background" Value="Yellow"/>
            <Setter Property="FontWeight" Value="Bold"/>
        </Style>
    </StackPanel.Resources>

    <TextBlock Text="Yellow and bold!" Style="{StaticResource MyTextBlockStyle}" />
    <TextBlock Text="Also yellow and bold!" Style="{DynamicResource MyTextBlockStyle}" />
    <TextBlock Text="Plain text." />
</StackPanel>
```

Definieren eines impliziten Stils

Ein impliziter Stil gilt für alle Elemente eines bestimmten Typs innerhalb des Bereichs. Bei einem impliziten Stil kann `x:Key` da er implizit der `TargetType` Eigenschaft des Stils `TargetType` .

```
<StackPanel>
    <StackPanel.Resources>
        <Style TargetType="TextBlock">
            <Setter Property="Background" Value="Yellow"/>
            <Setter Property="FontWeight" Value="Bold"/>
        </Style>
    </StackPanel.Resources>

    <TextBlock Text="Yellow and bold!"  />
    <TextBlock Text="Also yellow and bold!" />
    <TextBlock Style="{x:Null}" Text="I'm not yellow or bold; I'm the control's default
style!" />
</StackPanel>
```

Vererbung von einem Stil

Es ist üblich, einen `TextBlock` zu benötigen, der Eigenschaften / Werte definiert, die von mehreren Stilen gemeinsam sind, die zu demselben Steuerelement gehören, insbesondere für `TextBlock` . Dies wird mithilfe der `BasedOn` Eigenschaft erreicht. Werte werden vererbt und können dann

überschrieben werden.

```
<Style x:Key="BaseTextBlockStyle" TargetType="TextBlock">
    <Setter Property="FontSize" Value="12"/>
    <Setter Property="Foreground" Value="#FFBBBBBB" />
    <Setter Property="FontFamily" Value="Arial" />
</Style>

<Style x:Key="WarningTextBlockStyle"
       TargetType="TextBlock"
       BasedOn="{StaticResource BaseTextBlockStyle">
    <Setter Property="Foreground" Value="Red"/>
    <Setter Property="FontWeight" Value="Bold" />
</Style>
```

Im obigen Beispiel würde jeder `TextBlock` mit dem Stil `WarningTextBlockStyle` als 12px Arial rot und fett dargestellt.

Da implizite Stile einen impliziten `x:Key` , der mit ihrem `TargetType` , können Sie auch diese erben:

```
<!-- Implicit -->
<Style TargetType="TextBlock">
    <Setter Property="FontSize" Value="12"/>
    <Setter Property="Foreground" Value="#FFBBBBBB" />
    <Setter Property="FontFamily" Value="Arial" />
</Style>

<Style x:Key="WarningTextBlockStyle"
       TargetType="TextBlock"
       BasedOn="{StaticResource {x:Type TextBlock}}">
    <Setter Property="Foreground" Value="Red"/>
    <Setter Property="FontWeight" Value="Bold" />
</Style>
```

Styles in WPF online lesen: https://riptutorial.com/de/wpf/topic/4090/styles-in-wpf

Kapitel 16: System.Windows.Controls.WebBrowser

Einführung

Dadurch können Sie einen Webbrowser in Ihre WPF-Anwendung einfügen.

Bemerkungen

Ein wichtiger Punkt zu beachten, der aus der Dokumentation nicht ersichtlich ist, und Sie könnten jahrelang davon ausgehen, dass er sich standardmäßig wie InternetExplorer7 verhält, und nicht wie Ihre InternetExplorer-Installation (siehe https: // weblog). west-wind.com/posts/2011/may/21/web-browser-control-specifying-the-ie-version).

Dies kann nicht durch Festlegen einer Eigenschaft im Steuerelement behoben werden. Sie müssen entweder die angezeigten Seiten ändern, indem Sie ein HTML-Meta-Tag hinzufügen oder eine Registrierungseinstellung (!) anwenden. (Details zu beiden Ansätzen finden Sie unter dem Link oben.)

Beispielsweise kann dieses bizarre Designverhalten dazu führen, dass Sie eine Meldung erhalten, in der "Skriptfehler" / "Ein Fehler im Skript auf dieser Seite" angezeigt wird. Wenn Sie diesen Fehler durchlaufen, werden Sie möglicherweise der Meinung, dass die Lösung darin besteht, zu versuchen, den Fehler zu unterdrücken, anstatt das eigentliche Problem zu verstehen und die richtige Lösung anzuwenden.

Examples

Beispiel eines Webbrowsers innerhalb eines BusyIndicator

Beachten Sie, dass das WebBrowser-Steuerelement nicht mit Ihrer XAML-Definition übereinstimmt und sich selbst über andere Dinge stellt. Wenn Sie ihn beispielsweise in einen BusyIndicator einfügen, der als beschäftigt markiert ist, wird er immer noch über dem Steuerelement dargestellt. Die Lösung besteht darin, die Sichtbarkeit des WebBrowsers an den von BusyIndicator verwendeten Wert zu binden und den Boolean mit einem Konverter zu invertieren und in eine Sichtbarkeit umzuwandeln. Zum Beispiel:

```
    <telerik:RadBusyIndicator IsBusy="{Binding IsBusy}">
        <WebBrowser Visibility="{Binding IsBusy, Converter={StaticResource
InvertBooleanToVisibilityConverter}}"/>
    </telerik:RadBusyIndicator>
```

System.Windows.Controls.WebBrowser online lesen: https://riptutorial.com/de/wpf/topic/9115/system-windows-controls-webbrowser

Kapitel 17: Thread-Affinität Zugriff auf Benutzeroberflächenelemente

Examples

Zugriff auf ein Oberflächenelement innerhalb einer Aufgabe

Alle Elemente der Benutzeroberfläche, die erstellt wurden und sich im Haupt-Thread eines Programms befinden. Der Zugriff auf diese von einem anderen Thread aus ist in der .net Framework-Laufzeitumgebung verboten. Grundsätzlich liegt dies daran, dass alle Elemente der Benutzeroberfläche **Thread-empfindliche Ressourcen sind** und der Zugriff auf eine Ressource in einer Multithread-Umgebung Thread-sicher sein muss. Wenn dieser Cross-Thread-Objektzugriff zulässig ist, wird die Konsistenz in erster Linie beeinträchtigt.

Betrachten Sie dieses Szenario:

Wir haben eine Berechnung innerhalb einer Aufgabe. Aufgaben werden in einem anderen Thread als dem Hauptthread ausgeführt. Während der Berechnung müssen wir eine Fortschrittsleiste aktualisieren. Um dies zu tun:

```
//Prepare the action
Action taskAction = new Action( () => {
    int progress = 0;
    Action invokeAction = new Action( () => { progressBar.Value = progress; });
    while (progress <= 100) {
        progress = CalculateSomething();
        progressBar.Dispatcher.Invoke( invokeAction );
    }
} );

//After .net 4.5
Task.Run( taskAction );

//Before .net 4.5
Task.Factory.StartNew( taskAction ,
    CancellationToken.None,
    TaskCreationOptions.DenyChildAttach,
    TaskScheduler.Default);
```

Jedes Oberflächenelement verfügt über ein Dispatcher-Objekt, das von seinem `DispatcherObject` Vorfahren (innerhalb des `System.Windows.Threading` Namespace) stammt. Der Dispatcher führt den angegebenen Delegaten synchron mit der angegebenen Priorität in dem Thread aus, dem der Dispatcher zugeordnet ist. Da die Ausführung synchronisiert ist, sollte die aufrufende Task auf ihr Ergebnis warten. Dies gibt uns die Möglichkeit, `int progress` auch innerhalb eines Dispatcher-Delegierten einzusetzen.

Wir möchten ein UI-Element asynchron aktualisieren und dann die Änderungen der `invokeAction` Definition ändern:

