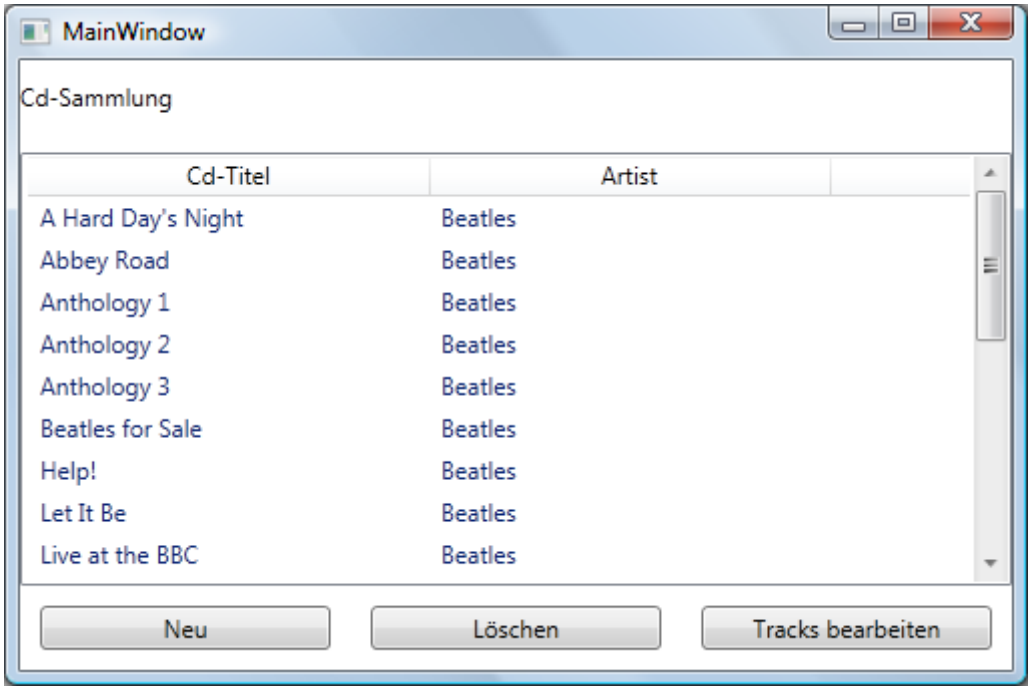
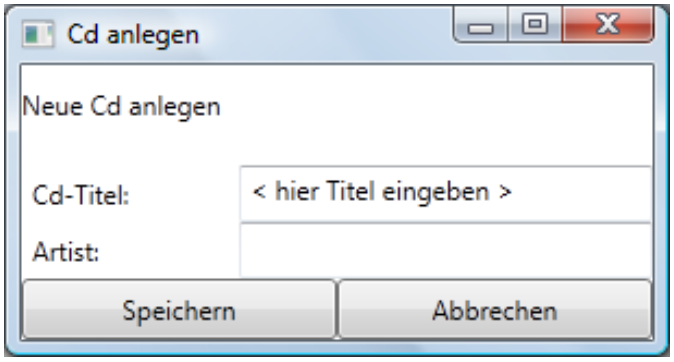
WPF-Tutorial 2: CdManager

Teil 2: WPF-Styles / Animation / ControlTemplates

Die Ausgangslage aus Tutorial 1 sieht derzeit folgendermaßen aus:





Styles

Da die Überschrift auf jeder Seite gleich aussehen soll, legen wir einen Style in der Datei App.xaml für Überschriften an:

```
<Application x:Class="CdManager.Wpf.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             StartupUri="MainWindow.xaml">
    <Application.Resources>
        <Style x:Key="FormCaptionStyle">
            <Setter Property="Control.FontSize" Value="20"></Setter>
            <Setter Property="Control.FontWeight" Value="Bold"></Setter>
            <Setter Property="Control.Foreground" Value="Sienna"></Setter>
            <Setter Property="Control.Margin" Value="5,10,0,20"></Setter>
        </Style>
    </Application.Resources>
</Application>
```

Für den Style haben wir den Key „FormCaptionStyle“ zugewiesen und Schriftgröße 20, Schriftart fett und Farbe „Sienna“ zugeordnet.

Damit die Überschriften am Formular nun dieses Style nutzen, müssen wir den entsprechenden TextBlock Komponenten in den einzelnen Formularen den Style zuordnen:

