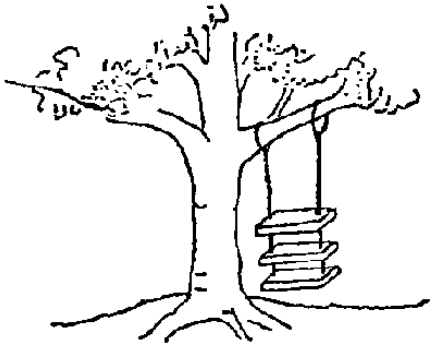
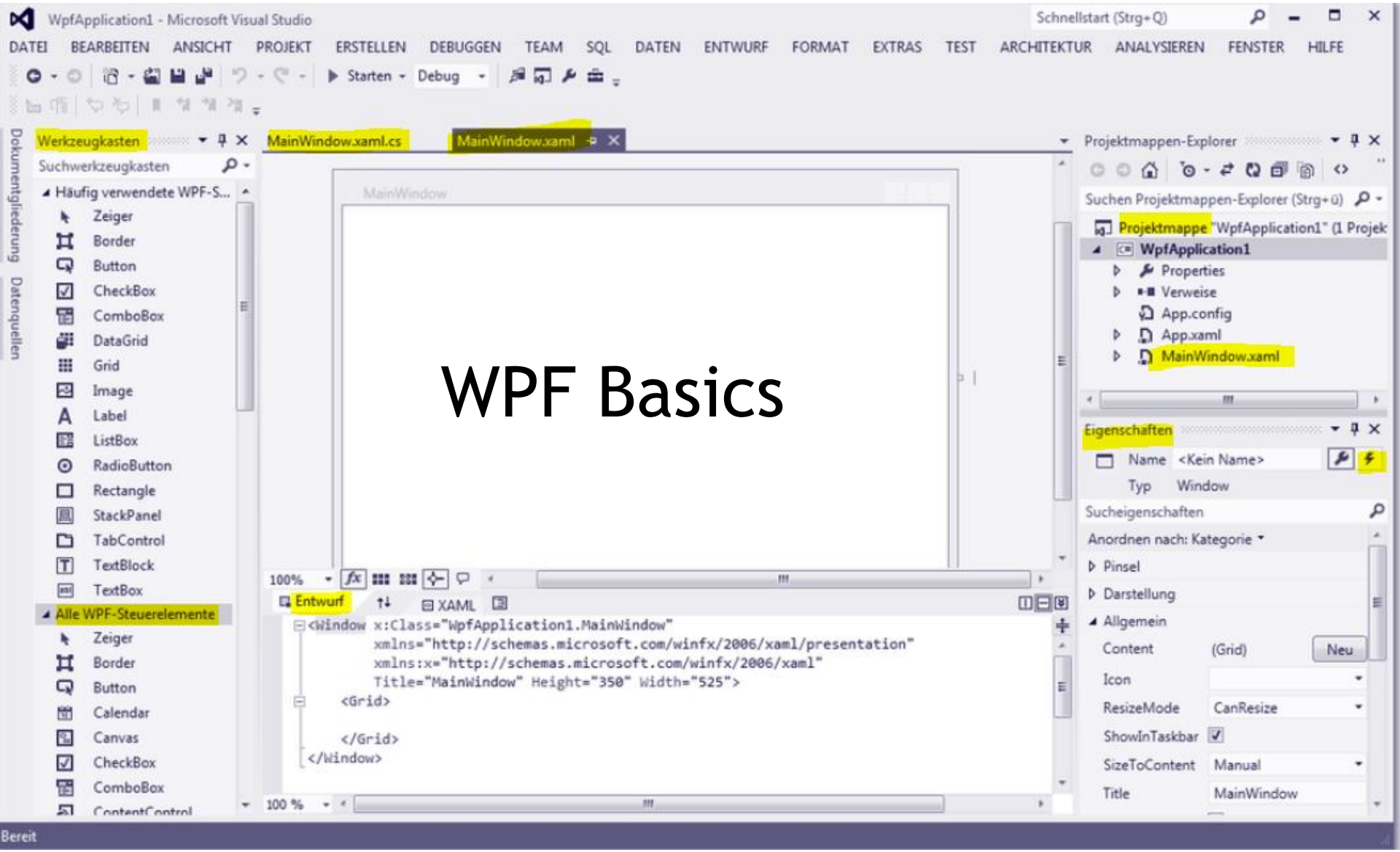
WPF Controls

Software Entwicklung



Overview

- WPF Basics
  - New Window
  - MessageBox
- WPF Controls
- WPF Panels
- WPF Data Binding
  - https://www.codeproject.com/Articles/1112919/MVVM-for-beginners
- WPF MVVM
  - https://www.codeproject.com/Articles/1052346/ICommand-Interface-in-WPF
- WPF ICommand
  - https://www.codeproject.com/Articles/1052346/ICommand-Interface-in-WPF
  - https://msdn.microsoft.com/en-us/magazine/dd419663.aspx#id0090030


WpfApplication1 - Microsoft Visual Studio
Schnellstart (Strg+Q)
DATEI BEARBEITEN ANSICHT PROJEKT ERSTELLEN DEBUGGEN TEAM SQL DATEN ENTWURF FORMAT EXTRAS TEST ARCHITEKTUR ANALYSIEREN FENSTER HILFE
Starten
Debug
Werkzeugkasten
Suchwerkzeugkasten
Häufig verwendete WPF-S...
Zeiger
Border
Button
CheckBox
ComboBox
DataGrid
Grid
Image
Label
ListBox
RadioButton
Rectangle
StackPanel
TabControl
TextBlock
TextBox
Alle WPF-Steuerelemente
Zeiger
Border
Button
Calendar
Canvas
CheckBox
ComboBox
MainWindow.xaml.cs
MainWindow.xaml
MainWindow
100%
Entwurf
XAML
<Window x:Class="WpfApplication1.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">
<Grid>
</Grid>
</Window>
100 %
Projektmappen-Explorer
Suchen Projektmappen-Explorer (Strg+ü)
Projektmappe "WpfApplication1" (1 Projek
WpfApplication1
Properties
Verweise
App.config
App.xaml
MainWindow.xaml
Eigenschaften
Name <Kein Name>
Typ Window
Sucheigenschaften
Anordnen nach: Kategorie
Pinsel
Darstellung
Allgemein
Content (Grid)
Neu
Icon
ResizeMode CanResize
ShowInTaskbar
SizeToContent Manual
Title MainWindow
Bereit

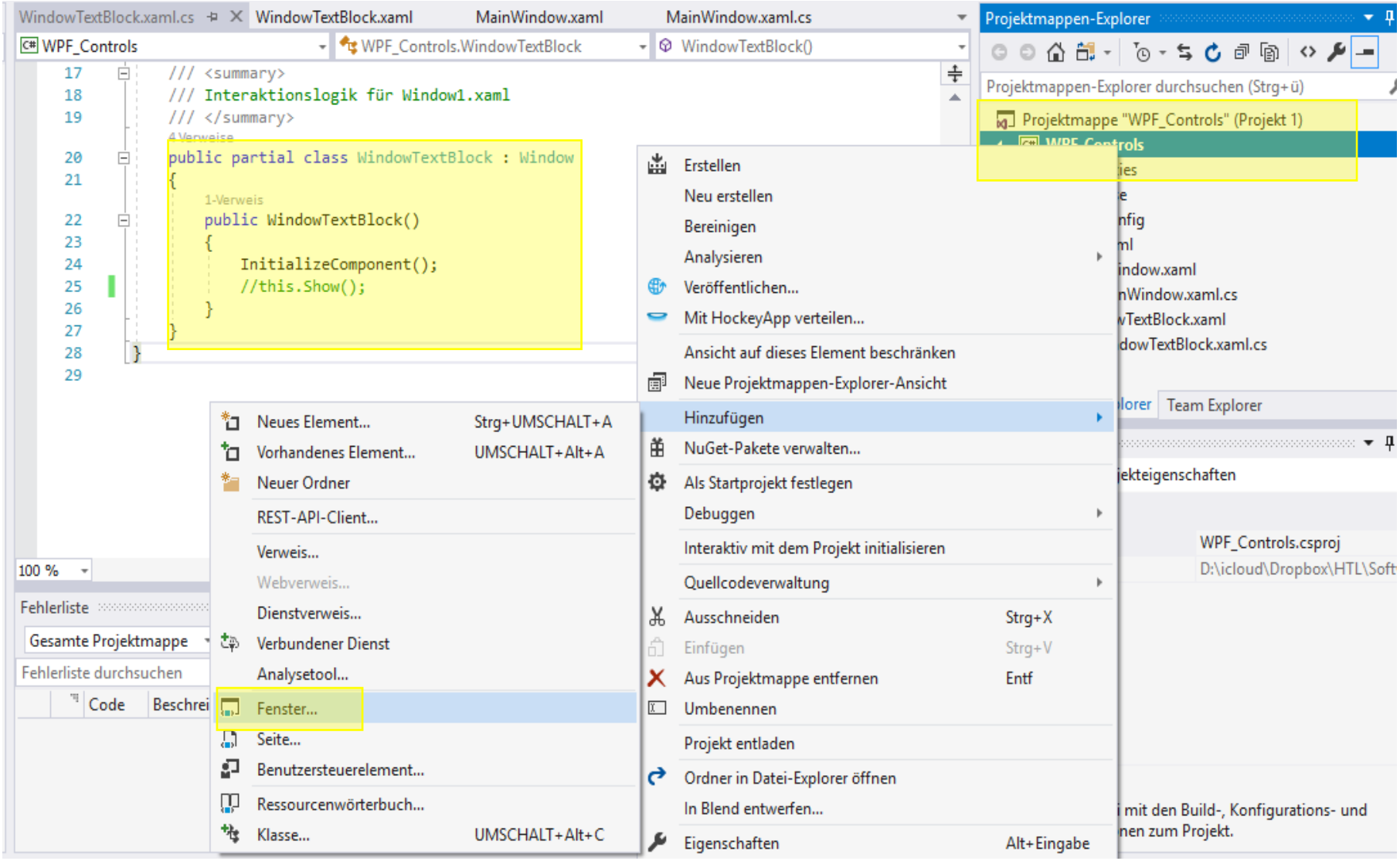
WPF Basics

CS & XAML Files

- XAML is a declarative markup language
  - XAML simplifies creating a UI
  - create visible UI elements in the declarative XAML markup
  - separate the UI definition from the run-time logic by using code-behind files
  - joined to the markup through partial class definitions

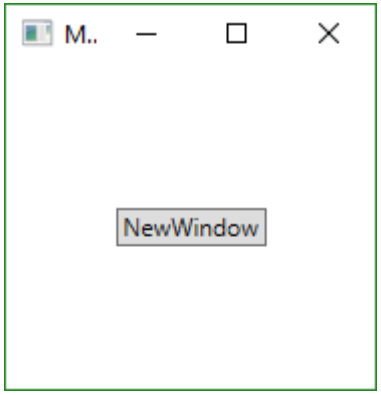
https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/xaml-in-wpf

Create a new Window(WPF)



New Window

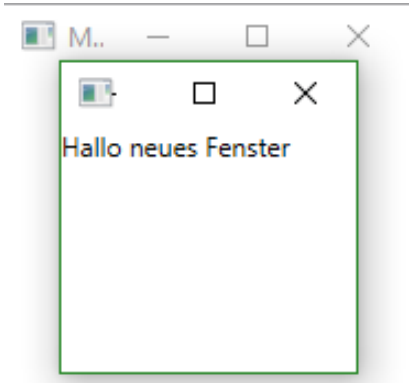
- Instantiate the Window and Show it:



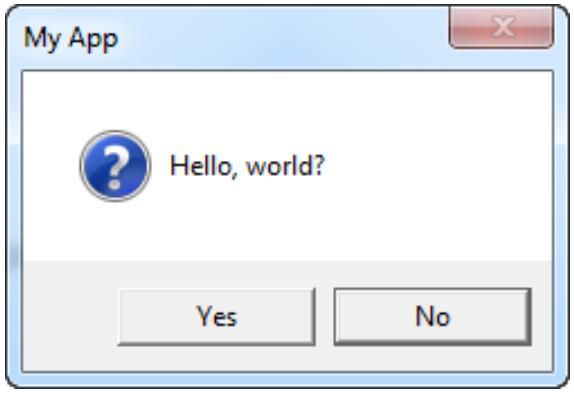

```
0 Verweise
public MainWindow()
{
    InitializeComponent();
}

1-Verweis
private void Button_Click(object sender, RoutedEventArgs e)
{
    WindowTextBlock w = new WindowTextBlock();
    w.Show();
}
```

- Add a new Window: WindowTextBlock

```
<Window x:Class="WPF_Controls.WindowTextBlock"
        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:WPF_Controls"
        mc:Ignorable="d"
        Title="NewWindow" Height="150" Width="150">
    <Grid>
        <TextBlock Text="Hallo neues Fenster"/>
    </Grid>
</Window>
```



Message Box



- purpose is:
  - show a message to the user
  - offer one or several ways for the user to respond to the message

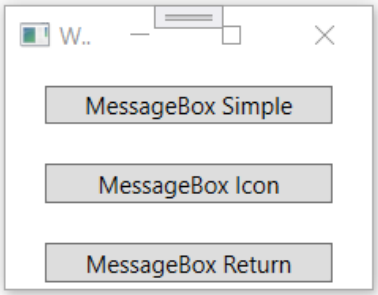
- MessageBox is used
  - by calling the static Show() method
  - MessageBox.Show("Hello, world!");

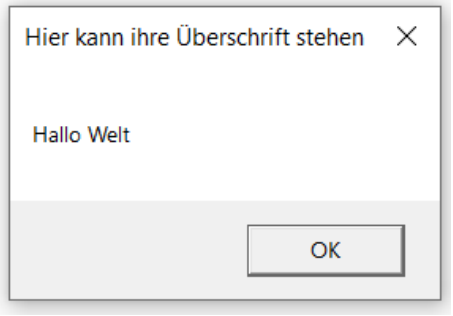
- Try the complete example:

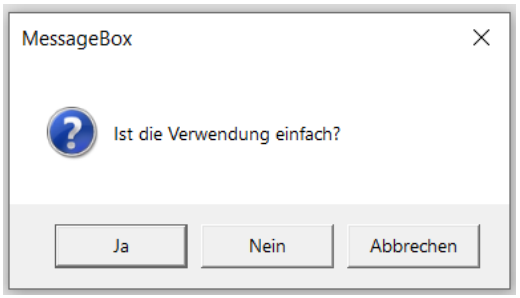
https://www.wpf-tutorial.com/dialogs/the-messagebox/

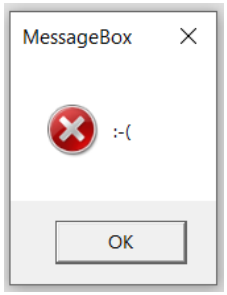
MessageBox

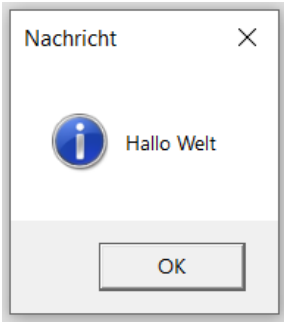
- Erstelle mehrere Messageboxen in einem Fenster









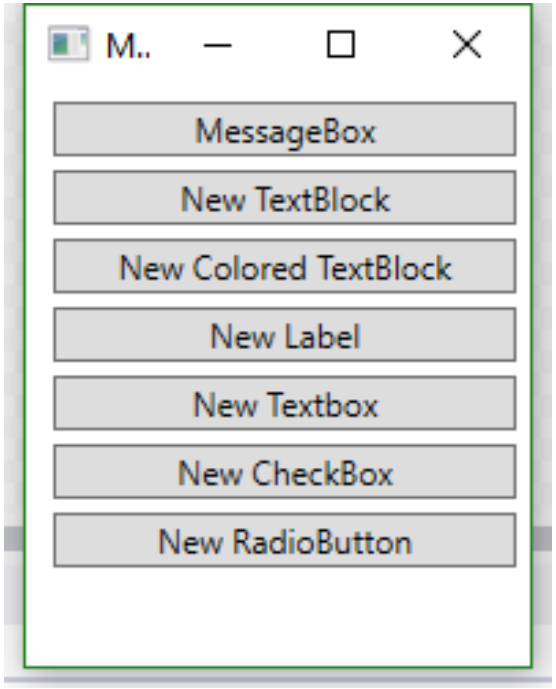


Message Box

```
private void btnSimple_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Hallo Welt", "Hier kann ihre Überschrift stehen");
}

private void btnIcon_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Hallo Welt", "Nachricht", MessageBoxButton.OK, MessageBoxImage.Information);
}

private void btnReturn_Click(object sender, RoutedEventArgs e)
{
    MessageBoxResult result;
    result = MessageBox.Show("Ist die Verwendung einfach?", "MessageBox",
        MessageBoxButton.YesNoCancel, MessageBoxImage.Question, MessageBoxResult.Yes);
    if (result == MessageBoxResult.Yes)
        MessageBox.Show("Erfreulich!", "MessageBox", MessageBoxButton.OK, MessageBoxImage.Exclamation);
    else
        MessageBox.Show(":-(", "MessageBox", MessageBoxButton.OK, MessageBoxImage.Error);
}
```