```
//Prepare the action
Action taskAction = new Action( () => {
    int progress = 0;
    Action<int> invokeAction = new Action<int>( (i) => { progressBar.Value = i; } )
    while (progress <= 100) {
        progress = CalculateSomething();
        progressBar.Dispatcher.BeginInvoke(
            invokeAction,
            progress );
    }
} );

//After .net 4.5
Task.Run( taskAction );

//Before .net 4.5
Task.Factory.StartNew( taskAction ,
    CancellationToken.None,
    TaskCreationOptions.DenyChildAttach,
    TaskScheduler.Default);
```

Dieses Mal haben wir `int progress` gepackt und als Parameter für Delegierte verwendet.

Thread-Affinität Zugriff auf Benutzeroberflächenelemente online lesen:
https://riptutorial.com/de/wpf/topic/6128/thread-affinitat-zugriff-auf-benutzeroberflachenelemente

Kapitel 18: Unterstützung für Video-Streaming und Pixel-Array-Zuweisung zu einem Bildsteuerelement

Parameter

Parameter	Einzelheiten
PixelHeight (System.Int32)	Die Höhe des Bildes in Einheiten von Bildpixeln
PixelWidth (System.Int32)	Die Breite des Bildes in Einheiten von Bildpixeln
PixelFormat (System.Windows.Media.PixelFormat)	Die Breite des Bildes in Einheiten von Bildpixeln
Pixel	Alles, was IList <T> implementiert - einschließlich des C # -Byte-Arrays
DpiX	Gibt die horizontale Dpi an - optional
DpiY	Gibt die vertikale Dpi an - optional

Bemerkungen

- Stellen Sie sicher, dass Sie auf die *System.Windows.Interactivity-Assembly* verweisen, damit der XAML-Parser die *xmlns: i*- Deklaration erkennt.
- Beachten Sie, dass die Anweisung *xmlns: b* mit dem Namespace übereinstimmt, in dem sich die Verhaltensimplementierung befindet
- Das Beispiel setzt Kenntnisse in Bezug auf Bindungsausdrücke und XAML voraus.
- Dieses Verhalten unterstützt das Zuweisen von Pixeln zu einem Bild in Form eines *Bytearrays* - auch wenn der Typ der Abhängigkeitseigenschaft als *IList angegeben ist* . Dies funktioniert, da das C # *-Byte-* Array die *IList* implementiert Schnittstelle.
- Das Verhalten erzielt eine sehr hohe Leistung und kann für das Videostreaming verwendet werden
- *Weisen Sie* stattdessen nicht der *Source* Dependency Property-Bindung des Bilds der *Pixels* Dependency Property zu
- Die Eigenschaften Pixel, PixelWidth, PixelHeight und PixelFormat müssen für die Darstellung der Pixel zugewiesen werden
- Reihenfolge der Abhängigkeit Die Zuordnung von Immobilien spielt keine Rolle

Examples

Verhaltensimplementierung

```
using System;
using System.Collections.Generic;
using System.Runtime.InteropServices;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Interactivity;
using System.Windows.Media;
using System.Windows.Media.Imaging;

namespace MyBehaviorAssembly
{
    public class PixelSupportBehavior : Behavior<Image>
    {

    public static readonly DependencyProperty PixelsProperty = DependencyProperty.Register(
        "Pixels", typeof (IList<byte>), typeof (PixelSupportBehavior), new
PropertyMetadata(default(IList<byte>),OnPixelsChanged));

    private static void OnPixelsChanged(DependencyObject d, DependencyPropertyChangedEventArgs
e)
    {
        var b = (PixelSupportBehavior) d;
        var pixels = (IList<byte>) e.NewValue;

        b.RenderPixels(pixels);
    }

    public IList<byte> Pixels
    {
        get { return (IList<byte>) GetValue(PixelsProperty); }
        set { SetValue(PixelsProperty, value); }
    }

    public static readonly DependencyProperty PixelFormatProperty =
DependencyProperty.Register(
        "PixelFormat", typeof (PixelFormat), typeof (PixelSupportBehavior), new
PropertyMetadata(PixelFormats.Default,OnPixelFormatChanged));


    private static void OnPixelFormatChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
    {
        var b = (PixelSupportBehavior) d;
        var pixelFormat = (PixelFormat) e.NewValue;

        if(pixelFormat==PixelFormats.Default)
            return;

        b._pixelFormat = pixelFormat;

        b.InitializeBufferIfAttached();
    }

    public PixelFormat PixelFormat
```

```
    {
        get { return (PixelFormat) GetValue(PixelFormatProperty); }
        set { SetValue(PixelFormatProperty, value); }
    }

    public static readonly DependencyProperty PixelWidthProperty =
DependencyProperty.Register(
        "PixelWidth", typeof (int), typeof (PixelSupportBehavior), new
PropertyMetadata(default(int),OnPixelWidthChanged));

    private static void OnPixelWidthChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
    {
        var b = (PixelSupportBehavior)d;
        var value = (int)e.NewValue;

        if(value<=0)
            return;

        b._pixelWidth = value;

        b.InitializeBufferIfAttached();
    }

    public int PixelWidth
    {
        get { return (int) GetValue(PixelWidthProperty); }
        set { SetValue(PixelWidthProperty, value); }
    }


    public static readonly DependencyProperty PixelHeightProperty =
DependencyProperty.Register(
        "PixelHeight", typeof (int), typeof (PixelSupportBehavior), new
PropertyMetadata(default(int),OnPixelHeightChanged));

    private static void OnPixelHeightChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
    {
        var b = (PixelSupportBehavior)d;
        var value = (int)e.NewValue;

        if (value <= 0)
            return;

        b._pixelHeight = value;

        b.InitializeBufferIfAttached();
    }

    public int PixelHeight
    {
        get { return (int) GetValue(PixelHeightProperty); }
        set { SetValue(PixelHeightProperty, value); }
    }

    public static readonly DependencyProperty DpiXProperty = DependencyProperty.Register(
        "DpiX", typeof (int), typeof (PixelSupportBehavior), new
PropertyMetadata(96,OnDpiXChanged));

    private static void OnDpiXChanged(DependencyObject d, DependencyPropertyChangedEventArgs
```

```
e)
    {
        var b = (PixelSupportBehavior)d;
        var value = (int)e.NewValue;

        if (value <= 0)
            return;

        b._dpiX = value;

        b.InitializeBufferIfAttached();
    }

    public int DpiX
    {
        get { return (int) GetValue(DpiXProperty); }
        set { SetValue(DpiXProperty, value); }
    }

    public static readonly DependencyProperty DpiYProperty = DependencyProperty.Register(
        "DpiY", typeof (int), typeof (PixelSupportBehavior), new
PropertyMetadata(96,OnDpiYChanged));

    private static void OnDpiYChanged(DependencyObject d, DependencyPropertyChangedEventArgs
e)
    {
        var b = (PixelSupportBehavior)d;
        var value = (int)e.NewValue;

        if (value <= 0)
            return;

        b._dpiY = value;

        b.InitializeBufferIfAttached();
    }

    public int DpiY
    {
        get { return (int) GetValue(DpiYProperty); }
        set { SetValue(DpiYProperty, value); }
    }

    private IntPtr _backBuffer = IntPtr.Zero;
    private int _bytesPerPixel;
    private const int BitsPerByte = 8;
    private int _pixelWidth;
    private int _pixelHeight;
    private int _dpiX;
    private int _dpiY;
    private Int32Rect _imageRectangle;
    private readonly GCHandle _defaultGcHandle = new GCHandle();
    private PixelFormat _pixelFormat;

    private int _byteArraySize;
    private WriteableBitmap _bitMap;

    private bool _attached;

    protected override void OnAttached()
    {
```

```
        _attached = true;
        InitializeBufferIfAttached();
    }

    private void InitializeBufferIfAttached()
    {
        if(_attached==false)
            return;