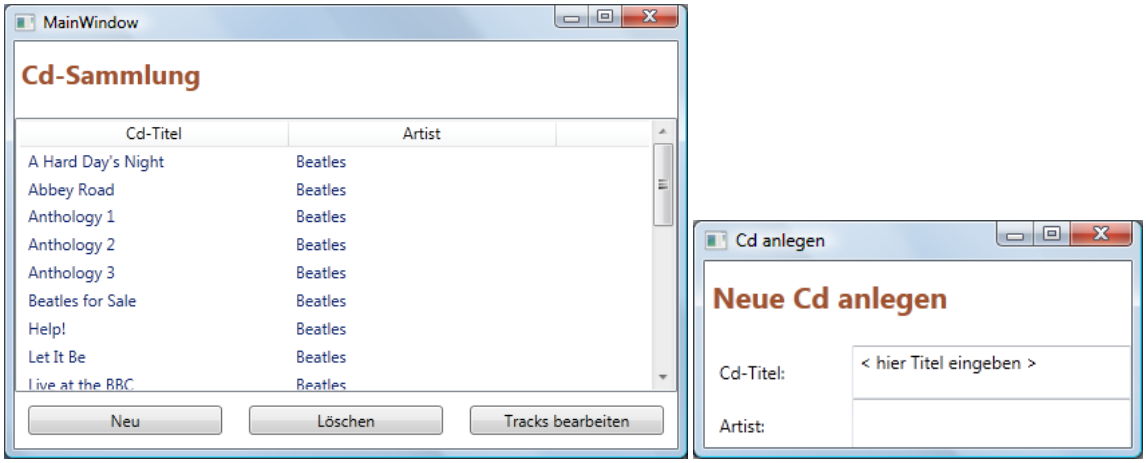
Beachte! Da der Style nun bereits ein Margin Attribut beinhaltet, entfernen wir das Margin Attribut aus den TextBlock Komponente denen wir den Style zuordnen!

```
<Window x:Class="CdManager.Wpf.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="340" Width="511" MinHeight="180" MinWidth="392"
        Name="wdMain">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"></RowDefinition>
            <RowDefinition></RowDefinition>
            <RowDefinition Height="Auto"></RowDefinition>
        </Grid.RowDefinitions>
        <TextBlock Grid.Row="0" Style="{StaticResource ResourceKey=FormCaptionStyle}">
            Cd-Sammlung
        </TextBlock>
```

Auch der Überschrift im **AddCdWindow** ordnen wir diesen Style zu!

```
<Window x:Class="CdManager.Wpf.AddCdWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Cd anlegen" Height="162" Width="306">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <TextBlock Grid.Row="0" Style="{StaticResource ResourceKey=FormCaptionStyle}">
            Neue Cd anlegen
        </TextBlock>
```

Nun sehen unsere Fenster bereits etwas anders aus! Vor allem können wir durch Styles das Aussehen für alle Formulare an zentraler Stelle ändern.



Als nächstes möchten wir, dass alle TextBox Komponenten (Eingabefelder) einen kleinen Abstand zu deren Nachbarn in alle Richtungen haben. Also erstellen wir einen Style für den TargetTyp TextBox in dem wir einen Margin von jeweils 3 festlegen.

```
<Style TargetType="TextBox">
    <Setter Property="Margin" Value="3,3,3,3"></Setter>
</Style>
```

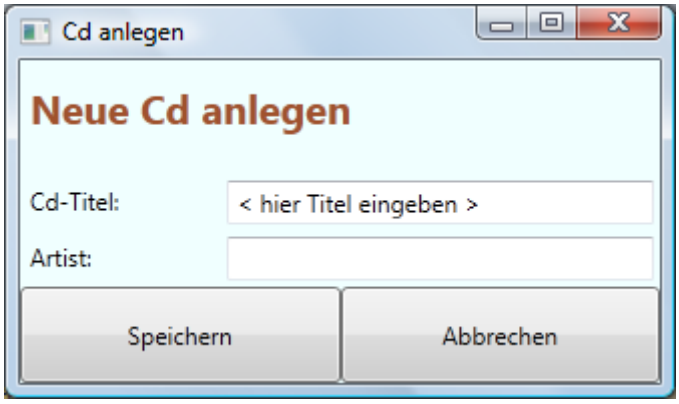
Da bei diesem Style kein Key angegeben wurde, muss dieser Style nicht extra jeder TextBox Komponente zugewiesen werden. Er gilt automatisch für alle TextBox Komponenten!

Als Hintergrundsfarbe für unsere Fenster möchten wir gerne „Azure“ verwenden. Dazu erstellen wir im App.xaml einen Style mit dem Key „WindowStyle“ und setzen zusätzlich den TargetType „Window“. Dann definieren wir die gewünschte Hintergrundsfarbe:

```
<Style x:Key="WindowStyle" TargetType="Window">
    <Setter Property="Background" Value="Azure"></Setter>
</Style>
```

Nicht vergessen den Window Komponenten diesen Style zuzuweisen! (Nur die Angabe von TargetType ohne Key, wie bei der obigen TextBox, funktioniert in diesem Fall nicht, da unsere Fenster nur abgeleitete Klassen von Window sind.)

```
<Window x:Class="CdManager.Wpf.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="340" Width="511" MinHeight="180" MinWidth="392"
        Style="{StaticResource ResourceKey=WindowStyle}"
        Name="wdMain">
    <Grid>
        <Grid.RowDefinitions>
```