WPF Controls

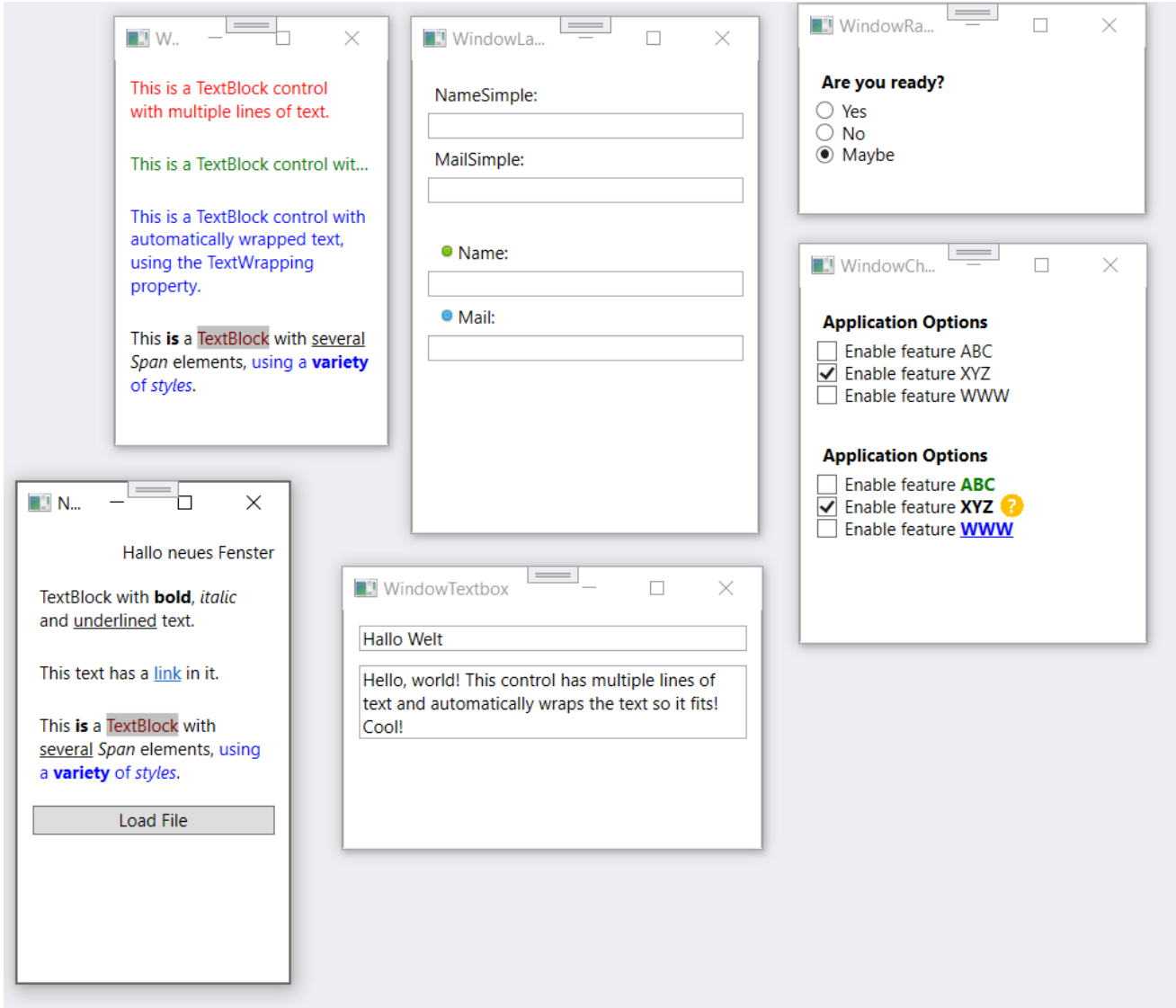
TextBlock
Label
TextBox
CheckBox
RadioButton

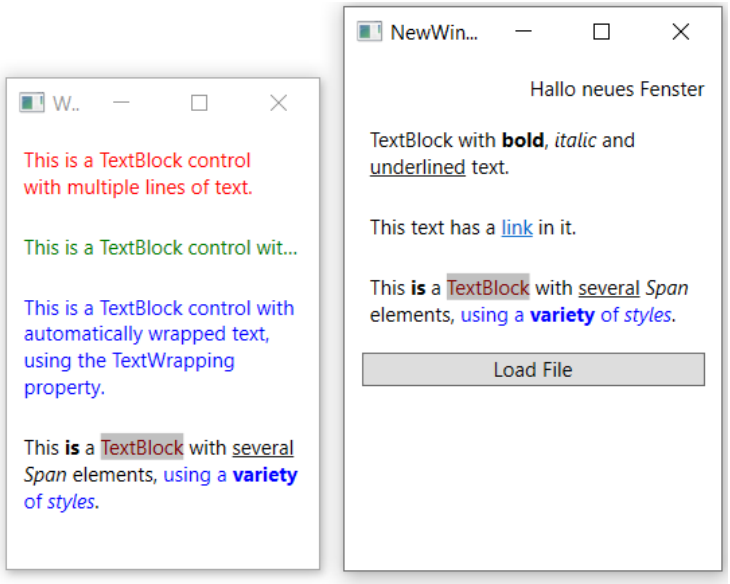


Create new Windows

Work with:

- Textblock
- Textbox
- Label
- Checkbox
- Radiobutton





TextBlock

Show a Text in a TextBlock

Read from a file and show the content in the TextBlock

Format the text colourful and stylish

https://wpf-tutorial.com/basic-controls/the-textblock-control/

TextBlock

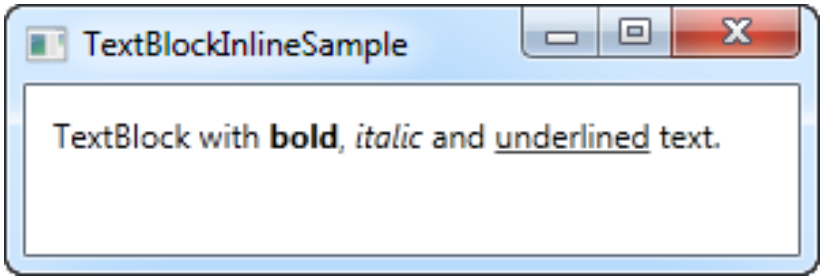
- Add
  - Textblock to a new window
- Set
  - Margin
  - TextWraping
  - TextAlignment
  - TextTrimming


```
<Grid>
    <TextBlock Name="TblFileContent" Margin="5" TextAlignment="Right"
               TextWrapping="Wrap" Text="Hallo neues Fenster"/>
</Grid>
```

https://docs.microsoft.com/en-us/dotnet/api/system.windows.controls.textblock?view=netframework-4.7.2

TextBlock

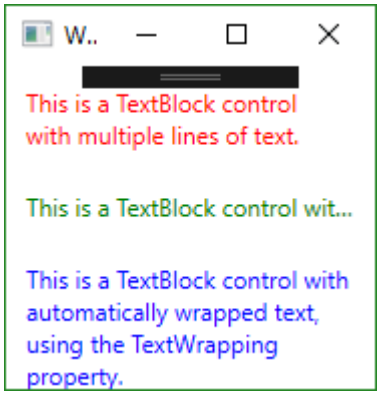
- Advanced formatting:
  - Bold
  - Italic
  - Underline




```
<TextBlock Margin="10" TextWrapping="Wrap">
    TextBlock with <Bold>bold</Bold>,
    <Italic>italic</Italic> and
    <Underline>underlined</Underline> text.
</TextBlock>
```

https://www.wpf-tutorial.com/basic-controls/the-textblock-control-inline-formatting/

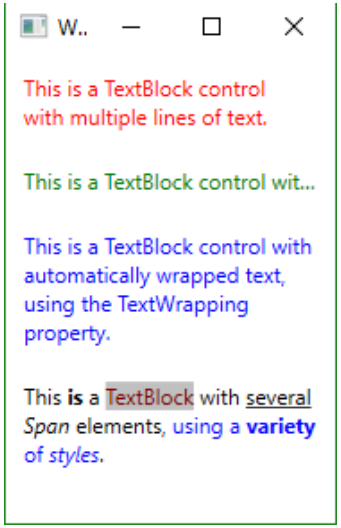
Color and wrap the Text




```
    Title="WindowTextBlockColor" Height="200" Width="200">
<StackPanel>
    <TextBlock Margin="10" Foreground="Red">
        This is a TextBlock control<LineBreak />
        with multiple lines of text.
    </TextBlock>
    <TextBlock Margin="10" TextTrimming="CharacterEllipsis" Foreground="Green">
        This is a TextBlock control with text that may not be
        rendered completely, which will be indicated with an ellipsis.
    </TextBlock>
    <TextBlock Margin="10" TextWrapping="Wrap" Foreground="Blue">
        This is a TextBlock control with automatically wrapped text,
        using the TextWrapping property.
    </TextBlock>
</StackPanel>
```

Formatting the Text in the Textblock

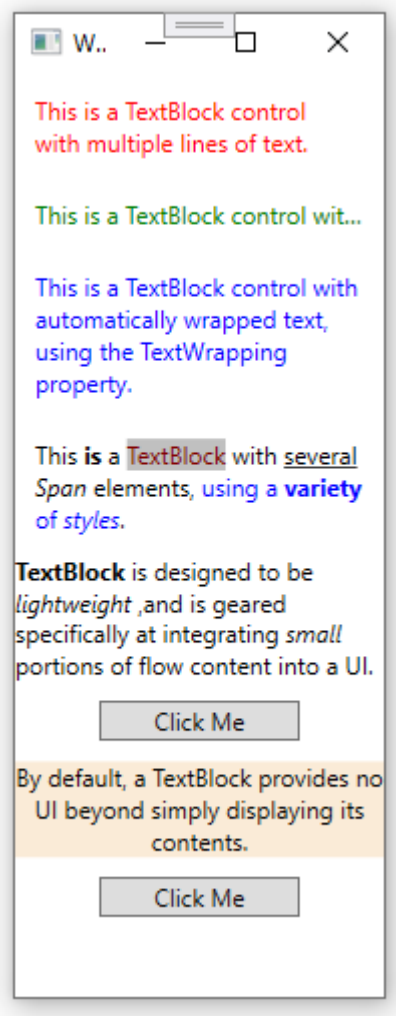
- Using Span
  - Background & Foreground
  - TextDecorations
  - FontStyle
  - Bold, Italic, Underline

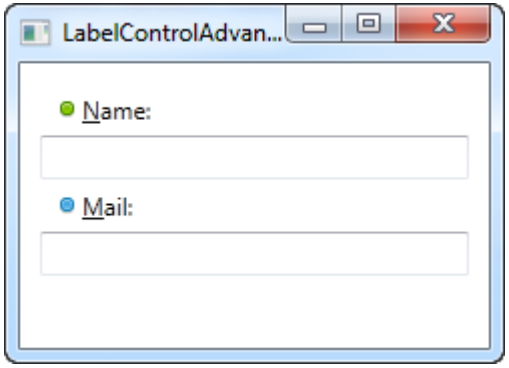



```
<TextBlock Margin="10" TextWrapping="Wrap">
    This <Span FontWeight="Bold">is</Span> a
    <Span Background="Silver" Foreground="Maroon">TextBlock</Span>
    with <Span TextDecorations="Underline">several</Span>
    <Span FontStyle="Italic">Span</Span> elements,
    <Span Foreground="Blue">
        using a <Bold>variety</Bold> of <Italic>styles</Italic>
    </Span>.
</TextBlock>
```

Textblock: Formatting the Text

```
<TextBlock Name="textBlock1" TextWrapping="Wrap">
    <Bold>TextBlock</Bold>
    is designed to be
    <Italic>lightweight</Italic>
    ,and is geared specifically at integrating
    <Italic>small</Italic>
    portions of flow content into a UI.
</TextBlock>
<Button Width="100" Margin="10">Click Me</Button>
<TextBlock  Name="textBlock2" TextWrapping="Wrap"
            Background="AntiqueWhite" TextAlignment="Center">
    By default, a TextBlock provides no UI beyond
    simply displaying its contents.
</TextBlock>
<Button Width="100" Margin="10">Click Me</Button>
```





Labels

Content instead of Text Property

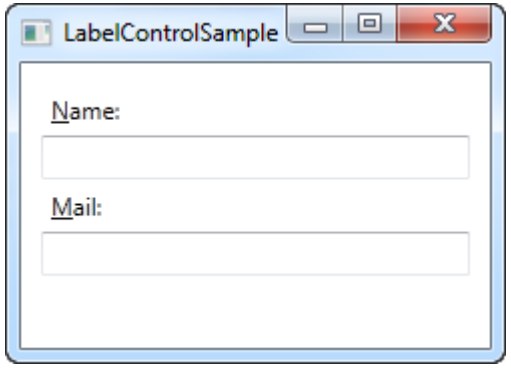
https://wpf-tutorial.com/basic-controls/the-label-control/

Labels

- use the **Target** property to connect the Label and the designated control

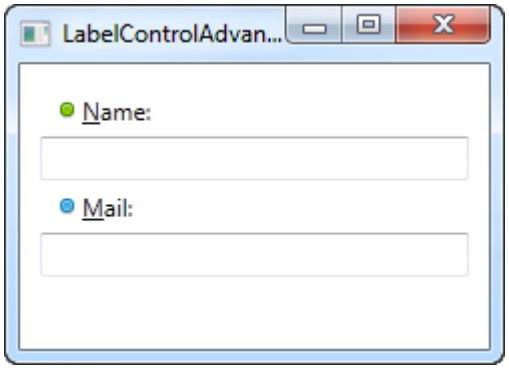
```
<StackPanel Margin="10">
    <Label Content="_Name:" Target="{Binding ElementName=txtName}" />
        <TextBox Name="txtName" />
        <Label Content="_Mail:" Target="{Binding ElementName=txtMail}" />
        <TextBox Name="txtMail" />
</StackPanel>
```

- Label a Textbox:
  - Name
  - Mail



Advanced Using of Labels

- Labels have a Content Property
  - Add a StackPanel as Content




```
    Title="WindowLabel" Height="180" Width="250">
<StackPanel Margin="10">
    <Label Target="{Binding ElementName=txtName}">
        <StackPanel Orientation="Horizontal">
            <Image Source="http://cdn1.iconfinder.com/data/icons/fatcow/16/bullet_green.png" />
            <AccessText Text="_Name:" />
        </StackPanel>
    </Label>
    <TextBox Name="txtName" />
    <Label Target="{Binding ElementName=txtMail}">
        <StackPanel Orientation="Horizontal">
            <Image Source="http://cdn1.iconfinder.com/data/icons/fatcow/16/bullet_blue.png" />
            <AccessText Text="_Mail:" />
        </StackPanel>
    </Label>
    <TextBox Name="txtMail" />
</StackPanel>
```

Textbox

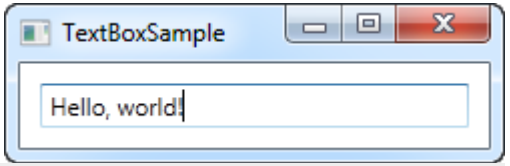
Text-input control in WPF

write plain text, on a single line, for dialog input, or in multiple lines, like an editor

https://wpf-tutorial.com/basic-controls/the-textbox-control/

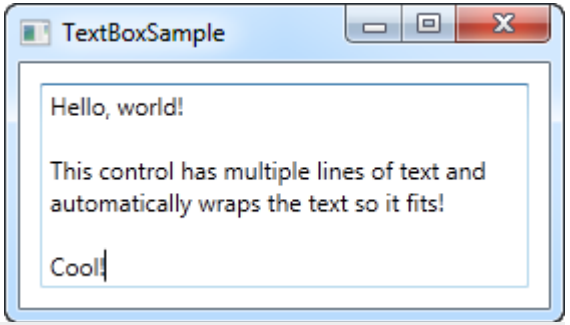
Textbox

- Single Line
  - Text

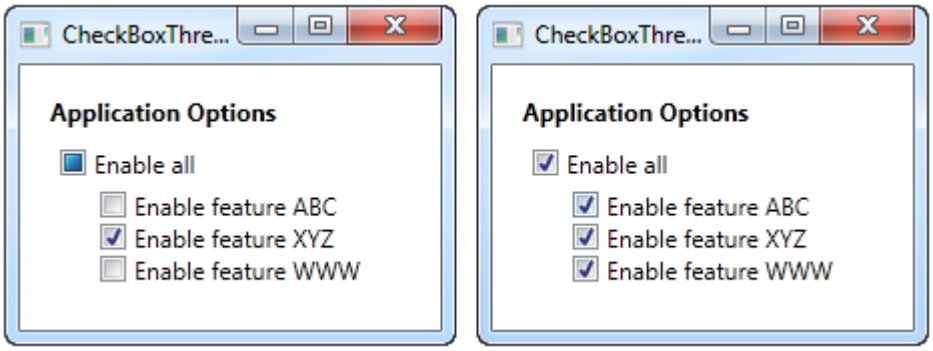



```
<TextBox Text="Hello, world!" />
```

- Multiple Lines
  - AcceptsReturn
  - TextWrapping




```
<Grid Margin="10">
            <TextBox AcceptsReturn="True" TextWrapping="Wrap" />
    </Grid>
```



CheckBox

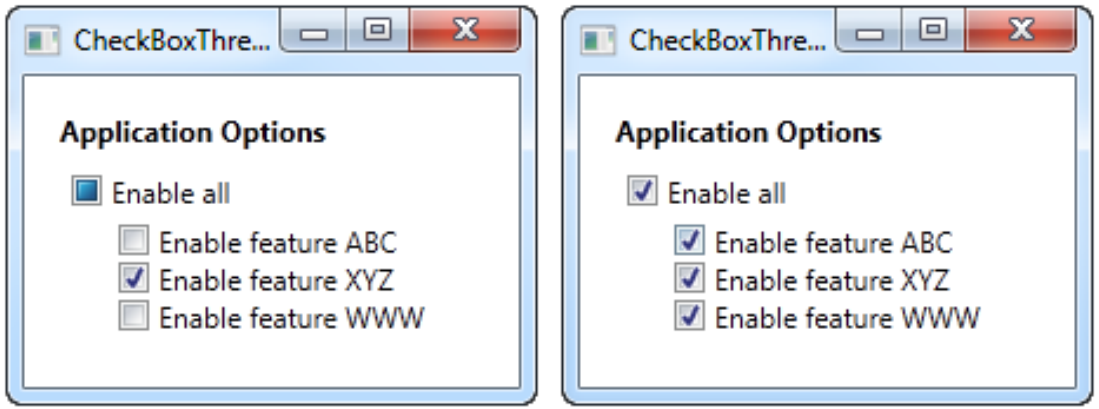
Select one or multiple choices

https://wpf-tutorial.com/basic-controls/the-checkbox-control/

Checkbox

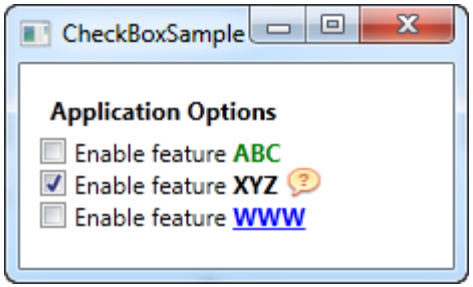
- Simple Style

```
    Title="WindowCheckBox" Height="140" Width="250">
<StackPanel Margin="10">
    <Label FontWeight="Bold">Application Options</Label>
    <CheckBox>Enable feature ABC</CheckBox>
    <CheckBox IsChecked="True">Enable feature XYZ</CheckBox>
    <CheckBox>Enable feature WWW</CheckBox>
</StackPanel>
```



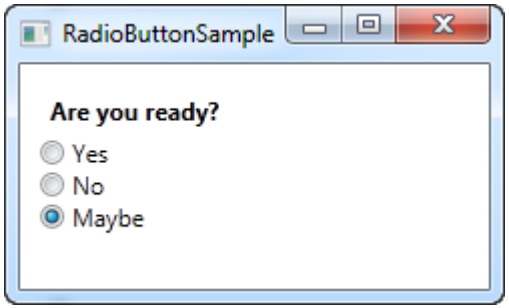
Checkbox Advanced

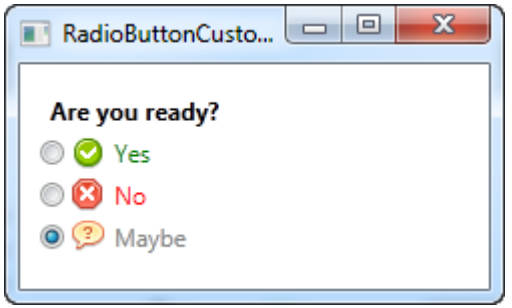
- With **Custom content**




```
<StackPanel Margin="10">
    <Label FontWeight="Bold">Application Options</Label>
    <CheckBox>
        <TextBlock>
            Enable feature <Run Foreground="Green"
                                FontWeight="Bold">ABC</Run>
        </TextBlock>
    </CheckBox>
    <CheckBox IsChecked="True">
        <WrapPanel>
            <TextBlock>
                Enable feature <Run FontWeight="Bold">XYZ</Run>
            </TextBlock>
            <Image Source="https://upload.wikimedia.org/wikipedia/commons/f/f6/Lol_question_mark.png"
                   Width="16" Height="16" Margin="5,0" />
        </WrapPanel>
    </CheckBox>
    <CheckBox>
        <TextBlock>
            Enable feature <Run Foreground="Blue"
                                TextDecorations="Underline"
                                FontWeight="Bold">WWW</Run>
        </TextBlock>
    </CheckBox>
</StackPanel>
```

You typically use the Run element only when you want to format a discrete section of text within the TextBlock.