        ReevaluateBitsPerPixel();

        RecomputeByteArraySize();

        ReinitializeImageSource();
    }

    private void ReevaluateBitsPerPixel()
    {
        var f = _pixelFormat;

        if (f == PixelFormats.Default)
        {
            _bytesPerPixel = 0;
            return;
        };

        _bytesPerPixel = f.BitsPerPixel/BitsPerByte;
    }

    private void ReinitializeImageSource()
    {
        var f = _pixelFormat;
        var h = _pixelHeight;
        var w = _pixelWidth;

        if (w<=0|| h<=0 || f== PixelFormats.Default)
            return;

        _imageRectangle = new Int32Rect(0, 0, w, h);
        _bitMap = new WriteableBitmap(w, h, _dpiX, _dpiY, f, null);
        _backBuffer = _bitMap.BackBuffer;
        AssociatedObject.Source = _bitMap;
    }

    private void RenderPixels(IList<byte> pixels)
    {
        if (pixels == null)
        {
            return;
        }

        var buffer = _backBuffer;
        if (buffer == IntPtr.Zero)
            return;

        var size = _byteArraySize;

        var gcHandle = _defaultGcHandle;
        var allocated = false;
        var bitMap = _bitMap;
        var rect = _imageRectangle;
```

```
        var w = _pixelWidth;
        var locked = false;
        try
        {
            gcHandle = GCHandle.Alloc(pixels, GCHandleType.Pinned);
            allocated = true;

            bitMap.Lock();
            locked = true;
            var ptr = gcHandle.AddrOfPinnedObject();
            _bitMap.WritePixels(rect, ptr, size,w);
        }
        finally
        {
            if(locked)
                bitMap.Unlock();

            if (allocated)
                gcHandle.Free();
        }
    }

    private void RecomputeByteArraySize()
    {
        var h = _pixelHeight;
        var w = _pixelWidth;
        var bpp = _bytesPerPixel;

        if (w<=0|| h<=0 || bpp<=0)
            return;

        _byteArraySize = (w * h * bpp);
    }

    public PixelSupportBehavior()
    {
        _pixelFormat = PixelFormats.Default;
    }
}
}
```

XAML-Nutzung

```
<UserControl x:Class="Example.View"
         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:b="clr-namespace:MyBehaviorAssembly;assembly=MyBehaviorAssembly"
         xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"
         mc:Ignorable="d"
         d:DesignHeight="200" d:DesignWidth="200"
         >

                <Image Stretch="Uniform">

                <i:Interaction.Behaviors>

                    <b:PixelSupportBehavior
```

```
                            PixelHeight="{Binding PixelHeight}"
                            PixelWidth="{Binding PixelWidth}"
                            PixelFormat="{Binding PixelFormat}"
                            Pixels="{Binding Pixels}"
                            />

                </i:Interaction.Behaviors>

            </Image>

</UserControl>
```

Unterstützung für Video-Streaming und Pixel-Array-Zuweisung zu einem Bildsteuerelement online lesen: https://riptutorial.com/de/wpf/topic/6435/unterstutzung-fur-video-streaming-und-pixel-array-zuweisung-zu-einem-bildsteuerelement

Kapitel 19: Wert- und Multivalue-Konverter

Parameter

Parameter	Einzelheiten
Wert	Der von der Bindungsquelle erzeugte Wert.
Werte	Das Werte-Array, das von der Bindungsquelle erzeugt wird.
targetType	Der Typ der Bindungszieleigenschaft.
Parameter	Der zu verwendende Konverter-Parameter.
Kultur	Die im Konverter zu verwendende Kultur.

Bemerkungen

Was sind IValueConverter und IMultiValueConvertert

IValueConverter und IMultiValueConverter - Schnittstellen, mit denen eine benutzerdefinierte Logik auf eine Bindung angewendet werden kann.

Wofür sind sie nützlich?

1. Sie haben einen Typwert, möchten aber Nullwerte auf eine Art und positive Zahlen auf andere Weise anzeigen
2. Sie haben einen Typwert und möchten ein Element in einem Fall anzeigen und in einem anderen ausblenden
3. Sie haben einen numerischen Geldwert, möchten ihn aber als Wörter anzeigen
4. Sie haben einen numerischen Wert, möchten jedoch unterschiedliche Bilder für andere Nummern anzeigen

Dies sind einige der einfachen Fälle, aber es gibt noch viele mehr.

Für diesen Fall können Sie einen Wertkonverter verwenden. Diese kleinen Klassen, die die IValueConverter-Schnittstelle oder IMultiValueConverter implementieren, fungieren als Zwischenhändler und übersetzen einen Wert zwischen Quelle und Ziel. In einer Situation, in der Sie einen Wert transformieren müssen, bevor er sein Ziel erreicht oder wieder zu seiner Quelle zurückkehrt, benötigen Sie wahrscheinlich einen Konverter.

Examples

Build-In BooleanToVisibilityConverter [IValueConverter]

Konverter zwischen Boolean und Sichtbarkeit. Holen Sie sich einen `bool` Wert für die Eingabe und gibt den `Visibility` .

HINWEIS: Dieser Konverter ist bereits im Namespace `System.Windows.Controls` .

```
public sealed class BooleanToVisibilityConverter : IValueConverter
{
    /// <summary>
    /// Convert bool or Nullable bool to Visibility
    /// </summary>
    /// <param name="value">bool or Nullable bool</param>
    /// <param name="targetType">Visibility</param>
    /// <param name="parameter">null</param>
    /// <param name="culture">null</param>
    /// <returns>Visible or Collapsed</returns>
    public object Convert(object value, Type targetType, object parameter, CultureInfo
culture)
    {
        bool bValue = false;
        if (value is bool)
        {
            bValue = (bool)value;
        }
        else if (value is Nullable<bool>)
        {
            Nullable<bool> tmp = (Nullable<bool>)value;
            bValue = tmp.HasValue ? tmp.Value : false;
        }
        return (bValue) ? Visibility.Visible : Visibility.Collapsed;
    }

    /// <summary>
    /// Convert Visibility to boolean
    /// </summary>
    /// <param name="value"></param>
    /// <param name="targetType"></param>
    /// <param name="parameter"></param>
    /// <param name="culture"></param>
    /// <returns></returns>
    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo
culture)
    {
        if (value is Visibility)
        {
            return (Visibility)value == Visibility.Visible;
        }
        else
        {
            return false;
        }
    }
}
```

Verwendung des Konverters

1. Ressource definieren

```
<BooleanToVisibilityConverter x:Key="BooleanToVisibilityConverter"/>
```

3. Verwenden Sie es verbindlich

```
<Button Visibility="{Binding AllowEditing,
                             Converter={StaticResource BooleanToVisibilityConverter}}"/>
```

Konverter mit Eigenschaft [IValueConverter]

Zeigen Sie, wie Sie einen einfachen Konverter mit einem Parameter über die Eigenschaft erstellen, und übergeben Sie ihn in der Deklaration. Konvertieren Sie den `bool` Wert in `Visibility` . Erlauben Sie den invertierten Ergebniswert, indem Sie die `Inverted` Eigenschaft auf `True` .

