WPF
Behaviors

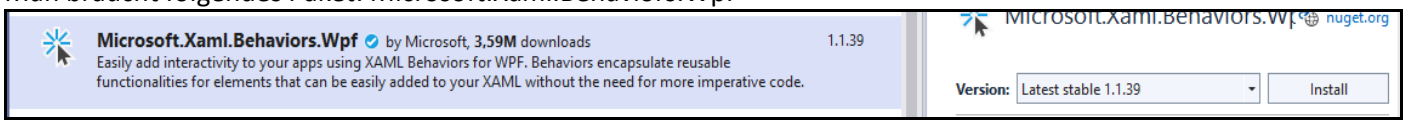
Inhalt

1 Behaviors

Wenn man eine zusätzliche DLL installiert, hat man eine sehr elegante Möglichkeit, beliebige Events auf beliebige Methoden des ViewModels zu binden.

1.1 Paket

Man braucht folgendes Paket: Microsoft.Xaml.Behaviors.Wpf



Oder: `dotnet add package Microsoft.Xaml.Behaviors.Wpf`

1.2 Ohne Parameter

1.2.1 XAML

Im XAML muss man dann ein Präfix für diese Assembly vergeben. Welche Namen man diesem Präfix gibt, ist wieder dem Programmierer überlassen, üblich ist aber **i**:

```
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:i="http://schemas.microsoft.com/xaml/behaviors"
xmlns:local="clr-namespace:RecordManagerWpf.Windows"
xmlns:viewmodels="clr-namespace:RecordManagerWpf.ViewModels"
```

Also: `xmlns:i="http://schemas.microsoft.com/xaml/behaviors"`

Jetzt kann man Events auf das ViewModel binden. Als Beispiel soll für ein Label das **MouseEnter**-Event auf die Methode **SomeDummyMethod** gebunden werden:

```
<Label Content="Fire MouseOver..." Margin="10,64,0,0" Height="50" Width="84" >
  <i:Interaction.Triggers>
    <i:EventTrigger EventName="MouseEnter" >
      <i:CallMethodAction
       TargetObject="{Binding}"
       MethodName="SomeDummyMethod"/>
    </i:EventTrigger>
  </i:Interaction.Triggers>
</Label>
```

1.2.2 ViewModel

Die Methode selbst ist eine ganz normale public Methode:

```
public void SomeDummyMethod()
{
  Console.WriteLine("SomeDummyMethod");
}
```

1.3 Mit Parameter

Um Parameter übergeben zu können, muss man den Trigger auf ein **ICommand** umleiten. Das geht über **InvokeCommandAction**.

1.3.1 XAML

Im XAML notiert man dann Folgendes:

```
<Label Content="Label with Param" HorizontalAlig
  <i:Interaction.Triggers>
    <i:EventTrigger EventName="MouseEnter">
      <i:InvokeCommandAction
        Command="{Binding MouseEnterCommand}"
        CommandParameter="quaxi" />
    </i:EventTrigger>
  </i:Interaction.Triggers>
</Label>
```

Der CommandParameter wird dabei (wie der Name vermuten lässt) dem Command als Parameter übergeben.

1.3.2 ViewModel

Man definiert im ViewModel ein RelayCommand wie gewohnt - der Code im ViewModel sieht dann so aus:

```
public ICommand MouseEnterCommand => new RelayCommand<string>(s =>
{
  Console.WriteLine($"MouseEnterCommand {s}");
});
```

1.4 Mit Parameter des Elements

Man gibt den Pfad auf die gewünschte Property im Binding unter **Path** an, und übergibt den Wert dann als CommandParameter.

1.4.1 XAML

```
<Label Name="lblInteract" Content="Label with Element Param" HorizontalAlignme
  <i:Interaction.Triggers>
    <i:EventTrigger EventName="MouseEnter">
      <i:InvokeCommandAction
        Command="{Binding MouseEnterCommand}"
        CommandParameter="{Binding ElementName=lblInteract, Path=Content}" />
    </i:EventTrigger>
  </i:Interaction.Triggers>
</Label>
```

1.4.2 Anderer Typ

Man kann beliebige Properties an die Methode übergeben, z.B. die Breite des Elements als double:

```
public ICommand MouseEnterCommandPara => new RelayCommand<double>(val =>
{
  Console.WriteLine($"MouseEnterCommandPara {val}");
});
```

Als Path wird dann eben Width angegeben:

```
<i:InvokeCommandAction
  Command="{Binding MouseEnterCommandPara}"
  CommandParameter="{Binding ElementName=lblInteract, Path=Width}" />
```