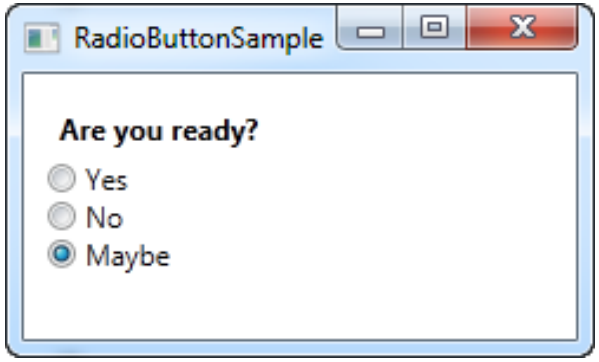
RadioButton

allows you to give your user a list of possible options

- achieve the same effect, using less space, with the ComboBox control

RadioButton

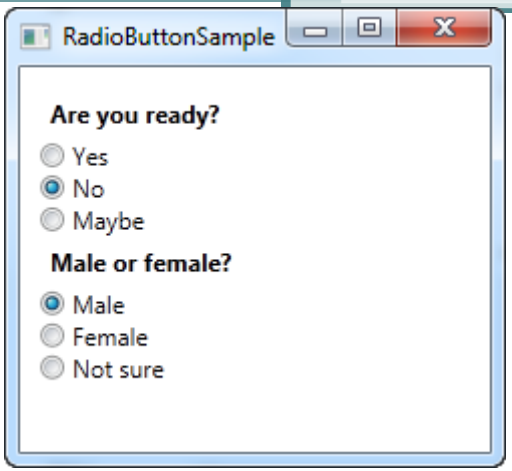
- Make your choice, only choose one…




```
    Title="WindowRadioButton" Height="150" Width="250">
<StackPanel Margin="10">
    <Label FontWeight="Bold">Are you ready?</Label>
    <RadioButton>Yes</RadioButton>
    <RadioButton>No</RadioButton>
    <RadioButton IsChecked="True">Maybe</RadioButton>
</StackPanel>
```

RadioButton Groups

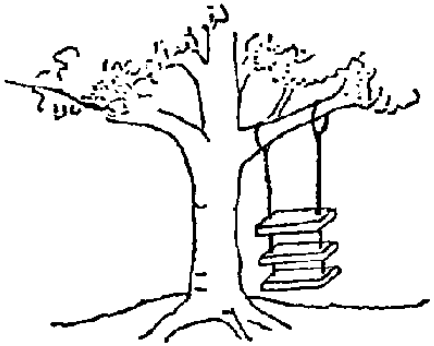
- Group your RadioButtons




```
<StackPanel Margin="10">
        <Label FontWeight="Bold">Are you ready?</Label>
        <RadioButton GroupName="ready">Yes</RadioButton>
        <RadioButton GroupName="ready">No</RadioButton>
        <RadioButton GroupName="ready" IsChecked="True">Maybe</RadioButton>

        <Label FontWeight="Bold">Male or female?</Label>
        <RadioButton GroupName="sex">Male</RadioButton>
        <RadioButton GroupName="sex">Female</RadioButton>
        <RadioButton GroupName="sex" IsChecked="True">Not sure</RadioButton>
</StackPanel>
```

WPF Panels

Panel is the base control that works as a parent control for other child controls

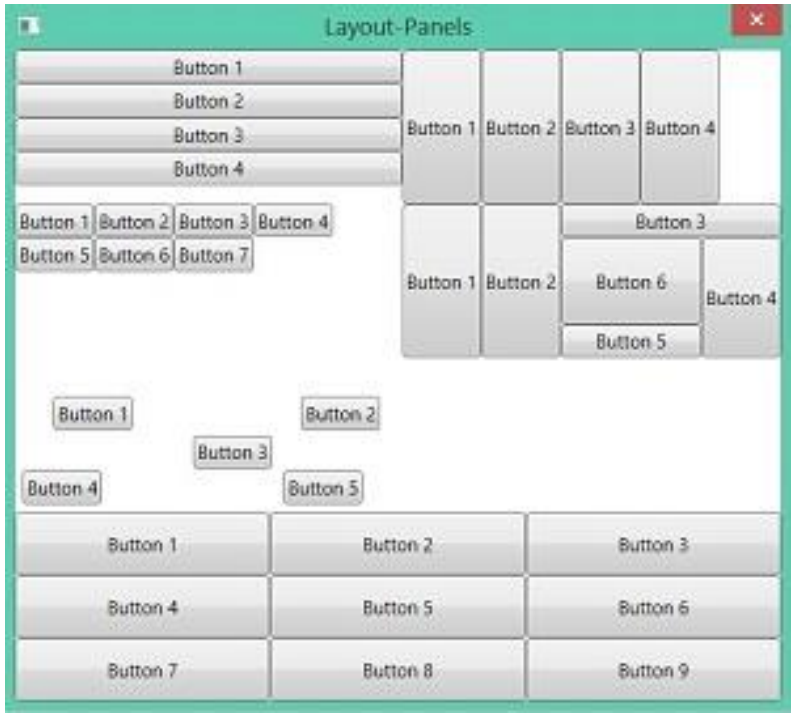


Overview

- WPF Basics
  - New Window
  - MessageBox
- WPF Controls
- WPF Panels
- WPF Data Binding
  - https://www.codeproject.com/Articles/1112919/MVVM-for-beginners
- WPF MVVM
  - https://www.codeproject.com/Articles/1052346/ICommand-Interface-in-WPF
- WPF ICommand
  - https://www.codeproject.com/Articles/1052346/ICommand-Interface-in-WPF
  - https://msdn.microsoft.com/en-us/magazine/dd419663.aspx#id0090030

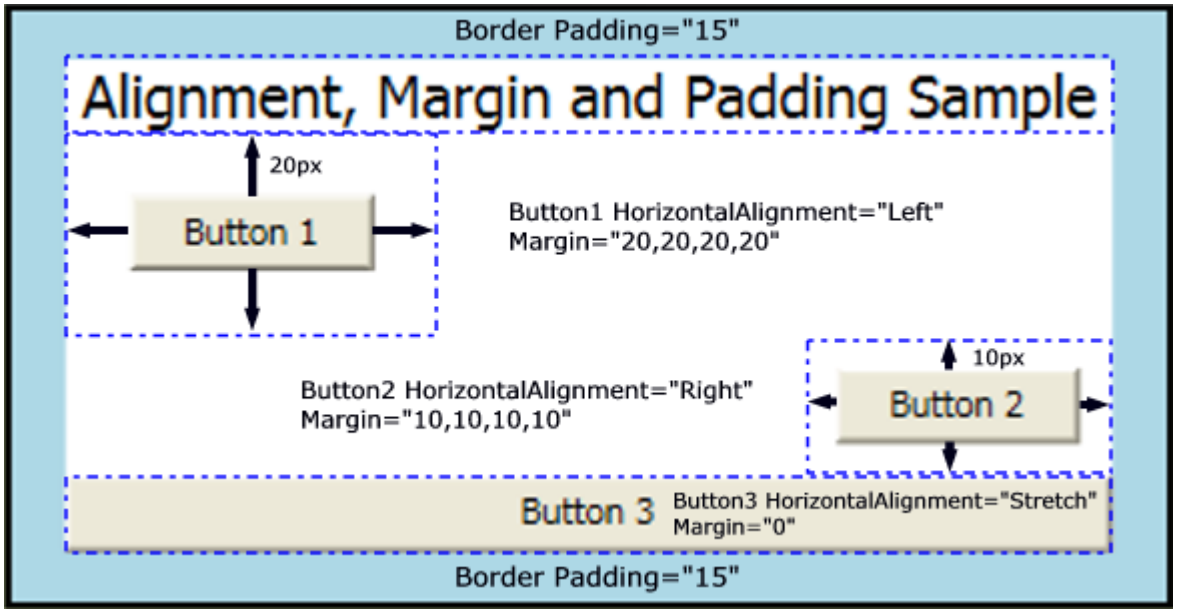
Panels in Detail

- Margin & Allignment
- Margin vs. Padding
- Panel Overview
  - Canvas
  - WrapPanel
  - StackPanel
  - DockPanel
  - Grid



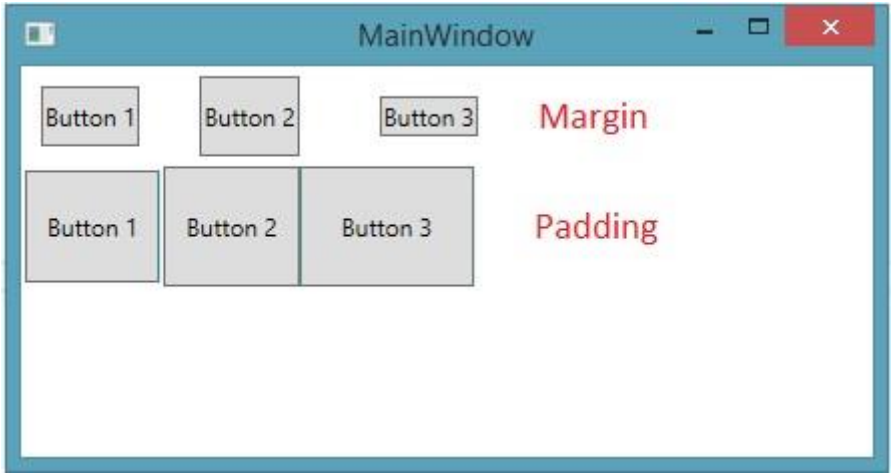
Margin & Allignment

- Margin: the distance between…
- Allignment: Left, Right, Top, Bottom

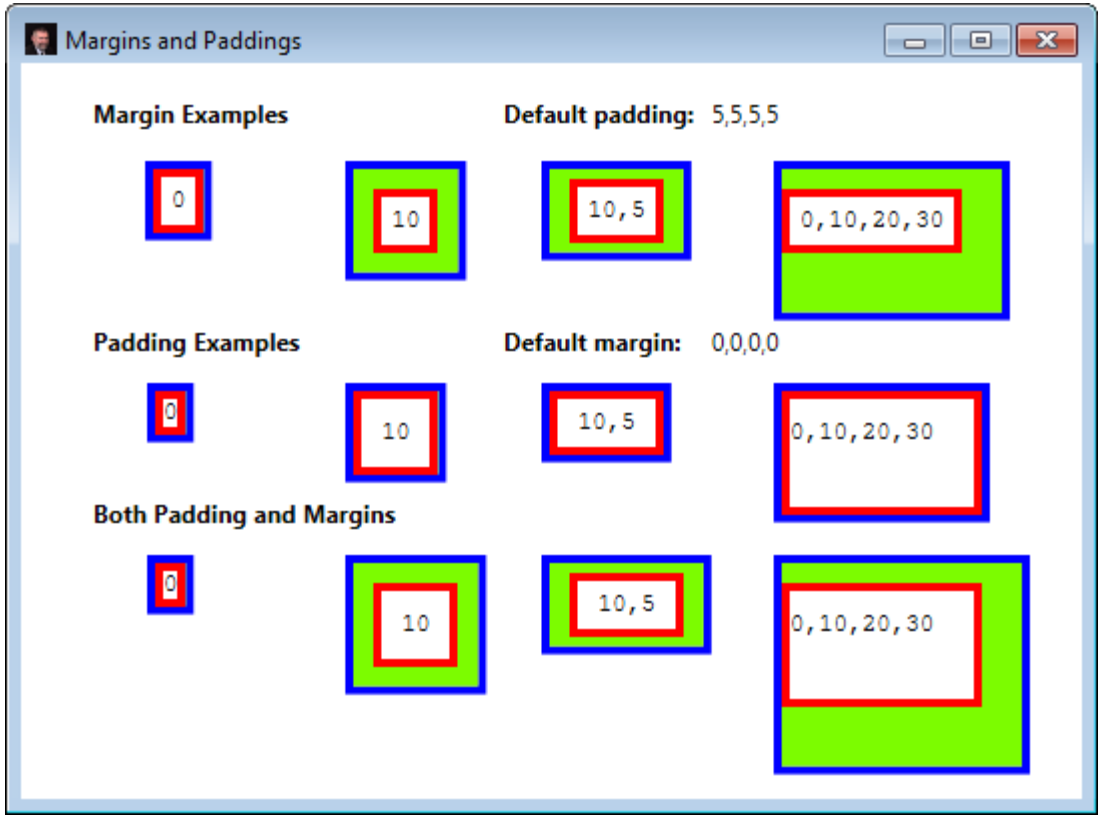


Margin vs Padding

- Margin
  - controls how much extra space gets placed around the outside edges of the element
- Padding
  - controls around the inside edges of the element

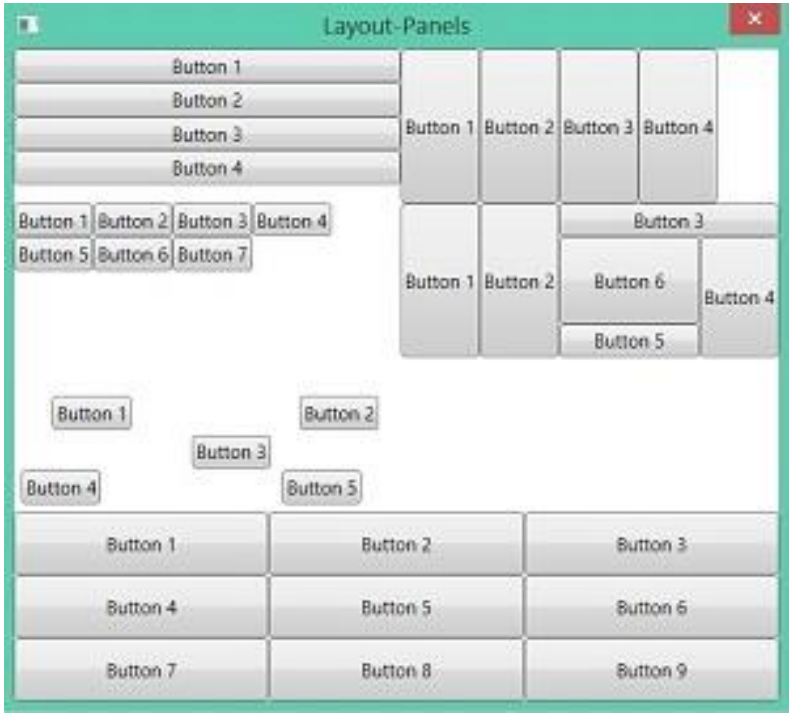


Example Padding & Margin



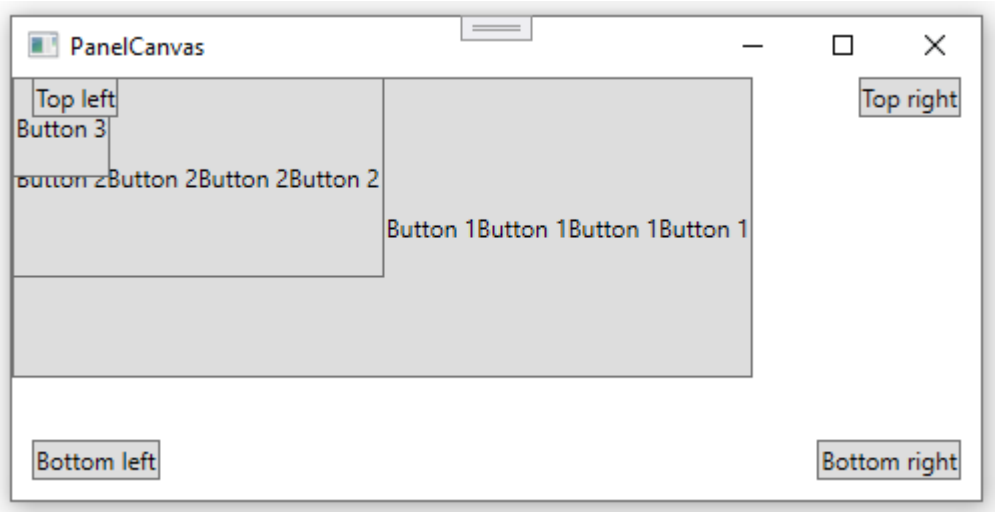
https://www.codeproject.com/Articles/321295/WPF-Controls-Tester

The Windows Presentation Foundation (WPF) provides a number of predefined Panel elements:



Panels

Canvas	used for graphical Objects
WrapPanel	puts one element next to another
StackPanel	puts element after element (expanding)
DockPanel	docks on top, bottom, left & right
Grid	uses rows and columns