```
public class BooleanToVisibilityConverter : IValueConverter
{
    public bool Inverted { get; set; }

    /// <summary>
    /// Convert bool or Nullable bool to Visibility
    /// </summary>
    /// <param name="value">bool or Nullable bool</param>
    /// <param name="targetType">Visibility</param>
    /// <param name="parameter">null</param>
    /// <param name="culture">null</param>
    /// <returns>Visible or Collapsed</returns>
    public object Convert(object value, Type targetType, object parameter, CultureInfo
culture)
    {
        bool bValue = false;
        if (value is bool)
        {
            bValue = (bool)value;
        }
        else if (value is Nullable<bool>)
        {
            Nullable<bool> tmp = (Nullable<bool>)value;
            bValue = tmp ?? false;
        }

        if (Inverted)
            bValue = !bValue;
        return (bValue) ? Visibility.Visible : Visibility.Collapsed;
    }

    /// <summary>
    /// Convert Visibility to boolean
    /// </summary>
    /// <param name="value"></param>
    /// <param name="targetType"></param>
    /// <param name="parameter"></param>
    /// <param name="culture"></param>
    /// <returns>True or False</returns>
    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo
culture)
```

```
    {
        if (value is Visibility)
        {
            return ((Visibility) value == Visibility.Visible) && !Inverted;
        }

        return false;
    }
}
```

Verwendung des Konverters

1. Namensraum definieren

```
xmlns:converters="clr-namespace:MyProject.Converters;assembly=MyProject"
```

2. Ressource definieren

```
<converters:BooleanToVisibilityConverter x:Key="BoolToVisibilityInvertedConverter"
                                         Inverted="False"/>
```

3. Verwenden Sie es verbindlich

```
<Button Visibility="{Binding AllowEditing, Converter={StaticResource
BoolToVisibilityConverter}}"/>
```

Einfacher Add-Converter [IMultiValueConverter]

Zeigen Sie, wie Sie einen einfachen `IMultiValueConverter` Konverter erstellen und `MultiBinding` in xaml verwenden. Holen Sie sich die Summe aller Werte, die vom `values` Array übergeben werden.

```
public class AddConverter : IMultiValueConverter
{
    public object Convert(object[] values, Type targetType, object parameter, CultureInfo
culture)
    {
        decimal sum = 0M;

        foreach (string value in values)
        {
            decimal parseResult;
            if (decimal.TryParse(value, out parseResult))
            {
                sum += parseResult;
            }
        }

        return sum.ToString(culture);
    }

    public object[] ConvertBack(object value, Type[] targetTypes, object parameter,
CultureInfo culture)
    {
        throw new NotSupportedException();
```

```
    }
}
```

Verwendung des Konverters

1. Namensraum definieren

```
xmlns:converters="clr-namespace:MyProject.Converters;assembly=MyProject"
```

2. Ressource definieren

```
<converters:AddConverter x:Key="AddConverter"/>
```

3. Verwenden Sie es verbindlich

```
<StackPanel Orientation="Vertical">
    <TextBox x:Name="TextBox" />
    <TextBox x:Name="TextBox1" />
    <TextBlock >
        <TextBlock.Text>
            <MultiBinding Converter="{StaticResource AddConverter}">
                <Binding Path="Text" ElementName="TextBox"/>
                <Binding Path="Text" ElementName="TextBox1"/>
            </MultiBinding>
         </TextBlock.Text>
    </TextBlock>
</StackPanel>
```

Nutzungskonverter mit ConverterParameter

Zeigen Sie, wie Sie einen einfachen Konverter erstellen und mit `ConverterParameter` Parameter an den Konverter übergeben. Multiplizieren Sie den Wert mit dem in ConverterParameter übergebenen Koeffizienten.

```
public class MultiplyConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo
culture)
    {
        if (value == null)
            return 0;

        if (parameter == null)
            parameter = 1;

        double number;
        double coefficient;

        if (double.TryParse(value.ToString(), out number) &&
double.TryParse(parameter.ToString(), out coefficient))
        {
            return number * coefficient;
        }
```

```
        return 0;
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo
culture)
    {
        throw new NotSupportedException();
    }
}
```

Verwendung des Konverters

1. Namensraum definieren

```
xmlns:converters="clr-namespace:MyProject.Converters;assembly=MyProject"
```

2. Ressource definieren

```
<converters:MultiplyConverter x:Key="MultiplyConverter"/>
```

3. Verwenden Sie es verbindlich

```
<StackPanel Orientation="Vertical">
    <TextBox x:Name="TextBox" />
    <TextBlock Text="{Binding Path=Text,
                             ElementName=TextBox,
                             Converter={StaticResource MultiplyConverter},
                             ConverterParameter=10}"/>
</StackPanel>
```

Mehrere Konverter gruppieren [IValueConverter]

Dieser Konverter kettet mehrere Konverter zusammen.

```
public class ValueConverterGroup : List<IValueConverter>, IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo
culture)
    {
        return this.Aggregate(value, (current, converter) => converter.Convert(current,
targetType, parameter, culture));
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo
culture)
    {
        throw new NotSupportedException();
    }
}
```

In diesem Beispiel wird das boolesche Ergebnis von `EnumToBooleanConverter` als Eingabe in `BooleanToVisibilityConverter` .

```
<local:ValueConverterGroup x:Key="EnumToVisibilityConverter">
    <local:EnumToBooleanConverter/>
    <local:BooleanToVisibilityConverter/>
</local:ValueConverterGroup>
```

Die Schaltfläche ist nur sichtbar, wenn die `CurrentMode` Eigenschaft auf `Ready` .

```
<Button Content="Ok" Visibility="{Binding Path=CurrentMode, Converter={StaticResource
EnumToVisibilityConverter}, ConverterParameter={x:Static local:Mode.Ready}"/>
```

Verwendung von MarkupExtension mit Konvertern zum Überspringen der Deklaration der Ressource

Um den Konverter zu verwenden, müssen Sie ihn normalerweise wie folgt als Ressource definieren:

```
<converters:SomeConverter x:Key="SomeConverter"/>
```

Sie können diesen Schritt überspringen, indem Sie einen Konverter als `MarkupExtension` und die Methode `ProvideValue` . Im folgenden Beispiel wird ein Wert in einen negativen Wert konvertiert:

```
namespace MyProject.Converters
{
public class Converter_Negative : MarkupExtension, IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo
culture)
    {
        return this.ReturnNegative(value);
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo
culture)
    {
        return this.ReturnNegative(value);
    }

    private object ReturnNegative(object value)
    {
        object result = null;
        var @switch = new Dictionary<Type, Action> {
            { typeof(bool), () => result=!(bool)value },
            { typeof(byte), () => result=-1*(byte)value },
            { typeof(short), () => result=-1*(short)value },
            { typeof(int), () => result=-1*(int)value },
            { typeof(long), () => result=-1*(long)value },
            { typeof(float), () => result=-1f*(float)value },
            { typeof(double), () => result=-1d*(double)value },
            { typeof(decimal), () => result=-1m*(decimal)value }
        };

        @switch[value.GetType()]();
        if (result == null) throw new NotImplementedException();
        return result;
    }
```

```
    public Converter_Negative()
        : base()
    {
    }

    private static Converter_Negative _converter = null;

    public override object ProvideValue(IServiceProvider serviceProvider)
    {
        if (_converter == null) _converter = new Converter_Negative();
        return _converter;
    }
}
}
```

Verwendung des Konverters:

1. Namensraum definieren

   xmlns: converters = "clr-namespace: MyProject.Converters; assembly = MyProject"

2. Beispiel Verwendung dieses Konverters in der Bindung