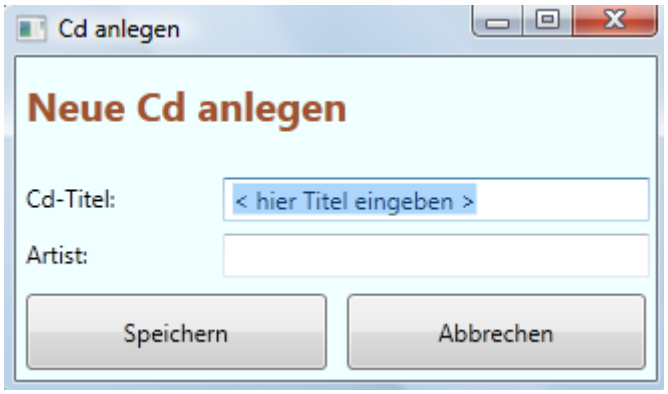
Noch für die Buttons einen Randabstand und minimale Höhe festlegen. Anschließend die im MainWindow beim Button eingetragenen Margins entfernen….

```
        <Style TargetType="Button">
            <Setter Property="Margin" Value="5,5,5,5"></Setter>
            <Setter Property="MinHeight" Value="30"></Setter>
        </Style>
```

Anhand des Hintergrunds der ListView Komponente soll ein Farbverlauf demonstriert werden dazu gibt es den LinearGradientBrush:

```
<Style TargetType="ListView">
    <Setter Property="Background">
        <Setter.Value>
            <LinearGradientBrush EndPoint="1,1" StartPoint="0,0">
                <GradientStop Color="#FFFFE07E" Offset="0"/>
                <GradientStop Color="#FFFFFAEA" Offset="1"/>
            </LinearGradientBrush>
        </Setter.Value>
    </Setter>
</Style>
```

Das Resultat (über Geschmack lässt sich natürlich streiten….):



Style-Triggers /Event-Trigger (Animation)

In WPF können relative einfach animierte Veränderungen an Komponenten vorgenommen werden. Z.B. kann die Größe einer Komponente in feinen Stufen verändert werden, wenn diese angeklickt wird, oder den Focus erhält.

Beispiel: Wir wollen eine TextBox-Komponenten „animiert“ vergrößern, sobald diese den Focus erhält und verkleinern wenn sie den Fokus wieder verliert.

Dazu fügen wir im App.xaml bei unserem TextBox-Style folgenden Style-Trigger ein:

```
<Style TargetType="TextBox">
    <Setter Property="Margin" Value="3,3,3,3"></Setter>
    <Style.Triggers>
        <EventTrigger RoutedEvent="GotFocus">
            <BeginStoryboard>
                <Storyboard>
                    <DoubleAnimation To="20" Duration="0:0:0.6"
                                     Storyboard.TargetProperty="FontSize"/>
                </Storyboard>
            </BeginStoryboard>
        </EventTrigger>
        <EventTrigger RoutedEvent="LostFocus">
            <BeginStoryboard>
                <Storyboard>
                    <DoubleAnimation To="12" Duration="0:0:0.6"
                                     Storyboard.TargetProperty="FontSize"/>
                </Storyboard>
            </BeginStoryboard>
        </EventTrigger>
    </Style.Triggers>
</Style>
```

Die jeweils angeklickte Textbox vergrößert sich nun schrittweise bis zur gewünschten Größe (20).

Dies funktioniert z.B. genauso mit Image Komponenten. Dadurch kann ein Bild auf einfachste Weise animiert vergrößert werden, wenn man mit dem Mauscursor darüberfährt. Somit können schnell beeindruckende Effekte erzielt werden!

Control Template:

Möchte man einen Button „von Grund auf“ anders gestalten. D.h. Rahmen definieren, festlegen wo der Content dargestellt werden soll u.ä., so bietet sich ein Control Template an. Beispiel für unsere Buttons (inkl. Triggers → siehe PowerPoint Präsentation zur Erläuterung):

```
<ControlTemplate x:Key="buttonTemplate" TargetType="{x:Type Button}">
    <Border x:Name="border" BorderBrush="Blue" BorderThickness="3" CornerRadius="3"
            Background="Azure" Padding="3" Margin="3">
        <TextBlock Margin="{TemplateBinding Padding}" Text="{TemplateBinding Content}"
                   VerticalAlignment="Center" HorizontalAlignment="Center"></TextBlock>
    </Border>
    <ControlTemplate.Triggers>
        <Trigger Property="IsMouseOver" Value="true">
            <Setter TargetName="border" Property="Background" Value="LightBlue"></Setter>
            <Setter TargetName="border" Property="Cursor" Value="Hand"></Setter>
        </Trigger>
    </ControlTemplate.Triggers>
</ControlTemplate>
```

In den jeweiligen Fenstern muss dann bei den Buttons das ControlTemplate über den Ressource Key ausgewählt werden:

Z.B. AddCdWindow:

```
<UniformGrid Rows="1" Grid.Row="2">
    <Button Template="{StaticResource ResourceKey=buttonTemplate}" Name="btSave">Speichern</Button>
    <Button Template="{StaticResource ResourceKey=buttonTemplate}" Name="btCancel">Abbrechen</Button>
</UniformGrid>
```