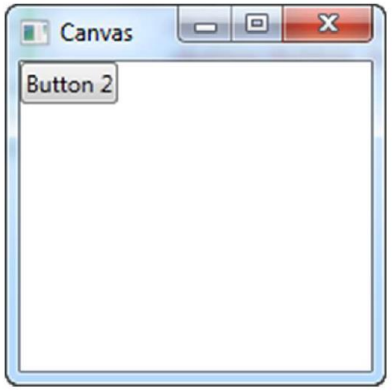


Canvas

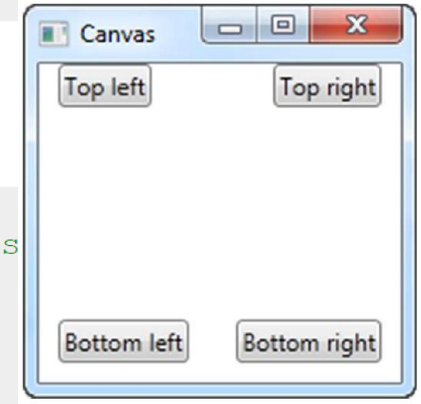
https://wpf-tutorial.com/panels/canvas/

Canvas with Buttons

```
<Window x:Class="WpfTutorialSamples.Panels.Canvas"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xa
        Title="Canvas" Height="200" Width="200">
        <Canvas>
                <Button>Button 1</Button>
                <Button>Button 2</Button>
        </Canvas>
</Window>
```

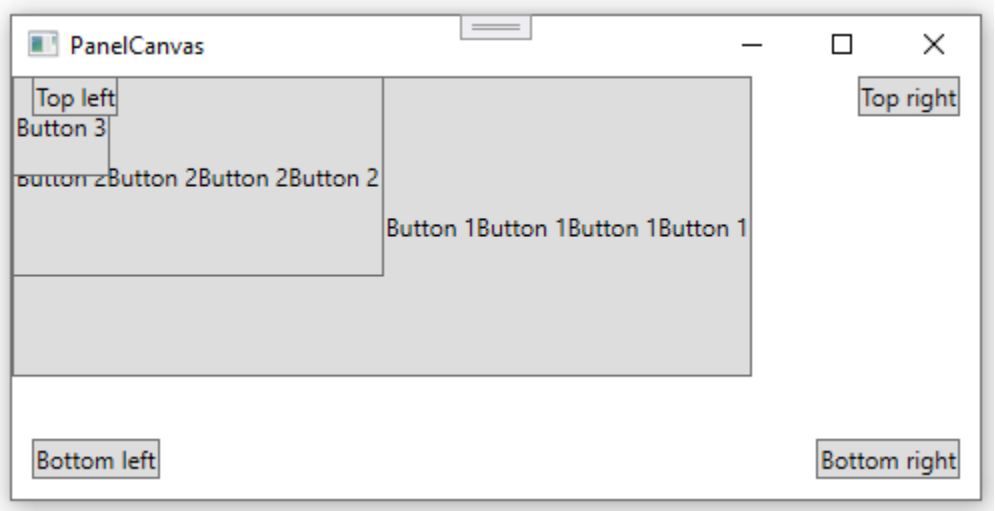


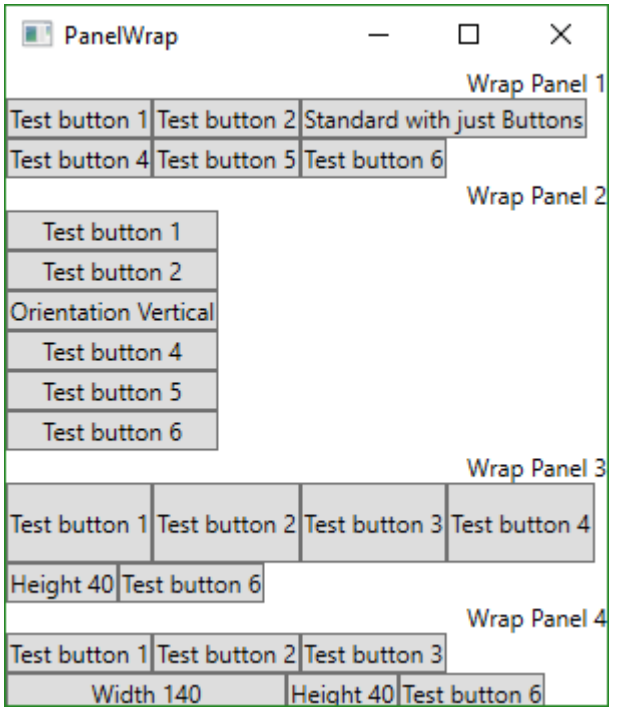

```
<Window x:Class="WpfTutorialSamples.Panels.Canvas"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/pres
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Canvas" Height="200" Width="200">
        <Canvas>
                <Button Canvas.Left="10">Top left</Button>
                <Button Canvas.Right="10">Top right</Button>
                <Button Canvas.Left="10" Canvas.Bottom="10">Bottom left</Button>
                <Button Canvas.Right="10" Canvas.Bottom="10">Bottom right</Button>
        </Canvas>
</Window>
```



Canvas with lots of Buttons

```
    Title="PanelCanvas" Height="250" Width="500">

<Canvas>
    <Button Height="150">Button 1 Button 1Button 1Button
        1Button 1Button 1Button 1Button 1</Button>
    <Button Height ="100">Button 2Button 2Button 2Button 2</Button>
    <Button Height="50">Button 3</Button>
    <Button Canvas.Left="10">Top left</Button>
    <Button Canvas.Right="10">Top right</Button>
    <Button Canvas.Left="10" Canvas.Bottom="10">Bottom left</Button>
    <Button Canvas.Right="10" Canvas.Bottom="10">Bottom right</Button>
</Canvas>
```





WrapPanel

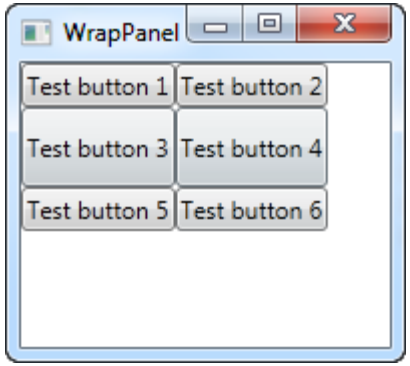
positions each of its child controls next to the other horizontally (default)

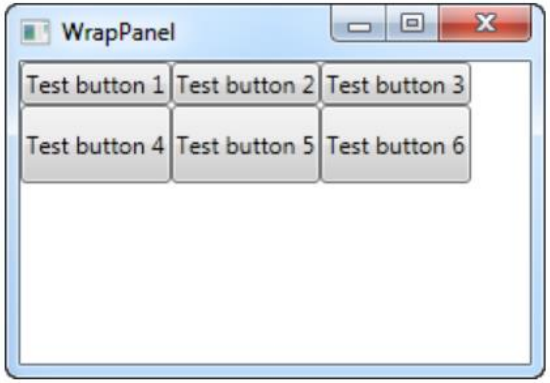
https://wpf-tutorial.com/panels/wrappanel/

WrapPanel

- position each of its child controls next to the other horizontally

  until there is no more room, where it will wrap to the next line and then continue



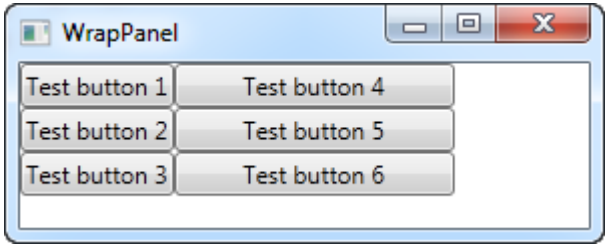



```
        Title="WrapPanel" Height="300" Width="300">
        <WrapPanel>
                <Button>Test button 1</Button>
                <Button>Test button 2</Button>
                <Button>Test button 3</Button>
                <Button Height="40">Test button 4</Button>
                <Button>Test button 5</Button>
                <Button>Test button 6</Button>
        </WrapPanel>
```

WrapPanel with diffrent Orientation

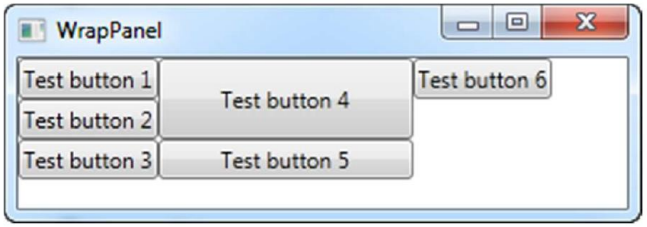
- Set Orientation: Vertical

```
<WrapPanel Orientation="Vertical">
    <Button>Test button 1</Button>
    <Button>Test button 2</Button>
    <Button>Test button 3</Button>
    <Button Width="140">Test button 4</Button>
    <Button>Test button 5</Button>
    <Button>Test button 6</Button>
</WrapPanel>
```



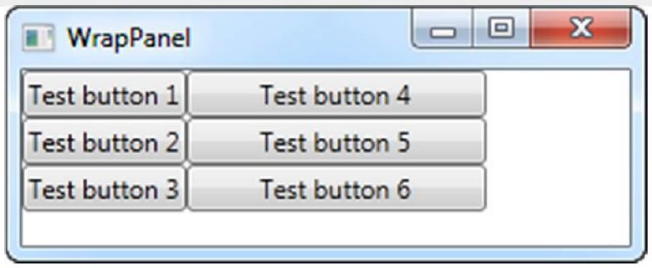
- Set Width & Hight:

```
<Button Width="140" Height="44">
                    Test button 4</Button>
```

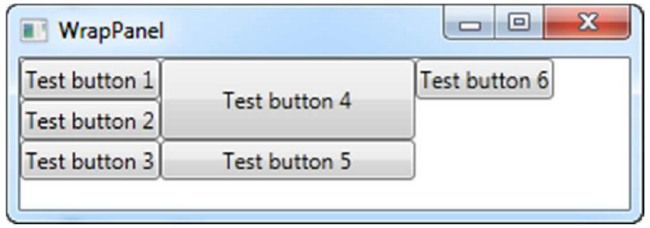


WrapPanel

- Test the Wrap Panel
  - with different Width and Height



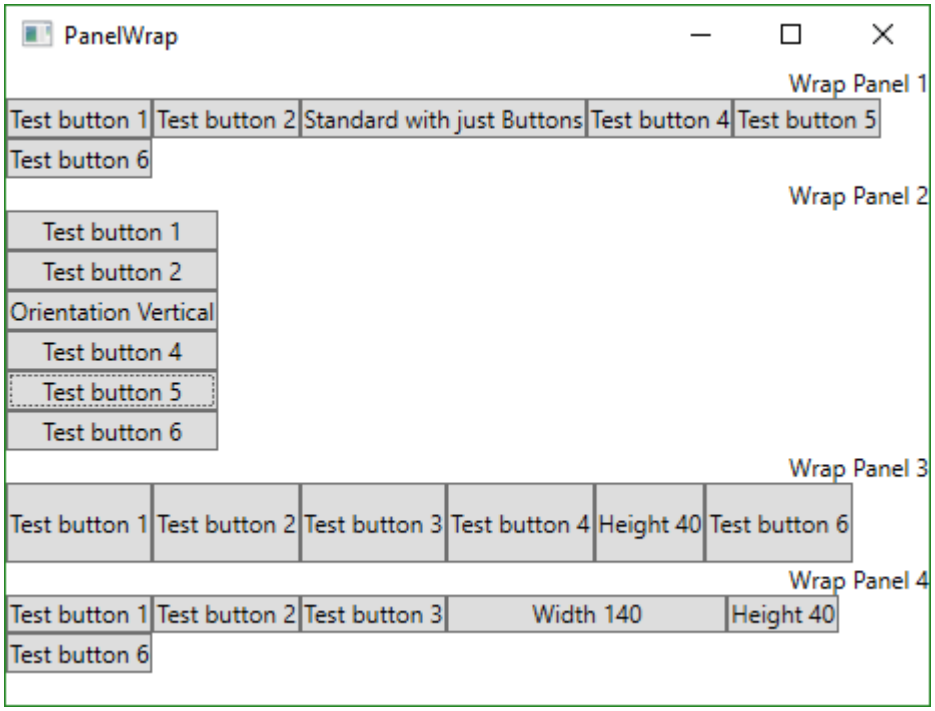

```
<Button Width="140">Test button 4</Button>
```

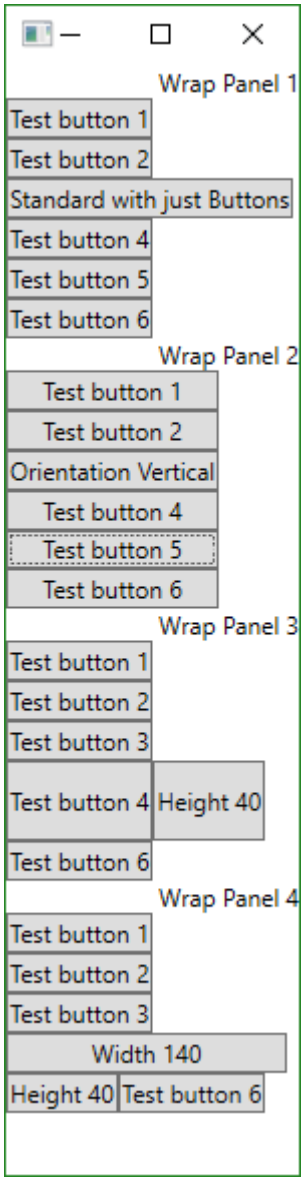


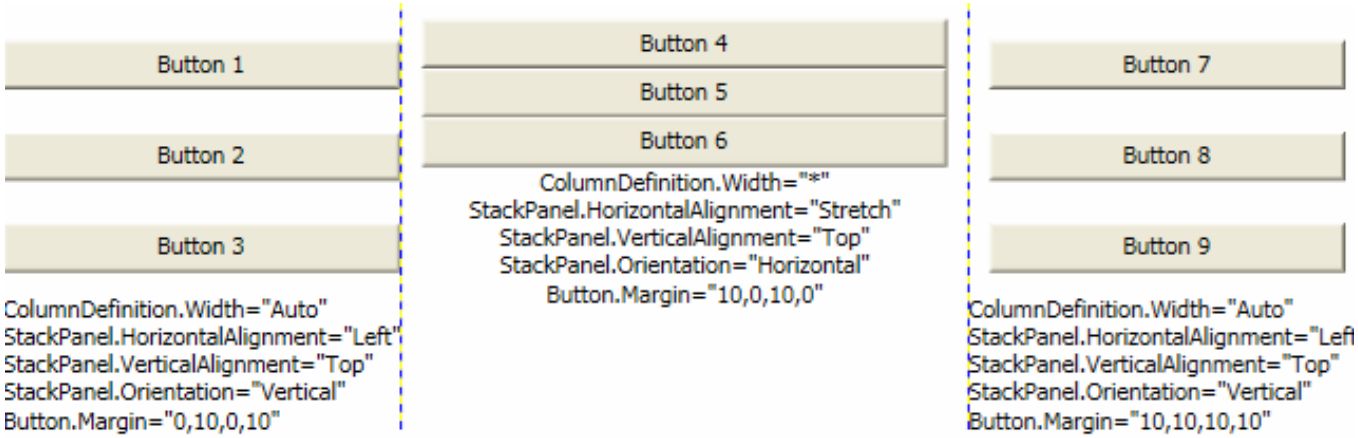

```
<Button Width="140" Height="44">Test button 4</Button>
```

Wrap Panel

- Resize your window during runtime







StackPanel

https://wpf-tutorial.com/panels/stackpanel/

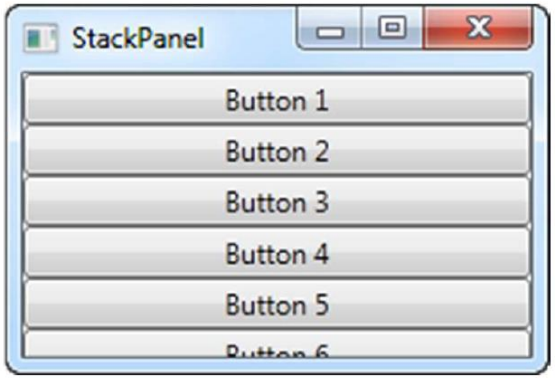
very similar to the WrapPanel,
but it doesn't wrap the content!

it stretches it content in one direction, allowing you to stack item after item on top of each other

StackPanel

- Default: Vertical Orientation

```
<StackPanel>
        <Button>Button 1</Button>
        <Button>Button 2</Button>
        <Button>Button 3</Button>
        <Button>Button 4</Button>
        <Button>Button 5</Button>
        <Button>Button 6</Button>
</StackPanel>
```

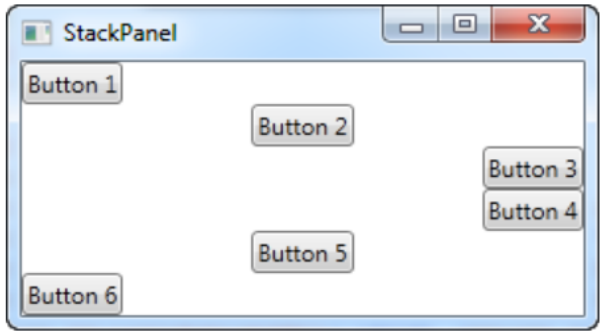


- Orientation: Horizontal:

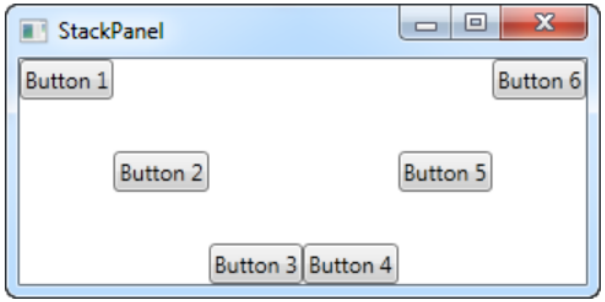
```
<StackPanel Orientation="Horizontal">
        <Button>Button 1</Button>
        <Button>Button 2</Button>
        <Button>Button 3</Button>
        <Button>Button 4</Button>
        <Button>Button 5</Button>
        <Button>Button 6</Button>
</StackPanel>
```

StackPanel

```
<StackPanel Orientation="Vertical">
        <Button HorizontalAlignment="Left">Button 1</Button>
        <Button HorizontalAlignment="Center">Button 2</Button>
        <Button HorizontalAlignment="Right">Button 3</Button>
        <Button HorizontalAlignment="Right">Button 4</Button>
        <Button HorizontalAlignment="Center">Button 5</Button>
        <Button HorizontalAlignment="Left">Button 6</Button>
</StackPanel>
```