```
<RichTextBox IsReadOnly="{Binding Path=IsChecked, ElementName=toggleIsEnabled,
Converter={converters:Converter_Negative}}"/>
```

Verwenden Sie IMultiValueConverter, um mehrere Parameter an einen Befehl zu übergeben

Es ist möglich, mehrere gebundene Werte als `CommandParameter` mit `MultiBinding` mit einem sehr einfachen `IMultiValueConverter` :

```
namespace MyProject.Converters
{
    public class Converter_MultipleCommandParameters : MarkupExtension, IMultiValueConverter
    {
        public object Convert(object[] values, Type targetType, object parameter, CultureInfo
culture)
        {
            return values.ToArray();
        }
        public object[] ConvertBack(object value, Type[] targetTypes, object parameter,
CultureInfo culture)
        {
            throw new NotSupportedException();
        }

        private static Converter_MultipleCommandParameters _converter = null;

        public override object ProvideValue(IServiceProvider serviceProvider)
        {
            if (_converter == null) _converter = new Converter_MultipleCommandParameters();
            return _converter;
        }
```

```
        public Converter_MultipleCommandParameters()
            : base()
        {
        }
    }
}
```

Verwendung des Konverters:

1. Beispielimplementierung - Methode, die aufgerufen wird, wenn `SomeCommand` ausgeführt wird (*Hinweis:* `DelegateCommand` *ist eine Implementierung von* `ICommand` *, die in diesem Beispiel nicht bereitgestellt wird*):

   ```
   private ICommand _SomeCommand;
   public ICommand SomeCommand
   {
       get { return _SomeCommand ?? (_SomeCommand = new DelegateCommand(a =>
   OnSomeCommand(a))); }
   }

   private void OnSomeCommand(object item)
   {
       object[] parameters = item as object[];

       MessageBox.Show(
           string.Format("Execute command: {0}\nParameter 1: {1}\nParamter 2: {2}\nParamter
   3: {3}",
           "SomeCommand", parameters[0], parameters[1], parameters[2]));
   }
   ```

2. Namensraum definieren

   xmlns: converters = "clr-namespace: MyProject.Converters; assembly = MyProject"

3. Beispiel Verwendung dieses Konverters in der Bindung

   ```
   <Button Width="150" Height="23" Content="Execute some command" Name="btnTestSomeCommand"
       Command="{Binding Path=SomeCommand}" >
       <Button.CommandParameter>
           <MultiBinding Converter="{converters:Converter_MultipleCommandParameters}">
               <Binding RelativeSource="{RelativeSource Self}" Path="IsFocused"/>
               <Binding RelativeSource="{RelativeSource Self}" Path="Name"/>
               <Binding RelativeSource="{RelativeSource Self}" Path="ActualWidth"/>
           </MultiBinding>
       </Button.CommandParameter>
   </Button>
   ```

Wert- und Multivalue-Konverter online lesen: https://riptutorial.com/de/wpf/topic/3950/wert--und-multivalue-konverter

Kapitel 20: WPF-Architektur

Examples

DispatcherObject

Kommt von

`Object`

Schlüsselmitglieder

```
public Dispatcher Dispatcher { get; }
```

Zusammenfassung

Die meisten Objekte in WPF werden von `DispatcherObject` , das die grundlegenden Konstrukte für den Umgang mit Parallelität und Threading bereitstellt. Solche Objekte sind einem Dispatcher zugeordnet.

Nur der Thread, in dem der Dispatcher erstellt wurde, kann direkt auf das DispatcherObject zugreifen. Um auf ein DispatcherObject von einem anderen Thread als dem Thread `BeginInvoke` für den das DispatcherObject erstellt wurde, ist ein Aufruf von `Invoke` oder `BeginInvoke` auf dem Dispatcher erforderlich, dem das Objekt zugeordnet ist.

Abhängigkeitsobjekt

Kommt von

`DispatcherObject`

Schlüsselmitglieder

```
public object GetValue(DependencyProperty dp);
public void   SetValue(DependencyProperty dp, object value);
```

Zusammenfassung

Von `DependencyObject` abgeleitete Klassen nehmen am Abhängigkeitseigenschaftssystem teil. Dazu gehören das Registrieren von Abhängigkeitseigenschaften sowie die Angabe von Informationen und Informationen zu diesen Eigenschaften. Da Abhängigkeitseigenschaften der Eckpfeiler der WPF-Entwicklung sind, leiten sich alle WPF-Steuerelemente letztlich von

`DependencyObject` .

WPF-Architektur online lesen: https://riptutorial.com/de/wpf/topic/3571/wpf-architektur

Kapitel 21: WPF-Lokalisierung

Bemerkungen

Der Inhalt von Steuerelementen kann mithilfe von Ressourcendateien lokalisiert werden, genau wie dies in Klassen möglich ist. Für XAML gibt es eine bestimmte Syntax, die sich zwischen einer C # - und einer VB-Anwendung unterscheidet.

Die Schritte sind:

- Für jedes WPF-Projekt: Machen Sie die Ressourcendatei öffentlich, der Standard ist "internal".
- Verwenden Sie für C # WPF-Projekte die im Beispiel bereitgestellte XAML
- Verwenden Sie für VB-WPF-Projekte die im Beispiel bereitgestellte XAML und ändern Sie die Eigenschaft Custom Tool in `PublicVbMyResourcesResXFileCodeGenerator` .
- So wählen Sie die Resources.resx-Datei in einem VB-WPF-Projekt aus:
    - Wählen Sie das Projekt im Solution Explorer aus
    - Wählen Sie "Alle Dateien anzeigen".
    - Erweitern Sie Mein Projekt

Examples

XAML für VB

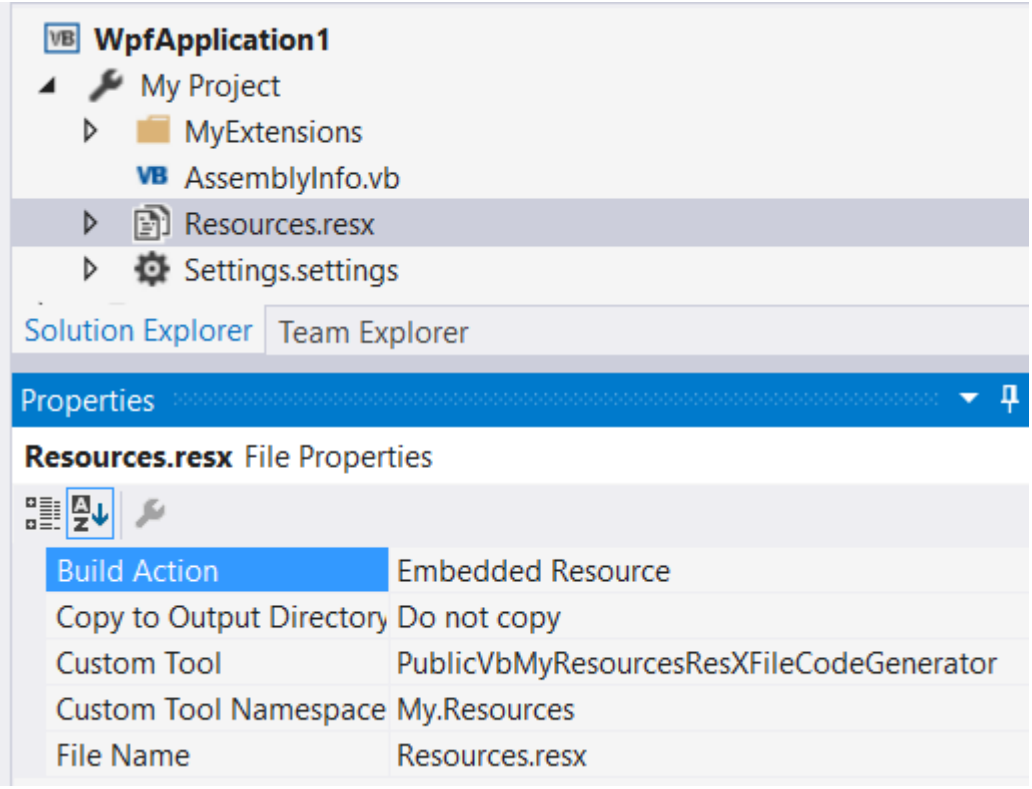
```
<Window x:Class="MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:local="clr-namespace:WpfApplication1"
    xmlns:my="clr-namespace:WpfApplication1.My.Resources"
    mc:Ignorable="d"
    Title="MainWindow" Height="350" Width="525">
<Grid>
    <StackPanel>
        <Label Content="{Binding Source={x:Static my:Resources.MainWindow_Label_Country}}" />
    </StackPanel>
</Grid>
```

Eigenschaften für die Ressourcendatei in VB

Standardmäßig ist die Eigenschaft des benutzerdefinierten Tools für eine VB-Ressourcendatei `VbMyResourcesResXFileCodeGenerator` . Mit diesem Codegenerator kann die Sicht (XAML) jedoch nicht auf die Ressourcen zugreifen. Um dieses Problem zu lösen, fügen Sie vor dem Eigenschaftenwert des benutzerdefinierten Tools `Public` hinzu.

So wählen Sie die Resources.resx-Datei in einem VB-WPF-Projekt aus:

- Wählen Sie das Projekt im Solution Explorer aus
- Wählen Sie "Alle Dateien anzeigen".
- Erweitern Sie "Mein Projekt"

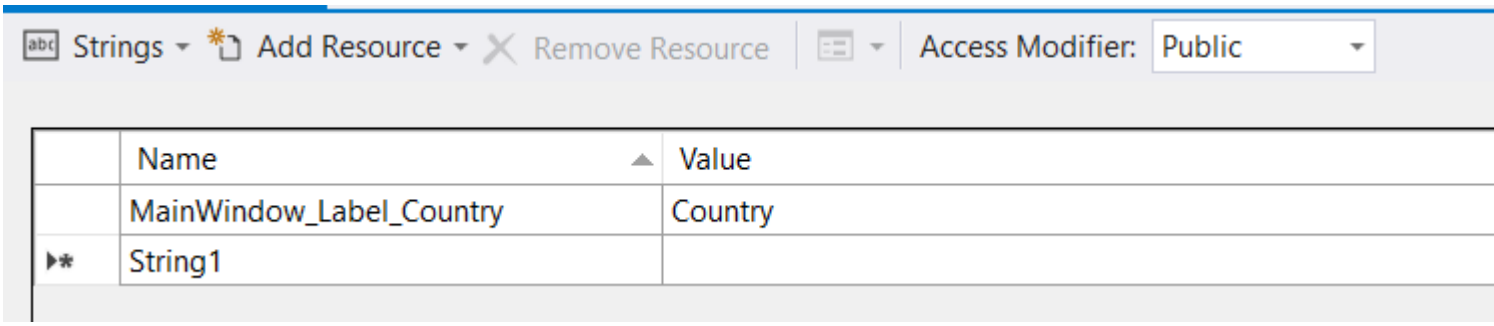


XAML für C