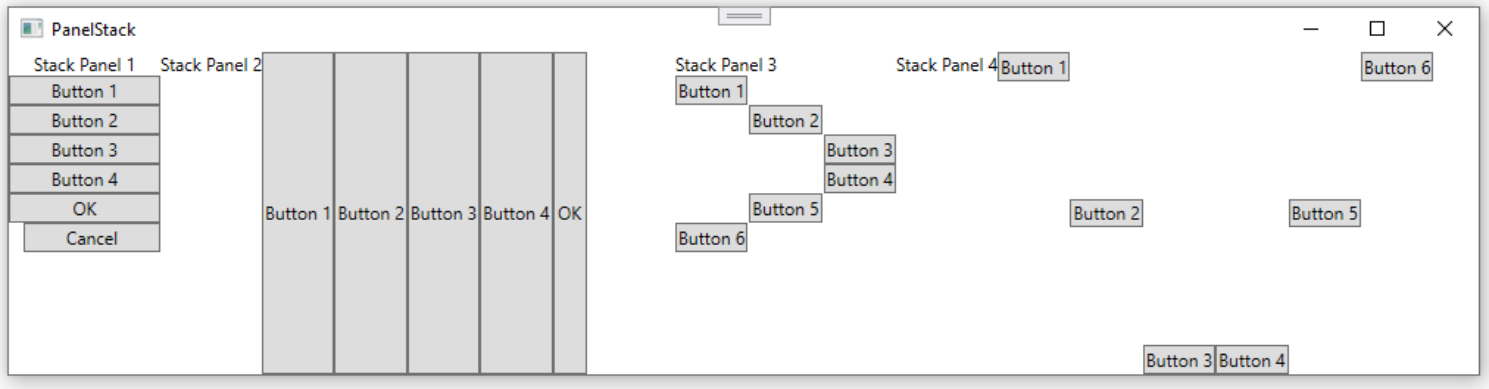



```
<StackPanel Orientation="Horizontal">
        <Button VerticalAlignment="Top">Button 1</Button>
        <Button VerticalAlignment="Center">Button 2</Button>
        <Button VerticalAlignment="Bottom">Button 3</Button>
        <Button VerticalAlignment="Bottom">Button 4</Button>
        <Button VerticalAlignment="Center">Button 5</Button>
        <Button VerticalAlignment="Top">Button 6</Button>
</StackPanel>
```



Test Stack Panels

- 5xStackPanel in an WrapPanel:

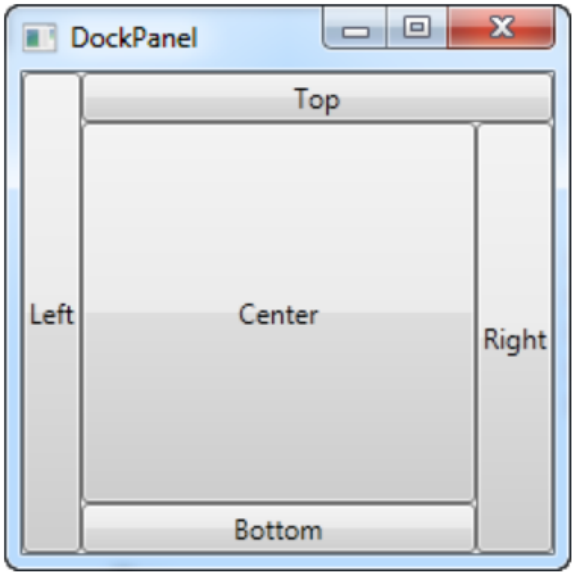


DockPanel

https://wpf-tutorial.com/panels/dockpanel/

DockPanel

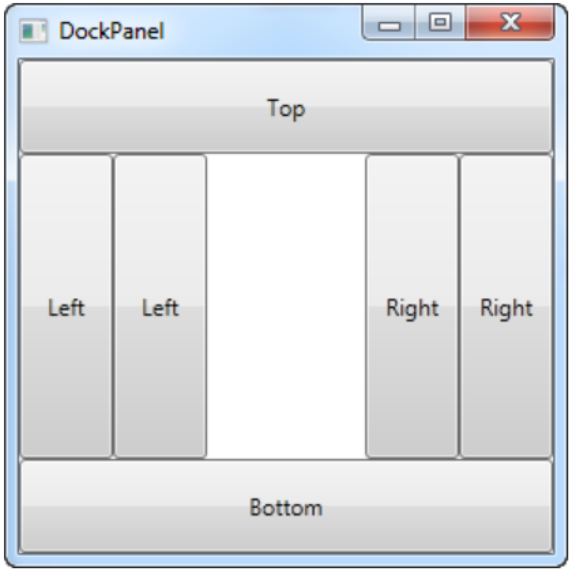
- makes it easy to dock content
  in all four directions:
  - top, bottom, left and right




```
<DockPanel>
        <Button DockPanel.Dock="Left">Left</Button>
        <Button DockPanel.Dock="Top">Top</Button>
        <Button DockPanel.Dock="Right">Right</Button>
        <Button DockPanel.Dock="Bottom">Bottom</Button>
        <Button>Center</Button>
</DockPanel>
```

DockPanel

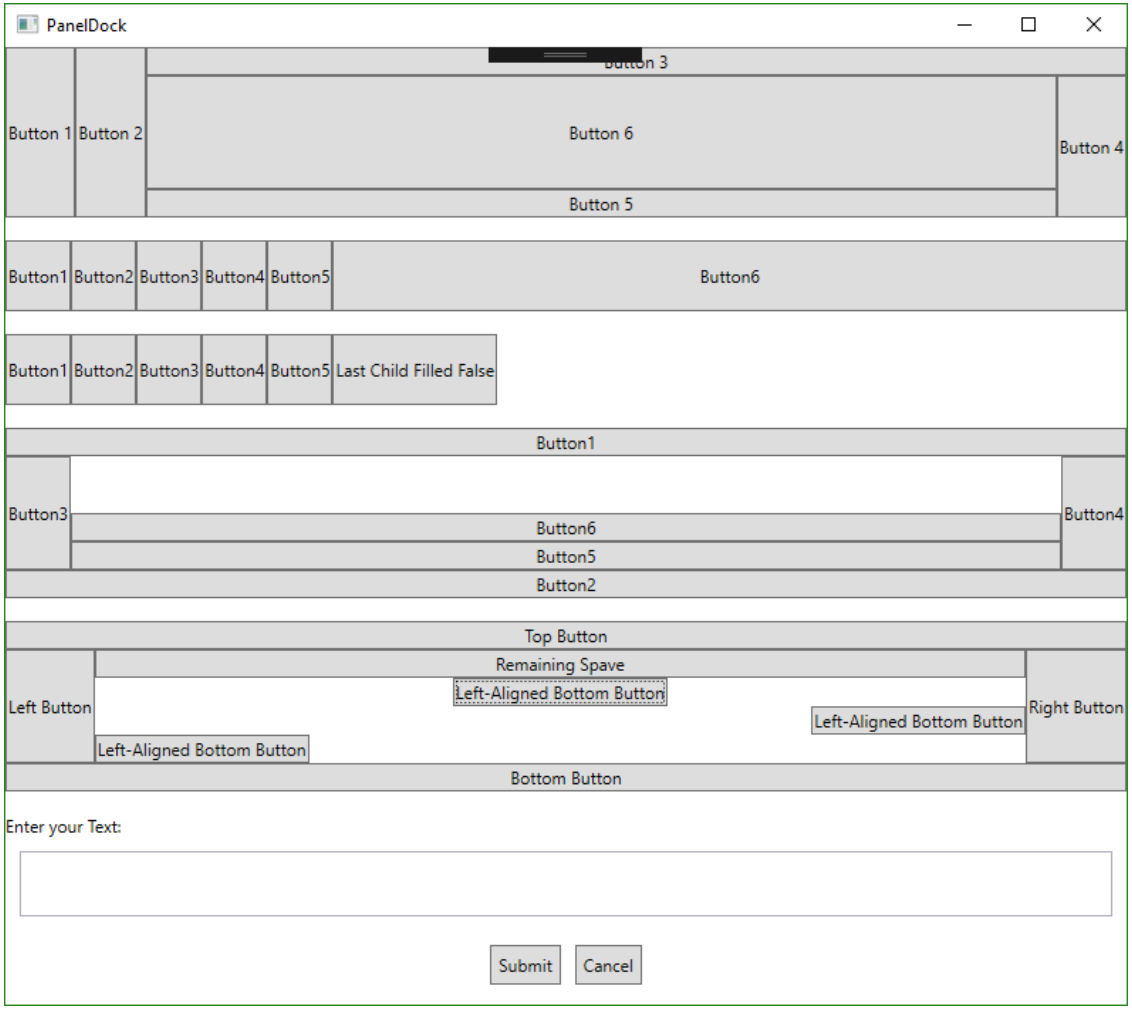
- Last Child Filled disabled



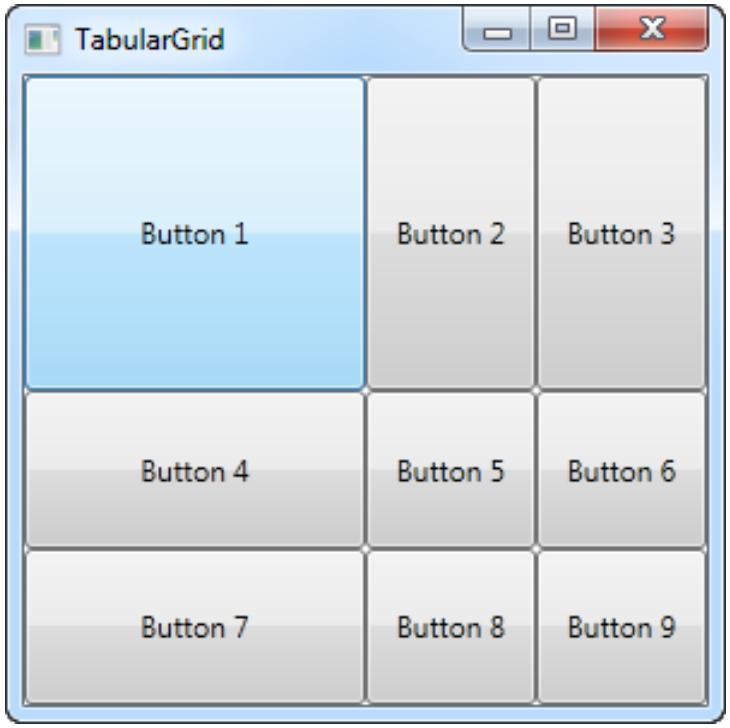

```
Title="DockPanel" Height="300" Width="300">
<DockPanel LastChildFill="False">
        <Button DockPanel.Dock="Top" Height="50">Top</Button>
        <Button DockPanel.Dock="Bottom" Height="50">Bottom</Button>
        <Button DockPanel.Dock="Left" Width="50">Left</Button>
        <Button DockPanel.Dock="Left" Width="50">Left</Button>
        <Button DockPanel.Dock="Right" Width="50">Right</Button>
        <Button DockPanel.Dock="Right" Width="50">Right</Button>
</DockPanel>
```

Dock Panel

- Different ways of using dock panels:



https://www.ittutorialswithexample.com/2014/08/dockpanel-in-wpf.html



Grid

https://www.wpf-tutorial.com/panels/grid/

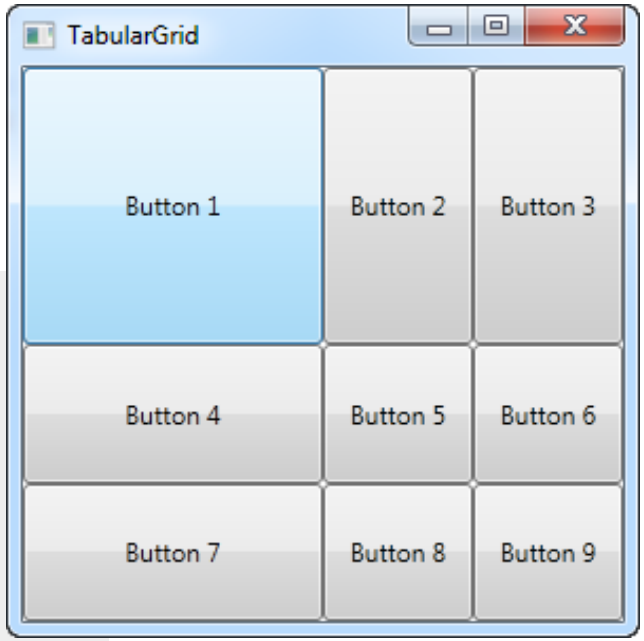
Grid

- most complex panel
- Can define
  - height for each of the rows
  - width for each of the columns
- Define the size
  - in absolute amount of pixels
  - in a percentage of the available space or
  - as auto where the row or column
    - will automatically adjust its size depending on the content

https://www.wpf-tutorial.com/panels/grid/

Grid

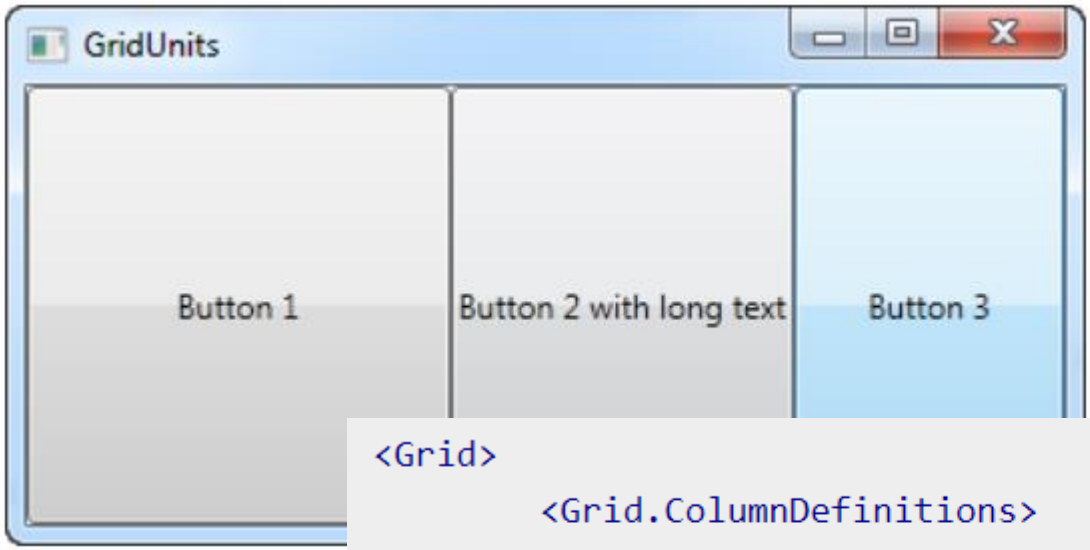
- with rows and colums

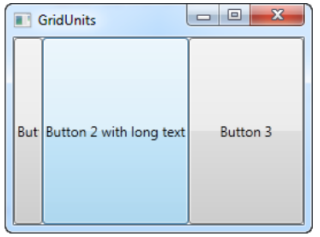



```
<Grid>
            <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="2*" />
                    <ColumnDefinition Width="1*" />
                    <ColumnDefinition Width="1*" />
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                    <RowDefinition Height="2*" />
                    <RowDefinition Height="1*" />
                    <RowDefinition Height="1*" />
            </Grid.RowDefinitions>
            <Button>Button 1</Button>
            <Button Grid.Column="1">Button 2</Button>
            <Button Grid.Column="2">Button 3</Button>
            <Button Grid.Row="1">Button 4</Button>
            <Button Grid.Column="1" Grid.Row="1">Button 5</Button>
            <Button Grid.Column="2" Grid.Row="1">Button 6</Button>
            <Button Grid.Row="2">Button 7</Button>
            <Button Grid.Column="1" Grid.Row="2">Button 8</Button>
            <Button Grid.Column="2" Grid.Row="2">Button 9</Button>
    </Grid>
```

https://www.wpf-tutorial.com/panels/grid-rows-and-columns/

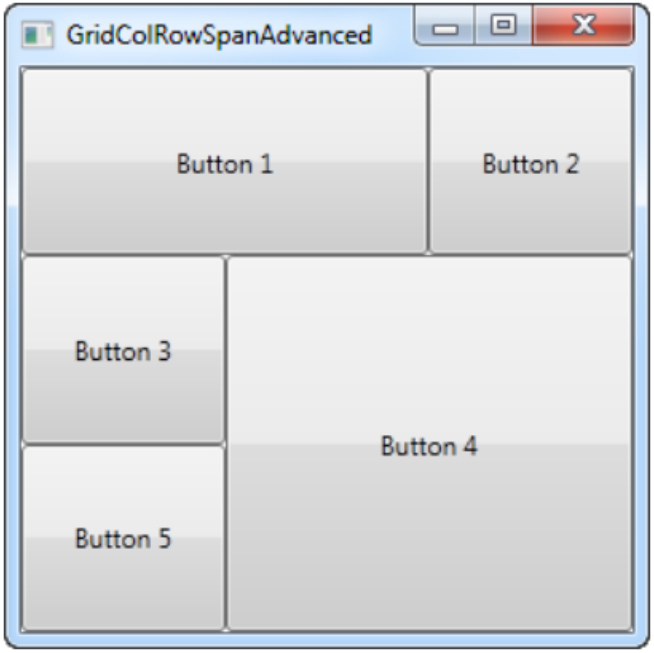
Grid: Units



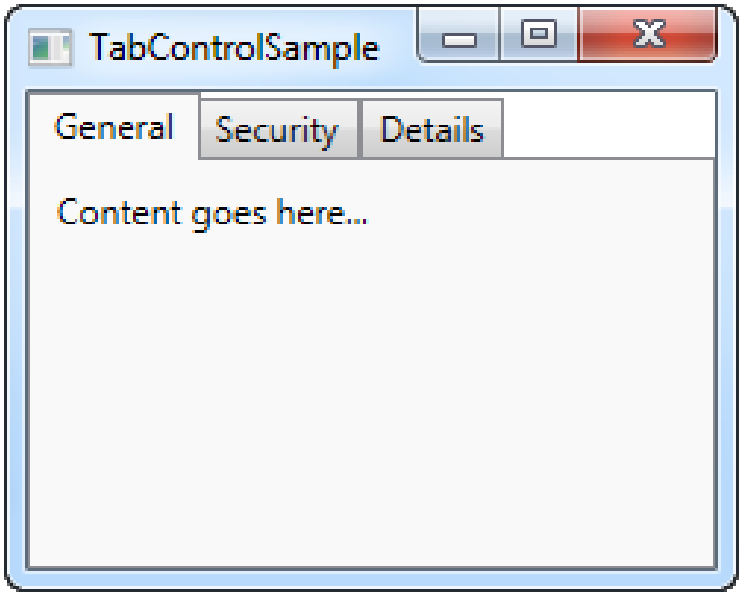



```
<Grid>
        <Grid.ColumnDefinitions>
                <ColumnDefinition Width="1*" />
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="100" />
        </Grid.ColumnDefinitions>
        <Button>Button 1</Button>
        <Button Grid.Column="1">Button 2 with long text</Button>
        <Button Grid.Column="2">Button 3</Button>
</Grid>
```

Grid - Spanning




```
<Grid.ColumnDefinitions>
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="*" />
        <RowDefinition Height="*" />
</Grid.RowDefinitions>
<Button Grid.ColumnSpan="2">Button 1</Button>
<Button Grid.Column="3">Button 2</Button>
<Button Grid.Row="1">Button 3</Button>
<Button Grid.Column="1" Grid.Row="1" Grid.RowSpan="2" Grid.ColumnSpan="2">Button 4</Button>
<Button Grid.Column="0" Grid.Row="2">Button 5</Button>
```



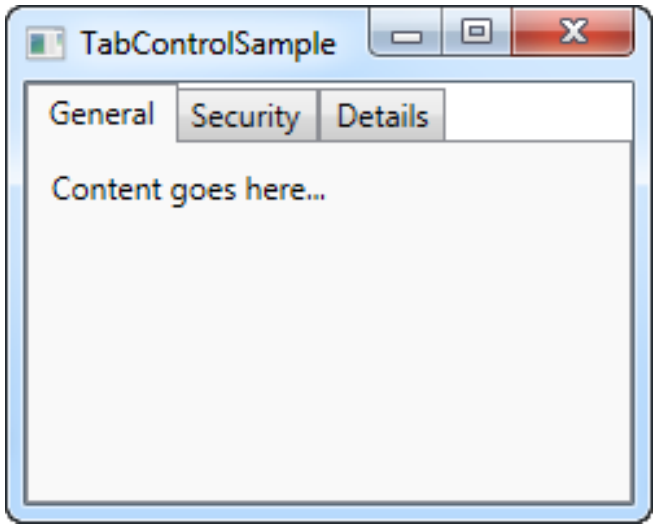
Tab Control

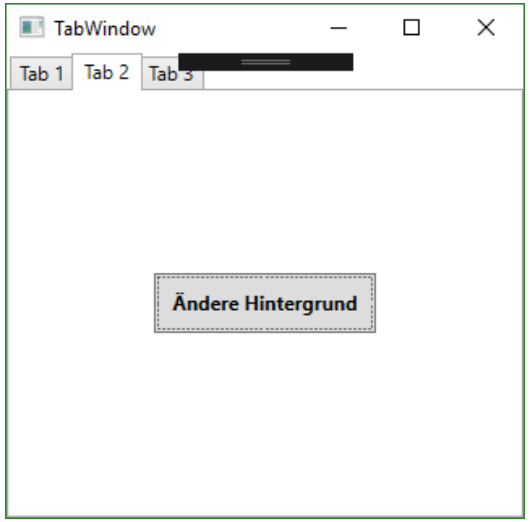
https://www.wpf-tutorial.com/tabcontrol/using-the-tabcontrol/

Tab Control

- split your interface up into different areas, each accessible by clicking on the tab header

```
<Grid>
    <TabControl>
        <TabItem Header="General">
            <Label Content="Content goes here..." />
        </TabItem>
        <TabItem Header="Security" />
        <TabItem Header="Details" />
    </TabControl>
</Grid>
```



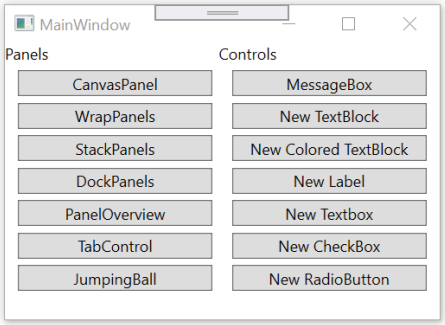


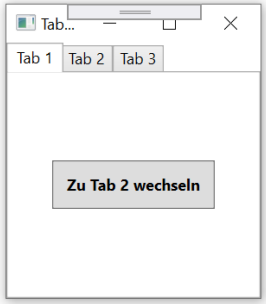
Tab Example

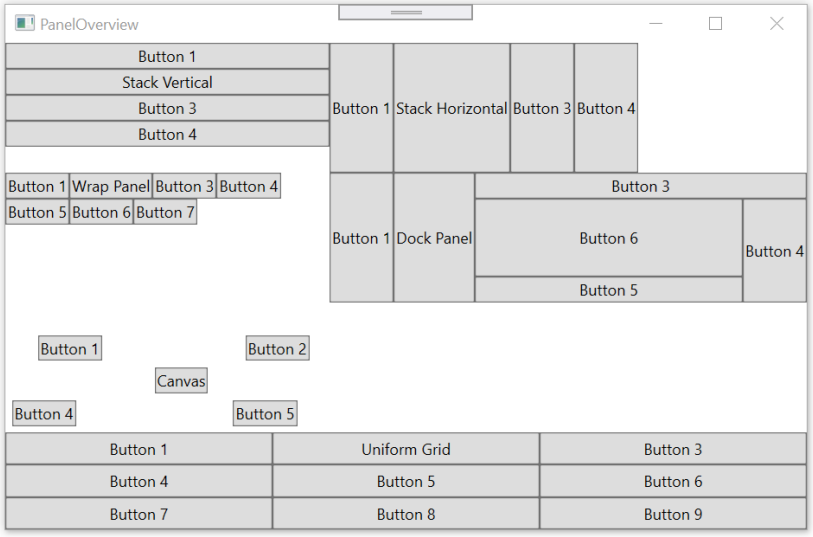
http://www.das-grosse-computer-abc.de/CSharp/index.php?chapter=12&topic=6

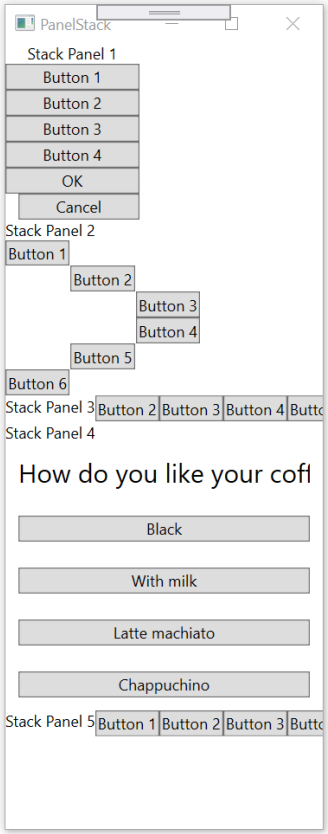
```
<TabControl Name="tabControlFenster">
    <TabItem Header="Tab 1">
        <Grid Background="White">
            <Button VerticalAlignment="Center" Hor
        </Grid>
    </TabItem>
    <TabItem Header="Tab 2">
        <Grid Background="White">
            <Button VerticalAlignment="Center" Hor
        </Grid>
    </TabItem>
    <TabItem Header="Tab 3">
        <Grid Background="White">
            <Button VerticalAlignment="Center" Hor
        </Grid>
    </TabItem>
</TabControl>
```

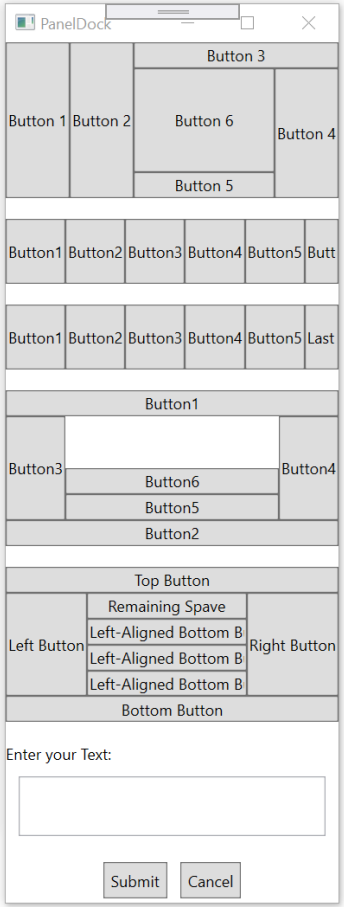
All together now… all Panels produced

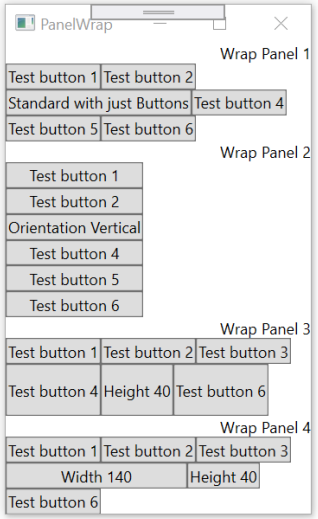


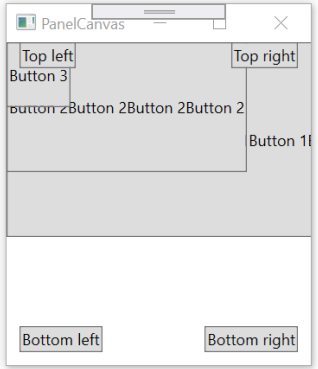






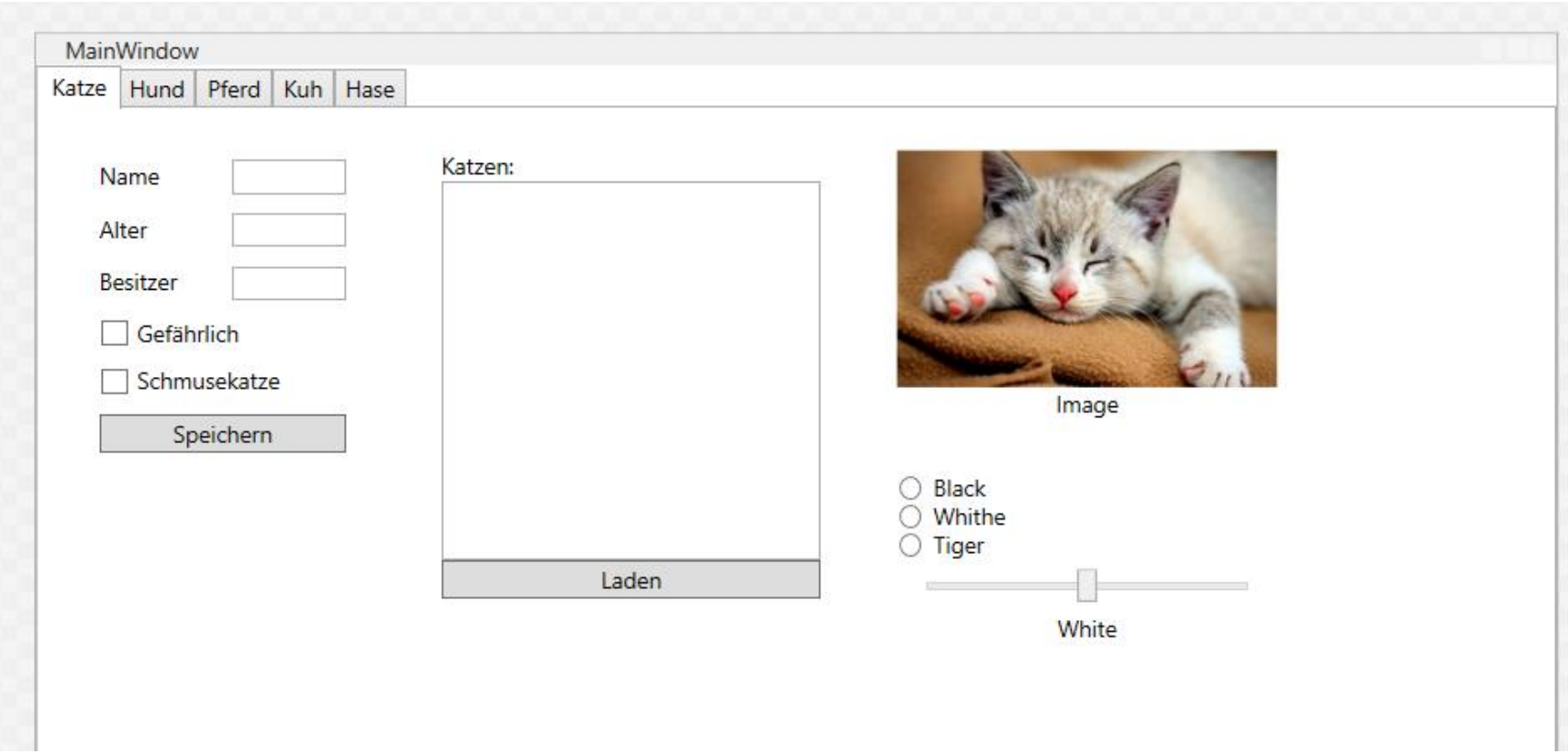






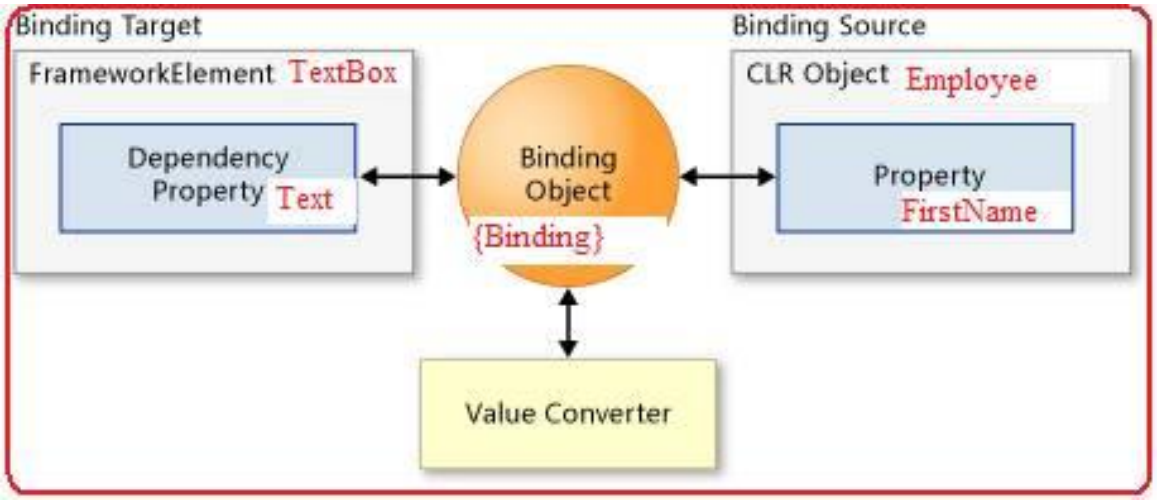
Exercise: Design a view

- Detailed Information: AnimalExample



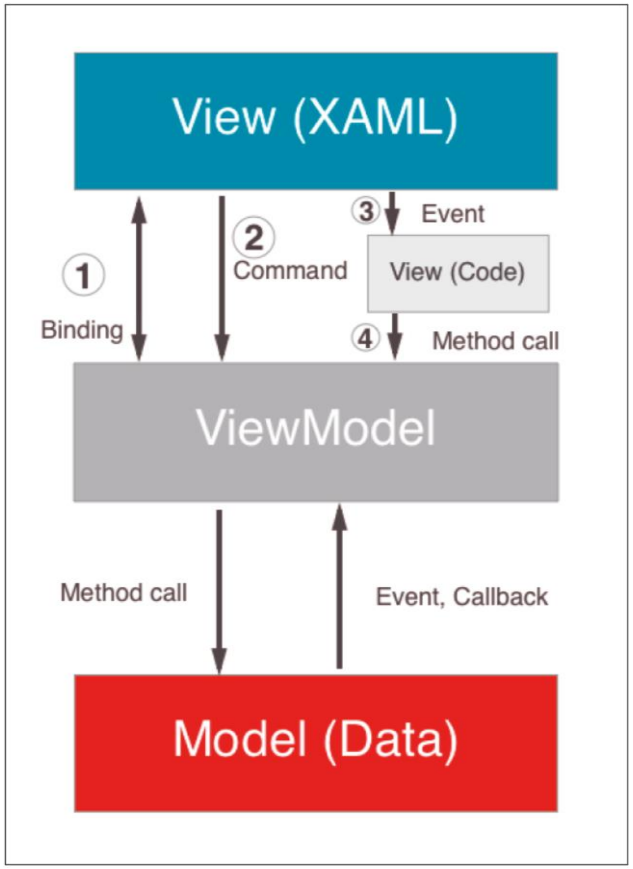
WPF Data Binding

binding a broad range of properties to different kinds of data sources

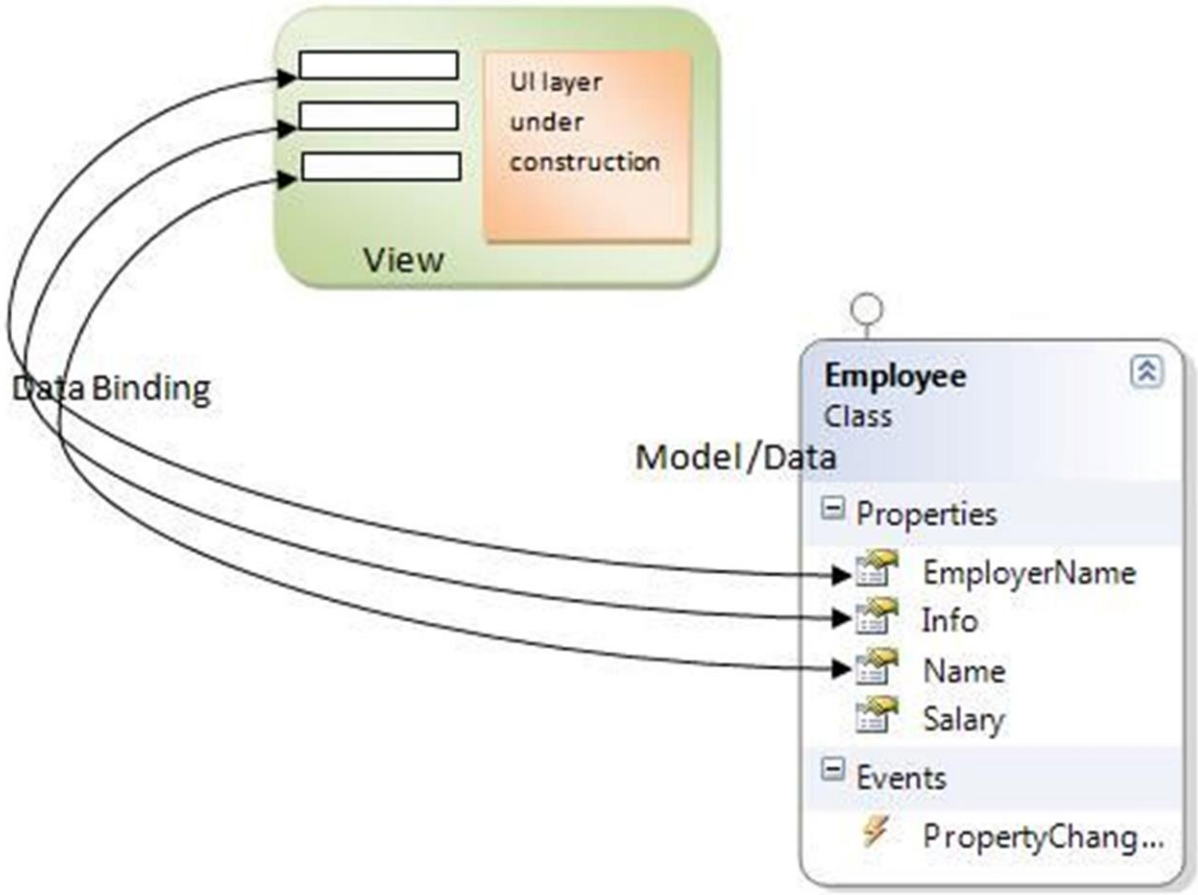


Overview

- WPF Basics
  - New Window
  - MessageBox
- WPF Controls
- WPF Panels
- WPF Data Binding
  - Model View ViewModel (MVVM)
  - Some WPF Controls and there Bindingoptions
  - Binding Modes
  - Value Converter using Resources
- WPF ICommand
  - Relay Command
- WPF Exercises