```
<Window x:Class="WpfApplication2.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:WpfApplication2"
    xmlns:resx="clr-namespace:WpfApplication2.Properties"
    mc:Ignorable="d"
    Title="MainWindow" Height="350" Width="525">
<Grid>
    <StackPanel>
        <Label Content="{Binding Source={x:Static resx:Resources.MainWindow_Label_Country}}"/>
    </StackPanel>
</Grid>
```

Machen Sie die Ressourcen öffentlich

Öffnen Sie die Ressourcendatei mit einem Doppelklick. Ändern Sie den Zugriffsmodifizierer in "Public".



WPF-Lokalisierung online lesen: https://riptutorial.com/de/wpf/topic/3905/wpf-lokalisierung

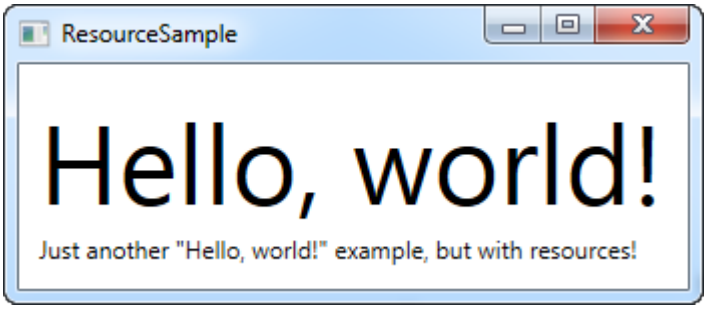
Kapitel 22: WPF-Ressourcen

Examples

Hallo Ressourcen

WPF führt ein sehr praktisches Konzept ein: Die Möglichkeit, Daten als Ressource zu speichern, entweder lokal für ein Steuerelement, lokal für das gesamte Fenster oder global für die gesamte Anwendung. Die Daten können nahezu beliebig sein, von tatsächlichen Informationen bis hin zu einer Hierarchie von WPF-Steuerelementen. Auf diese Weise können Sie Daten an einem Ort ablegen und dann von einem oder mehreren anderen Orten aus verwenden, was sehr nützlich ist. Das Konzept wird häufig für Stile und Vorlagen verwendet.

```
<Window x:Class="WPFApplication.ResourceSample"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:sys="clr-namespace:System;assembly=mscorlib"
        Title="ResourceSample" Height="150" Width="350">
    <Window.Resources>
        <sys:String x:Key="strHelloWorld">Hello, world!</sys:String>
    </Window.Resources>
    <StackPanel Margin="10">
        <TextBlock Text="{StaticResource strHelloWorld}" FontSize="56" />
        <TextBlock>Just another "<TextBlock Text="{StaticResource strHelloWorld}" />" example,
but with resources!</TextBlock>
    </StackPanel>
</Window>
```



Ressourcen erhalten einen Schlüssel mit dem Attribut x: Key, mit dem Sie von anderen Teilen der Anwendung mithilfe dieses Schlüssels in Kombination mit der Markierungserweiterung StaticResource darauf verweisen können. In diesem Beispiel speichere ich nur eine einfache Zeichenfolge, die ich dann von zwei verschiedenen TextBlock-Steuerelementen verwende.

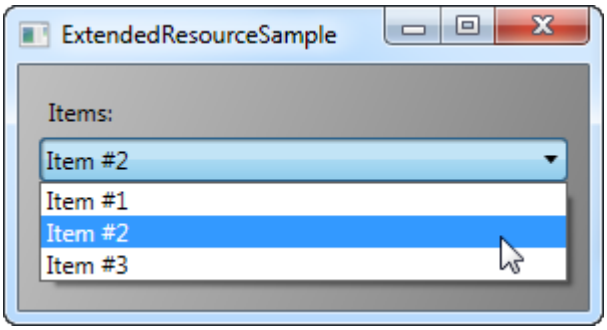
Ressourcentypen

Das Teilen einer einfachen Zeichenfolge war einfach, aber Sie können noch viel mehr tun. In diesem Beispiel speichere ich auch ein komplettes Array von Strings zusammen mit einem Farbverlaufspinsel für den Hintergrund. Dies sollte Ihnen eine ziemlich gute Vorstellung davon geben, wie viel Sie mit Ressourcen machen können:

```
<Window x:Class="WPFApplication.ExtendedResourceSample"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:sys="clr-namespace:System;assembly=mscorlib"
        Title="ExtendedResourceSample" Height="160" Width="300"
        Background="{DynamicResource WindowBackgroundBrush}">
    <Window.Resources>
        <sys:String x:Key="ComboBoxTitle">Items:</sys:String>

        <x:Array x:Key="ComboBoxItems" Type="sys:String">
            <sys:String>Item #1</sys:String>
            <sys:String>Item #2</sys:String>
            <sys:String>Item #3</sys:String>
        </x:Array>

        <LinearGradientBrush x:Key="WindowBackgroundBrush">
            <GradientStop Offset="0" Color="Silver"/>
            <GradientStop Offset="1" Color="Gray"/>
        </LinearGradientBrush>
    </Window.Resources>
    <StackPanel Margin="10">
        <Label Content="{StaticResource ComboBoxTitle}" />
        <ComboBox ItemsSource="{StaticResource ComboBoxItems}" />
    </StackPanel>
</Window>
```



Dieses Mal haben wir ein paar zusätzliche Ressourcen hinzugefügt, sodass unser Fenster jetzt eine einfache Zeichenfolge, ein Array von Zeichenfolgen und einen LinearGradientBrush enthält. Die Zeichenfolge wird für die Beschriftung verwendet, das String-Array wird als Elemente für das ComboBox-Steuerelement verwendet und der Verlaufspinsel wird als Hintergrund für das gesamte Fenster verwendet. Wie Sie sehen, kann so ziemlich alles als Ressource gespeichert werden.

Lokale und anwendungsweite Ressourcen

Wenn Sie eine bestimmte Ressource nur für ein bestimmtes Steuerelement benötigen, können Sie sie lokaler machen, indem Sie sie anstelle des Fensters zu diesem Steuerelement hinzufügen. Es funktioniert auf dieselbe Weise, der einzige Unterschied besteht darin, dass Sie jetzt nur innerhalb des Bereichs des Steuerelements darauf zugreifen können, wo Sie es ablegen:

```
<StackPanel Margin="10">
    <StackPanel.Resources>
        <sys:String x:Key="ComboBoxTitle">Items:</sys:String>
    </StackPanel.Resources>
    <Label Content="{StaticResource ComboBoxTitle}" />
</StackPanel>
```

In diesem Fall fügen wir die Ressource dem StackPanel hinzu und verwenden sie dann von ihrem untergeordneten Steuerelement Label (Label). Andere Steuerelemente im StackPanel hätten es ebenfalls verwenden können, genauso wie Kinder dieser untergeordneten Steuerelemente auf das Steuerelement zugreifen konnten. Steuerelemente außerhalb dieses StackPanel-Objekts haben jedoch keinen Zugriff darauf.

Wenn Sie die Möglichkeit benötigen, von mehreren Fenstern auf die Ressource zuzugreifen, ist dies ebenfalls möglich. Die App.xaml-Datei kann Ressourcen wie das Fenster und jede Art von WPF-Steuerelement enthalten. Wenn Sie sie in App.xaml speichern, können Sie in allen Fenstern und Benutzersteuerelementen des Projekts global darauf zugreifen. Es funktioniert genauso wie beim Speichern und Verwenden in einem Fenster:

```
<Application x:Class="WpfSamples.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:sys="clr-namespace:System;assembly=mscorlib"
             StartupUri="WPFApplication/ExtendedResourceSample.xaml">
    <Application.Resources>
        <sys:String x:Key="ComboBoxTitle">Items:</sys:String>
    </Application.Resources>
</Application>
```

Es ist auch das Gleiche - WPF wechselt automatisch vom lokalen Steuerelement zum Fenster und dann zu App.xaml, um eine bestimmte Ressource zu finden:

```
<Label Content="{StaticResource ComboBoxTitle}" />
```

Ressourcen aus Code-Behind

In diesem Beispiel greifen wir von Code-behind aus auf drei verschiedene Ressourcen zu, die jeweils in einem anderen Bereich gespeichert sind

App.xaml:

```
<Application x:Class="WpfSamples.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:sys="clr-namespace:System;assembly=mscorlib"
             StartupUri="WPFApplication/ResourcesFromCodeBehindSample.xaml">
    <Application.Resources>
        <sys:String x:Key="strApp">Hello, Application world!</sys:String>
    </Application.Resources>
</Application>
```

Fenster:

```
<Window x:Class="WpfSamples.WPFApplication.ResourcesFromCodeBehindSample"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:sys="clr-namespace:System;assembly=mscorlib"
        Title="ResourcesFromCodeBehindSample" Height="175" Width="250">
    <Window.Resources>
```

```
        <sys:String x:Key="strWindow">Hello, Window world!</sys:String>
    </Window.Resources>
    <DockPanel Margin="10" Name="pnlMain">
        <DockPanel.Resources>
            <sys:String x:Key="strPanel">Hello, Panel world!</sys:String>
        </DockPanel.Resources>

        <WrapPanel DockPanel.Dock="Top" HorizontalAlignment="Center" Margin="10">
            <Button Name="btnClickMe" Click="btnClickMe_Click">Click me!</Button>
        </WrapPanel>

        <ListBox Name="lbResult" />
    </DockPanel>
</Window>
```

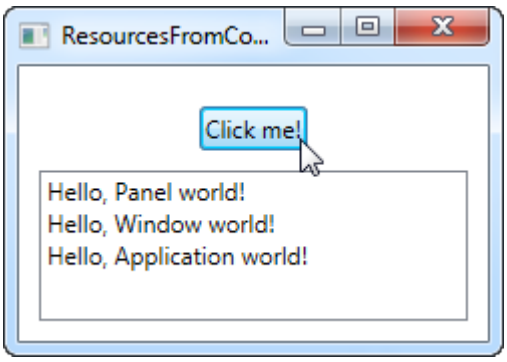
Code-Behind:

```
using System;
using System.Windows;

namespace WpfSamples.WPFApplication
{
        public partial class ResourcesFromCodeBehindSample : Window
        {
                public ResourcesFromCodeBehindSample()
                {
                        InitializeComponent();
                }

                private void btnClickMe_Click(object sender, RoutedEventArgs e)
                {
                        lbResult.Items.Add(pnlMain.FindResource("strPanel").ToString());
                        lbResult.Items.Add(this.FindResource("strWindow").ToString());