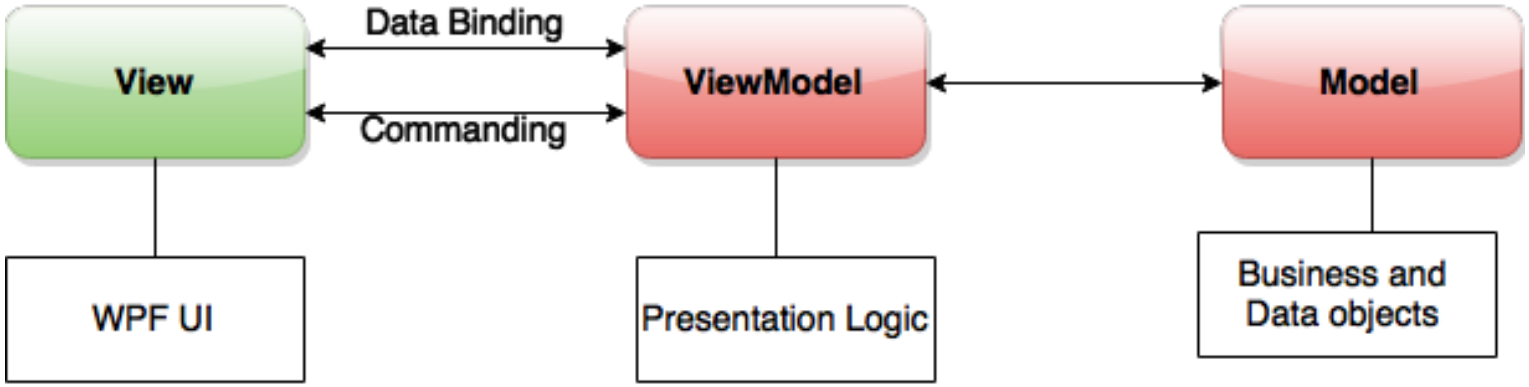
WPF Data Binding



- https://www.wpf-tutorial.com/data-binding/introduction/

Data Binding & MVVM

- organise your code using the 'MVVM' pattern:
  - Model, View, ViewModel



  - aim of ensuring that your View contains minimal (or no) code, and should be XAML-only

WPF & Data Binding

- WPF has two parts
  - XAML which describes your GUI layout and effects code-behind that is tied to the XAML

- Display some data
  'Bind' your XAML to the data
  - <Label Content ="{Binding Name}" />
  - <TextBox Text ="{Binding Name}" />
  - <TextBlock Text ="{Binding Name}" />
  - <Silder Value ="{Binding Age}" />
  - <ProgressBar Value ="{Binding Age}" />
  - <CheckBox IsChecked ="{Binding IsOfFullAge}" />
  - <RadioButton IsChecked ="{Binding IsOfFullAge}" />

Implement INotifyPropertyChanged

- Interface 'INotifyPropertyChanged’
  - Used to communicate any changes in the data between the GUI and your code
- PropertyChangedEventHandler

```
public event PropertyChangedEventHandler PropertyChanged;
```

- [CallerMemberName]

```
void OnPropertyChanged([CallerMemberName] string propertyName = null) {
    if(PropertyChanged!= null)
        PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
}
```

Abstract ViewModel Base-Class

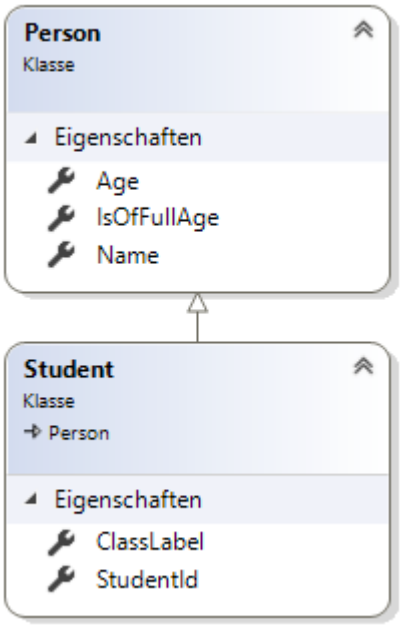
- Implement INotifyPropertyChanged
  - use it as a BaseClass for Concrete ViewModels

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

    protected void CallPropertyChanged
        ([CallerMemberName] string property = null)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new
                PropertyChangedEventArgs(property));
    }
}
```

StudentViewModel

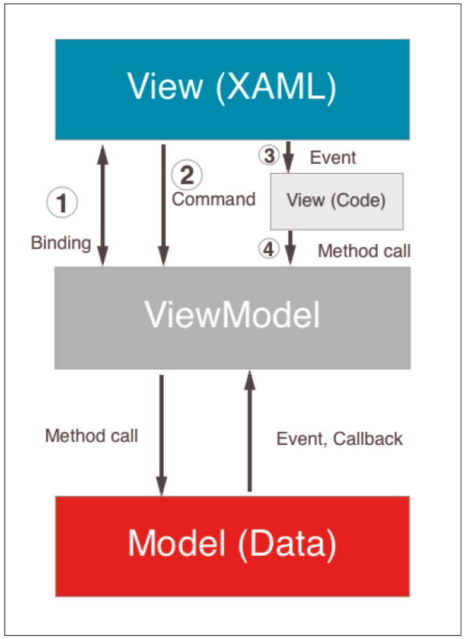
```
class StudentVM : AViewModel
{
    private Student student;
    public StudentVM(Student student)
        { this.student = student; }

    public int StudentId
    {
        get => student.StudentId;
        set
        {
            student.StudentId = value;
            CallPropertyChanged();
        }
    }
    public string Name
    {
        get => student.Name;
        set
        {
            student.Name = value;
            CallPropertyChanged();
        }
    }
}
```

Student Model



Advantages of MVVM

- Lossley coupled architecture:
  - can change one layer without affecting the other layers
- Extensible code:
  - can extends View, ViewModel and the Model layer separately without affecting the other layers



- Testable code:
  - can write unit test cases for both ViewModel and Model layer without referencing the View

MVC vs MVVM

	Windows Forms WF	Windows Presentation Foundation WPF
View	Klasse partial mit zwei .cs Dateien	xaml Datei und cs. Datei
Zwischenschicht	Logikklassen alle Methoden parametrisiert bzw mit Rückgabewerte, die in der View genutzt werden	Properties für Databinding und Commandbinding, welche im XAML verwendet werden
Modelschicht	Modelklassen von Entity Framework	
Persistente Daten	csv, txt, mdf … beliebige Datenbank	

Data Binding on TextBlock

- Add DataContext in Window or Control:

```
<Window.DataContext>
    <local:SkillVM/>
</Window.DataContext>
```

  - Set Binding:

```
<TextBlock Text="{Binding Name}" />
```

  - Set Context and Binding in the Control:

```
<TextBlock DataContext="SkillVM" Text="{Binding Name}" />
```

Data Binding on Label

- DataContext auf UserVM setzen
  `Content="{Binding PropName}"`

```
<Label Content="UserName:"
<Label Content="E-Mail:"
<Label Content="BirthDate:"
```


```
<Label Content="{Binding User.UserName}"  Background="Yellow"
<Label Content="{Binding User.Email}"     Background="Yellow"
<Label Content="{Binding User.BirthDate}" Background="Yellow"
```

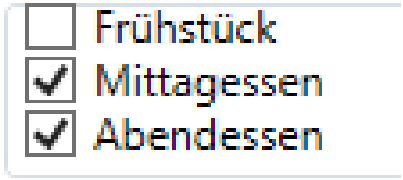
Data Binding on TextBox

- Set DataContext : VM Class
- Text = „{ Binding PropName}“

```
<Window.DataContext>
    <local:PersonalDetail/>
</Window.DataContext>
```

```
<TextBlock TextAlignment="Center" VerticalAlignment="Center">Firstname:</TextBlock>
<TextBox Grid.Column="1" Margin="10" Text="{Binding FirstName}"/>

<TextBlock Grid.Row="1" TextAlignment="Center" VerticalAlignment="Center">Lastname:</TextBlock>
<TextBox Grid.Column="1"  Grid.Row="1"  Margin="10" Text="{Binding LastName}"/>
```

CheckBox Binding



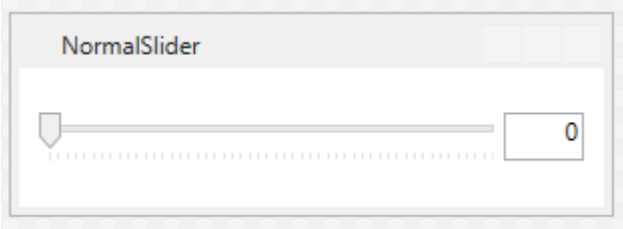
ViewModel

```
private bool dinner;
private bool lunch;
private bool breakfast;

public bool Dinner {
  get => dinner;
  set { dinner = value; OnPropertyChanged(); } }
public bool Lunch {
  get => lunch; set { lunch = value; OnPropertyChanged();  } }
public bool Breakfast {
  get => breakfast;
  set { breakfast = value; OnPropertyChanged(); }}
```

XAML

```
<GroupBox Margin="20,20,20,20">
    <StackPanel >
        <GroupItem>
            <CheckBox IsChecked="{Binding Path=Breakfast}">Frühstück</CheckBox>
        </GroupItem>
        <GroupItem>
            <CheckBox IsChecked="{Binding Path=Lunch}">Mittagessen</CheckBox>
        </GroupItem>
        <GroupItem>
            <CheckBox IsChecked="{Binding Path=Dinner}">Abendessen</CheckBox>
        </GroupItem>
    </StackPanel>
</GroupBox>
```

Slider



- Simple Slider

```
<TextBox Text="{Binding ElementName=slider, Path=Value, UpdateSourceTrigger=PropertyChanged}"
         DockPanel.Dock="Right" TextAlignment="Right" Width="40" />
<Slider Maximum="255" TickPlacement="BottomRight"
        TickFrequency="5" IsSnapToTickEnabled="True" Name="slider" />
```

- Slider Binding to a Color

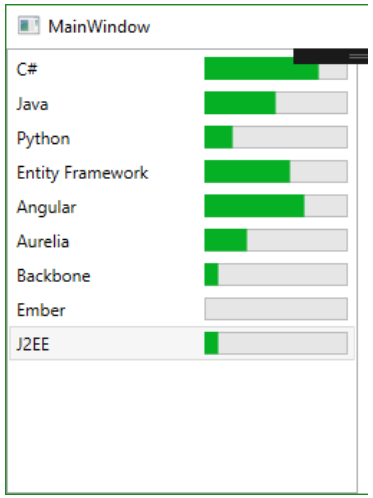
```
<Slider Maximum="255"
        TickPlacement="BottomRight"
        TickFrequency="5"
        Value="{Binding RedValue,Mode=TwoWay}"
        IsSnapToTickEnabled="True"
        x:Name="slider_red"
        ValueChanged="ColorSlider_ValueChanged" />
```

https://www.wpf-tutorial.com/misc-controls/the-slider-control/

ProgressBar

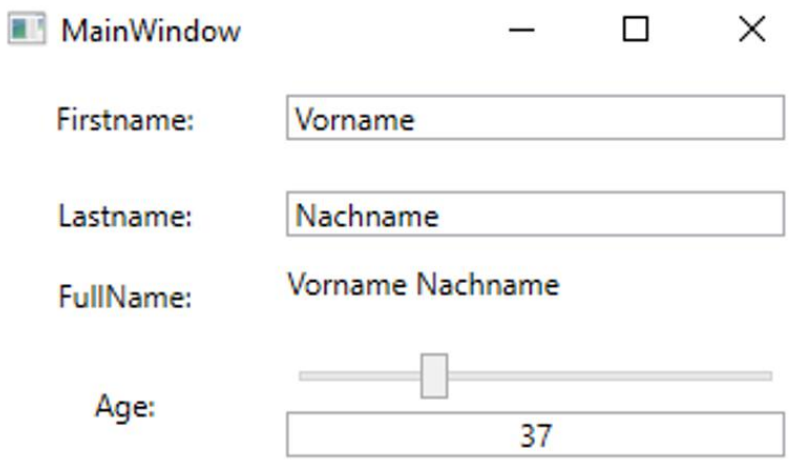
- Minimum, Maximum & Value with Binding

```
<TextBlock Text="{Binding Name}" />

<ProgressBar Grid.Column="1"
             Minimum="0" Maximum="100"
             Value="{Binding Percent}" />
```



https://www.wpf-tutorial.com/misc-controls/the-progressbar-control/

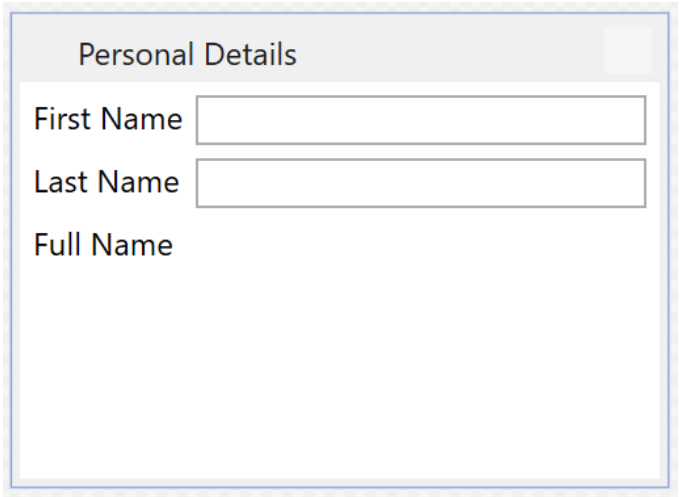
http://www.blackwasp.co.uk/StatusBar.aspx



Personal Details

Personal Detail

- Create TextBlock and TextBox
  - to Enter FirstName and Last Name
  - show FullName in a TextBlock below




```
<TextBlock>First Name</TextBlock>
<TextBox Grid.Column="1" Margin="5 0 0 5" Text="{Binding FirstName}"/>

<TextBlock Grid.Row="1">Last Name</TextBlock>
<TextBox Grid.Column="1" Grid.Row="1" Margin="5 0 0 5" Text="{Binding LastName}"/>

<TextBlock Grid.Row="2">Full Name</TextBlock>
<TextBlock Grid.Column="1" Grid.Row="2" Margin="5 0 0 5" Text="{Binding FullName}"/>
```

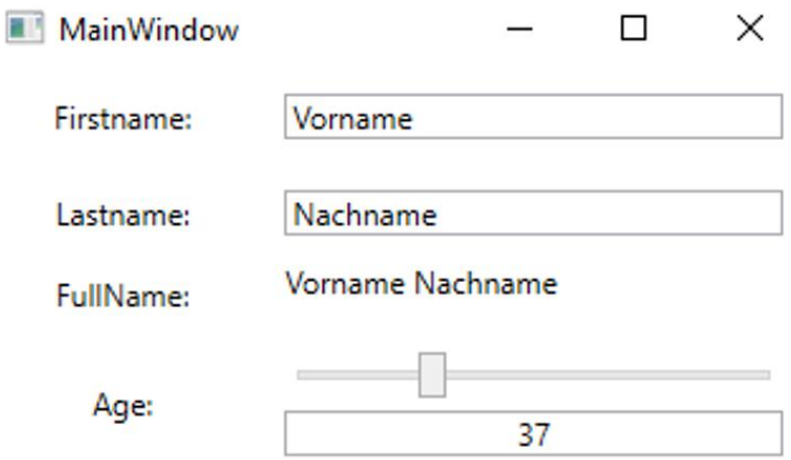
XAML – GUI Design

```
<Window x:Class="WPF_DataCommandBinding.WindowPerson"          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:WPF_DataCommandBinding"      mc:Ignorable="d"
Title="Personal Details"          Height="180"          Width="250"          ResizeMode="NoResize">
    <Grid Margin="5">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
             <RowDefinition Height="Auto"/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <TextBlock>First Name</TextBlock>
        <TextBox Grid.Column="1" Margin="5 0 0 5"/>
        <TextBlock Grid.Row="1">Last Name</TextBlock>
        <TextBox Grid.Column="1" Grid.Row="1" Margin="5 0 0 5"/>
        <TextBlock Grid.Row="2">Age</TextBlock>
        <StackPanel Grid.Column="1" Grid.Row="2" Margin="5 0 0 5">
            <Slider Minimum="16" Maximum="120" />
            <TextBlock Text="16" HorizontalAlignment="Center"/>
        </StackPanel>
        <Button Grid.Column="1" Grid.Row="3" HorizontalAlignment="Right">Close Window</Button>
    </Grid>
</Window>
```

Copy and add the DataBinding

Data Binding

- Add the Age Slider & TextBlock



Firstly, replace the XAML for the first TextBox with the code below:

```
<TextBox Grid.Column="1" Margin="5 0 0 5" Text="{Binding FirstName}"/>
```

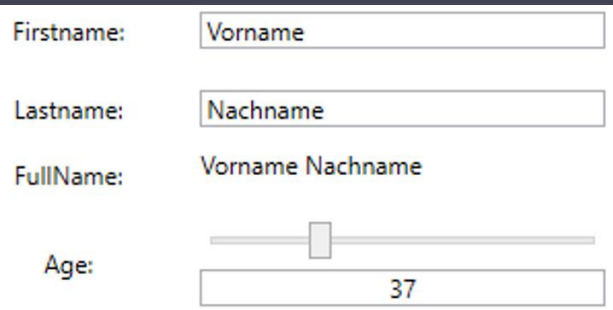
For the second TextBox use:

```
<TextBox Grid.Column="1" Grid.Row="1" Margin="5 0 0 5" Text="{Binding LastName}"/>
```

Finally, remove the explicit Paths from the Slider and TextBlock that show the age.