lbResult.Items.Add(Application.Current.FindResource("strApp").ToString());
                }
        }
}
```



Wie Sie sehen, speichern wir drei verschiedene "Hallo, Welt!" messages: Eine in App.xaml, eine im Fenster und eine lokal für das Hauptfenster. Die Schnittstelle besteht aus einer Schaltfläche und einer ListBox.

In Code-Behind behandeln wir das Klickereignis der Schaltfläche, in dem wir die einzelnen Textzeichenfolgen zur ListBox hinzufügen, wie auf dem Screenshot zu sehen ist. Wir verwenden die FindResource () -Methode, die die Ressource als Objekt zurückgibt (sofern gefunden), und

dann in die Zeichenfolge verwandeln, von der wir wissen, dass sie die ToString () -Methode ist.

Beachten Sie, wie wir die FindResource () - Methode in verschiedenen Bereichen verwenden - zuerst im Panel, dann im Fenster und dann im aktuellen Anwendungsobjekt. Es ist sinnvoll, nach der Ressource zu suchen, an der wir sie kennen. Wenn jedoch keine Ressource gefunden wird, wird die Suche in der Hierarchie fortgesetzt. Grundsätzlich hätten wir die FindResource () - Methode für das Panel in verwenden können alle drei Fälle, da es bis zum Fenster und später bis zur Anwendungsebene weitergegangen wäre, wenn es nicht gefunden wurde.

Das Gleiche gilt nicht umgekehrt: Die Suche navigiert nicht im Baum nach unten. Sie können also nicht auf der Anwendungsebene nach einer Ressource suchen, wenn diese lokal für das Steuerelement oder für das Fenster definiert wurde.

WPF-Ressourcen online lesen: https://riptutorial.com/de/wpf/topic/4371/wpf-ressourcen

Kapitel 23: WPF-Verhalten

Einführung

WPF-Verhalten ermöglicht es einem Entwickler, die Art und Weise zu ändern, wie WPF-Steuerelemente auf System- und Benutzerereignisse reagieren. Verhalten erben von der `Behavior` Klasse des `System.Windows.Interactity` Namespace. Dieser Namespace ist Teil des übergeordneten Expression Blend SDK, eine leichtere Version, die für Verhaltensbibliotheken geeignet ist, ist als [Nuget-Paket] [1] verfügbar. [1]: https://www.nuget.org/packages/System.Windows.Interactivity.WPF/

Examples

Einfaches Verhalten zum Abfangen von Mausradereignissen

Verhalten implementieren

Dieses Verhalten führt dazu, dass Mausrad-Ereignisse von einem inneren `ScrollViewer` zum übergeordneten `ScrollViewer` wenn sich der innere an der oberen oder unteren Grenze befindet. Ohne dieses Verhalten werden die Ereignisse den inneren `ScrollViewer` niemals `ScrollViewer` .

```
public class BubbleMouseWheelEvents : Behavior<UIElement>
{
    protected override void OnAttached()
    {
        base.OnAttached();
        this.AssociatedObject.PreviewMouseWheel += PreviewMouseWheel;
    }

    protected override void OnDetaching()
    {
        this.AssociatedObject.PreviewMouseWheel -= PreviewMouseWheel;
        base.OnDetaching();
    }

    private void PreviewMouseWheel(object sender, MouseWheelEventArgs e)
    {
        var scrollViewer = AssociatedObject.GetChildOf<ScrollViewer>(includeSelf: true);
        var scrollPos = scrollViewer.ContentVerticalOffset;
        if ((scrollPos == scrollViewer.ScrollableHeight && e.Delta < 0) || (scrollPos == 0 &&
e.Delta > 0))
        {
            UIElement rerouteTo = AssociatedObject;
            if (ReferenceEquals(scrollViewer, AssociatedObject))
            {
                rerouteTo = (UIElement) VisualTreeHelper.GetParent(AssociatedObject);
            }

            e.Handled = true;
            var e2 = new MouseWheelEventArgs(e.MouseDevice, e.Timestamp, e.Delta);
            e2.RoutedEvent = UIElement.MouseWheelEvent;
            rerouteTo.RaiseEvent(e2);
```

```
        }
    }
}
```

Behaviours subclass die `Behavior<T>` `UIElement` , wobei `T` der Typ des Steuerelements ist, mit dem es verbunden werden kann, in diesem Fall `UIElement` . Wenn das `Behavior` aus XAML instanziiert wird, wird die `OnAttached` Methode aufgerufen. Diese Methode ermöglicht es dem Verhalten, sich über Ereignisse (über `AssociatedControl`) an Ereignisse `AssociatedControl` . Eine ähnliche Methode, `OnDetached` wird aufgerufen, wenn das Verhalten von dem zugeordneten Element getrennt werden muss. Eventuelle Handler sollten entfernt oder auf andere Weise Objekte entfernt werden, um Speicherlecks zu vermeiden.

Dieses Verhalten hängt mit dem `PreviewMouseWheel` Ereignis zusammen. Dadurch kann das Ereignis abgefangen werden, bevor der `ScrollViewer` die `ScrollViewer` hat, es zu sehen. Er prüft die Position, um zu sehen, ob das Ereignis in der visuellen `ScrollViewer` höhere Hierarchie von `ScrollViewer` werden muss. Wenn ja, wird `e.Handled` auf `true` , um die Standardaktion des Ereignisses zu verhindern. Anschließend wird ein neues `MouseWheelEvent` an `AssociatedObject` . Andernfalls wird das Ereignis normal weitergeleitet.

Anhängen des Verhaltens an ein Element in XAML

Zunächst muss der XML-Namespace für die `interactivity` in den Gültigkeitsbereich aufgenommen werden, bevor er in XAML verwendet werden kann. Fügen Sie den Namespaces Ihrer XAML die folgende Zeile hinzu.

xmlns: interactivity = " http://schemas.microsoft.com/expression/2010/interaktivität "

Das Verhalten kann wie folgt angehängt werden:

```
<ScrollViewer>
    <!--...Content...-->
    <ScrollViewer>
        <interactivity:Interaction.Behaviors>
            <behaviors:BubbleMouseWheelEvents />
        </interactivity:Interaction.Behaviors>
        <!--...Content...-->
    </ScrollViewer>
    <!--...Content...-->.
</ScrollViewer>
```

Dadurch wird eine `Behaviors` als `ScrollViewer` Eigenschaft im inneren `ScrollViewer` , die ein `BubbleMouseWheelEvents` Verhalten enthält.

Dieses Verhalten kann auch an ein vorhandenes Steuerelement angehängt werden, das einen eingebetteten `ScrollViewer` enthält, z. B. eine `GridView` , und es funktioniert weiterhin ordnungsgemäß.

WPF-Verhalten online lesen: https://riptutorial.com/de/wpf/topic/8365/wpf-verhalten

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit wpf	Community, Derpcode, Gusdor, Matthew Cargille, Nasreddine, Sam, Stephen Wilson
2	"Half the Whitespace" Konstruktionsprinzip	Richardissimo
3	Abhängigkeitseigenschaften	Adi Lester, auticus, Clemens, Guttsy
4	Benutzerdefinierte UserControls mit Datenbindung erstellen	Itiveron, Mage Xy
5	Eine Einführung in WPF-Styles	J R
6	Einführung in die WPF-Datenbindung	Adi Lester, Arie, Gabor Barat, Guttsy, Ian Wold, Jirajha, vkluge, wkl
7	Löst aus	John Strit, Maxim
8	Markup-Erweiterungen	Alexander Pacha, Emad
9	MVVM in WPF	Andrew Stephens, Athafoud, Dutts, Felix D., Felix Too, H.B., James LaPenn, Kcvin, kowsky, Matt Klein, RamenChef, STiLeTT, TrBBoI, vkluge
10	Optimierung für die Touch-Interaktion	Martin Zikmund
11	Rastersteuerung	Alexander Mandt, vkluge
12	Slider-Bindung: Aktualisierung nur bei gezogenem Ziehen	Eyal Perry
13	Sprachsynthese	BKO
14	Startbild in WPF erstellen	Grx70, Sam
15	Styles in WPF	Guttsy, Jakub Lokša
16	System.Windows.Controls.WebBrowser	Richardissimo
17	Thread-Affinität Zugriff auf Benutzeroberflächenelemente	Mert Gülsoy

18	Unterstützung für Video-Streaming und Pixel-Array-Zuweisung zu einem Bildsteuerelement	Eyal Perry
19	Wert- und Multivalue-Konverter	Adi Lester, Arie, Dalstroem, galakt, ltiveron
20	WPF-Architektur	Adi Lester
21	WPF-Lokalisierung	Dabblernl
22	WPF-Ressourcen	Elangovan, John Strit, SUB-HDR
23	WPF-Verhalten	Bradley Uffner