```
<Slider Minimum="16" Maximum="120" Value="{Binding Age}" />
<TextBlock HorizontalAlignment="Center" Text="{Binding Age}"/>
```

Personal Detail




```
public class PersonalDetailVM : INotifyPropertyChanged {
    private string firstname;
    private string lastname;
    private int age;

    public string FirstName {
        get { return firstname; }
        set {
            firstname = value;
            OnPropertyChanged("FirstName");
            OnPropertyChanged("FullName");
        }
    }
    public string LastName {
        get { return lastname; }
        set {
            lastname = value;
            OnPropertyChanged("LastName");
            OnPropertyChanged("FullName");
        }
    }
    public string FullName {
        get { return string.Format("{0} {1}", FirstName, LastName); }
    }
```

```
public int Age {
    get { return age; }
    set {
        age = value;
        OnPropertyChanged("Age");
    }
}
public event PropertyChangedEventHandler PropertyChanged;
protected virtual void OnPropertyChanged(string property) {
    if (PropertyChanged != null) {
        PropertyChanged(this, new PropertyChangedEventArgs(property));
    }
}
```

```
using System.ComponentModel;
```

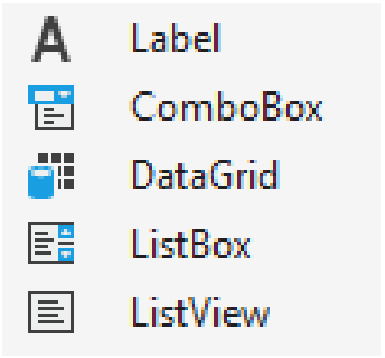
- Write a VM-Class
- Implement INotifyProperty Changed, raise PropertyChanged Event

Personal Detail

- Set DataContext
- XAML: Add Bindings to each Control

```
    Title="Personal Detail" Height="213" Width="341">

<Window.DataContext>
    <local:PersonalDetailVM/>
</Window.DataContext>

<Grid>
```

```
<TextBlock TextAlignment="Center" VerticalAlignment="Center">Firstname:</TextBlock>
<TextBox Grid.Column="1" Margin="10" Text="{Binding FirstName}"/>

<TextBlock Grid.Row="1" TextAlignment="Center" VerticalAlignment="Center">Lastname:</TextBlock>
<TextBox Grid.Column="1"  Grid.Row="1"  Margin="10" Text="{Binding LastName}"/>

<TextBlock Grid.Row="2" TextAlignment="Center" VerticalAlignment="Center">FullName:</TextBlock>
<TextBlock Grid.Column="1" Grid.Row="2" Margin="10 0 0 10" Text="{Binding FullName}"/>

<TextBlock Grid.Row="3" Grid.Column="0" TextAlignment="Center" VerticalAlignment="Center">Age:</TextBlock>
<StackPanel Grid.Row="3" Grid.Column="1" >
    <Slider Margin="10 10 10 0" Maximum="130" Value="{Binding Age}"></Slider>
    <TextBox Margin="10 5 10 10" Text="{Binding Age}" TextAlignment="Center"/>
</StackPanel>
```



List Controls

ComboBox

ListBox

ListView

DataGrid

bind to a ObservableCollection

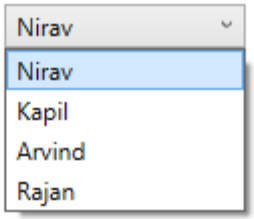
ObservableCollection

- Getting informed when collection changes
- use ObservableCollection<>
  - not a List or Dictionary
  - WPF window needs to be able to 'observe' your data
  - WPF controls (including 'Window's) have a 'DataContext’
  - Collection controls have an 'ItemsSource' attribute to bind to
  - Collection controls have an ‘SelectedItem' attribute to bind to

Binding ComboBox

- ViewModel – Personlist & SelectedPerson

```
private ObservableCollection<Person> _persons;

public ObservableCollection<Person> Persons
{
    get { return _persons; }
    set { _persons = value; }
}
private Person _sperson;

public Person SPerson
{
    get { return _sperson; }
    set { _sperson = value; }
}
```



- Binding XAML

```
<ComboBox HorizontalAlignment="Left"
          Margin="183,39,0,0"
          VerticalAlignment="Top"
          Width="120"
          ItemsSource="{Binding Path=Persons}"
          SelectedItem="{Binding Path=SPerson}"
          DisplayMemberPath="Name"/>
```

DataContext=PersonVM

Binding Properties

- ItemsSource
  - Sets a collection used to generate the content
- SelectedItem
  - to bind to an instance of a selected object
    - when the SelectedItem is changed,
      all other entities that are bound to it are also updated

MainWindow

Employees

MemberID	Name	Department	Phone	Email	Salary
1	John Hancock	IT	31234743	John.Hancock@Company.com	3450.44
2	Jane Hayes	Sales	31234744	Jane.Hayes@Company.com	3700
3	Larry Jones	Marketing	31234745	Larry.Jones@Company.com	3000
4	Patricia Palce	Secretary	31234746	Patricia.Palce@Company.com	2900
5	Jean L. Trickard	Director	31234747	Jean.L.Tricard@Company.com	5400

```
public class Employee
{
    public int MemberID { get; set; }
    public string Name { get; set; }
    public string Department { get; set; }
    public string Phone { get; set; }
    public string Email { get; set; }
    public string Salary { get; set; }
    public override string ToString() {
        return $"{Name};{Department};{Phone};{Email};{Salary};";
    }
}
```

Employee

ListControls & ObservableCollection

Selected Item

EmployeeVM & Employee

```
public class EmployeeVM : INotifyPropertyChanged {
    private ObservableCollection<Employee> _employees;
    private Employee _selectedEmployee;

    public ObservableCollection<Employee> Employees {
        get {
            if (_employees == null) {
                _employees = InitializeEmployees();
            }
            return _employees;
        }
        set {
            _employees = value;
            OnPropertyChanged("Employees");
        }
    }
    public Employee SelectedEmployee {
        get {
            if (_selectedEmployee == null)
                _selectedEmployee = _employees[0];
            return _selectedEmployee;
        }
        set {
            _selectedEmployee = value;
            OnPropertyChanged("SelectedEmployee");
        }
    }
    public event PropertyChangedEventHandler PropertyChanged;
    private void OnPropertyChanged([CallerMemberName] string property = null) ...


    private ObservableCollection<Employee> InitializeEmployees() ...
}
```

```
public class Employee
{
    public int MemberID { get; set; }
    public string Name { get; set; }
    public string Department { get; set; }
    public string Phone { get; set; }
    public string Email { get; set; }
    public string Salary { get; set; }
    public override string ToString() {
        return $"{Name};{Department};{Phone};{Email};{Salary};";
    }
}
```

Copy XAML View

```
<Window …
        Title="MainWindow" Height="450" Width="800">

    <Window.DataContext>
        <local:EmployeeVM/>
    </Window.DataContext>

    <Grid Margin="5">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition/>
            <RowDefinition/>
        </Grid.RowDefinitions>

        <TextBlock Text="Employees" FontSize="22" FontWeight="Bold"/>
        <DataGrid ItemsSource="{…}"
                  SelectedItem="{…}" Grid.Row="1"></DataGrid>

        <StackPanel Margin="10" Grid.Row="2">
            <TextBlock Text="{…}"></TextBlock>
            <TextBlock Text="{…}"></TextBlock>
            <TextBlock Text="{…}"></TextBlock>
            <TextBlock Text="{…}"></TextBlock>
            <TextBlock Text="{…}"></TextBlock>
            <TextBlock Text="{…}"></TextBlock>
        </StackPanel>
    </Grid>
</Window>
```

MemberID	Name	Department	Phone	Email	Salary
1	John Hancock	IT	31234743	John.Hancock@Company.com	3450.44
2	Jane Hayes	Sales	31234744	Jane.Hayes@Company.com	3700
3	Larry Jones	Marketing	31234745	Larry.Jones@Company.com	3000
4	Patricia Palce	Secretary	31234746	Patricia.Palce@Company.com	2900
5	Jean L. Trickard	Director	31234747	Jean.L.Tricard@Company.com	5400

DataGrid

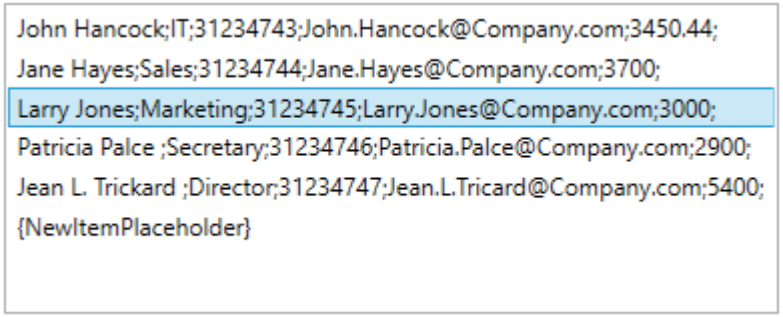
https://www.wpf-tutorial.com/datagrid-control/introduction/

DataGrid Binding

Selected Employee DataGrid:
4
Patricia Palce
Secretary
Patricia.Palce@Company.com
31234746
2900

MemberID	Name	Department	Phone	Email	Salary
1	John Hancock	IT	31234743	John.Hancock@Company.com	3450.44
2	Jane Hayes	Sales	31234744	Jane.Hayes@Company.com	3700
3	Larry Jones	Marketing	31234745	Larry.Jones@Company.com	3000
4	Patricia Palce	Secretary	31234746	Patricia.Palce@Company.com	2900
5	Jean L. Trickard	Director	31234747	Jean.L.Tricard@Company.com	5400

```
<StackPanel Margin="10" Grid.Row="2">
    <TextBlock Text="Selected Employee" FontSize="15" FontWeight="Bold"/>
    <TextBlock Text="DataGrid:" FontSize="15" FontWeight="Bold"/>
    <TextBlock Text="{Binding SelectedEmployee.MemberID}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Name}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Department}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Email}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Phone}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Salary}"></TextBlock>
</StackPanel>
<DataGrid ItemsSource="{Binding Employees}" SelectedItem="{Binding SelectedEmployee}" Width="520"/>
```



ListBox

Add a ListBox to the View
It uses the ToString-Method to represent an item of the list

https://www.wpf-tutorial.com/list-controls/listbox-control/

https://wpf-tutorial.com/de/72/listen-steuerelemente/die-listbox/

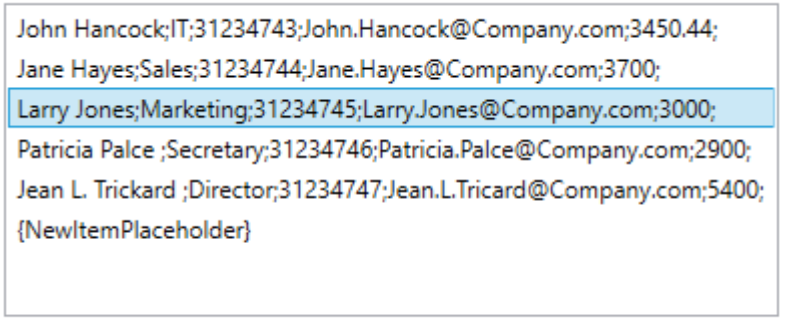
https://www.c-sharpcorner.com/UploadFile/mahesh/listbox-in-wpf/

ListBox Binding

Selected Employee ListBox:
3
Larry Jones
Marketing
Larry.Jones@Company.com
31234745
3000

John Hancock;IT;31234743;John.Hancock@Company.com;3450.44;
Jane Hayes;Sales;31234744;Jane.Hayes@Company.com;3700;
Larry Jones;Marketing;31234745;Larry.Jones@Company.com;3000;
Patricia Palce ;Secretary;31234746;Patricia.Palce@Company.com;2900;
Jean L. Trickard ;Director;31234747;Jean.L.Tricard@Company.com;5400;
{NewItemPlaceholder}

```
<StackPanel Margin="10" Grid.Row="2">
    <TextBlock Text="Selected Employee" FontSize="15" FontWeight="Bold"/>
    <TextBlock Text="ListBox:" FontSize="15" FontWeight="Bold"/>
    <TextBlock Text="{Binding SelectedEmployee.MemberID}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Name}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Department}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Email}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Phone}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Salary}"></TextBlock>
</StackPanel>

<ListBox ItemsSource="{Binding Employees}"
     SelectedItem="{Binding SelectedEmployee}"></ListBox>
```



ListView

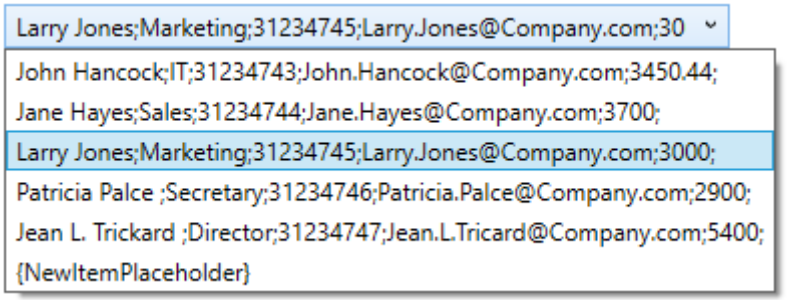
Add a ListView to the View
It uses the ToString-Method to represent an item of the list

ListView Binding

Selected Employee
ListView:
3
Larry Jones
Marketing
Larry.Jones@Company.com
31234745
3000

John Hancock;IT;31234743;John.Hancock@Company.com;3450.44;
Jane Hayes;Sales;31234744;Jane.Hayes@Company.com;3700;
Larry Jones;Marketing;31234745;Larry.Jones@Company.com;3000;
Patricia Palce ;Secretary;31234746;Patricia.Palce@Company.com;2900;
Jean L. Trickard ;Director;31234747;Jean.L.Tricard@Company.com;5400;
{NewItemPlaceholder}

```
<StackPanel Margin="10" Grid.Row="2">
    <TextBlock Text="Selected Employee" FontSize="15" FontWeight="Bold"/>
    <TextBlock Text="ListView:" FontSize="15" FontWeight="Bold"/>
    <TextBlock Text="{Binding SelectedEmployee.MemberID}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Name}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Department}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Email}"></TextBlock>
    <TextBlock Text="{Binding Selected
    <TextBlock Text="{Binding Selected
</StackPanel>
<ListView ItemsSource="{Binding Employees}"
      SelectedItem="{Binding SelectedEmployee}"></ListView>
```



ComboBox

Add a ComboBox to the View

like the ListBox control, but takes up a lot less space, because the list of items is hidden when not needed

https://wpf-tutorial.com/list-controls/combobox-control/

ComboBox Binding

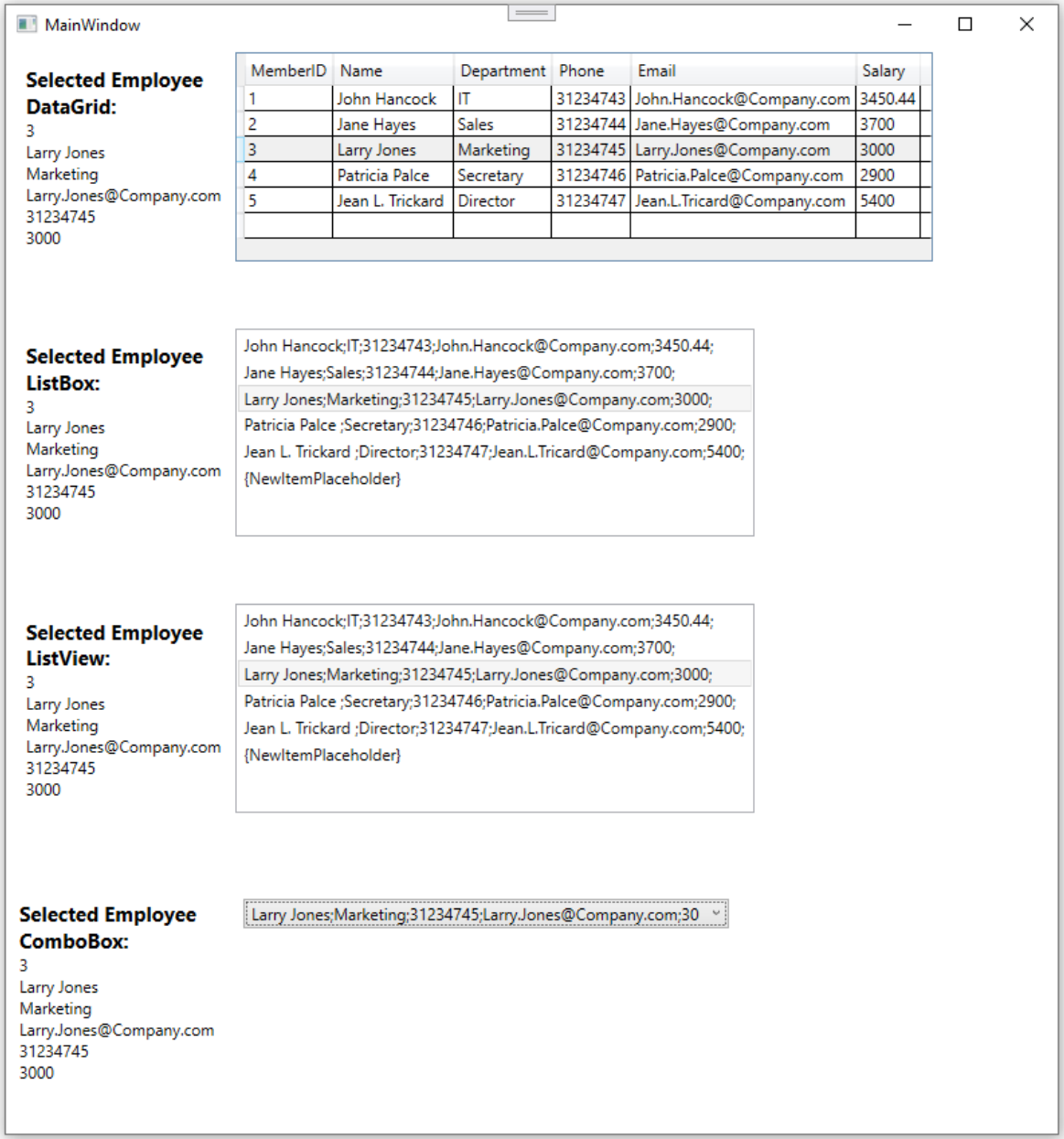

```
<StackPanel Margin="10" Grid.Row="2">
    <TextBlock Text="Selected Employee" FontSize="15" FontWeight="Bold"/>
    <TextBlock Text="ComboBox:" FontSize="15" FontWeight="Bold"/>
    <TextBlock Text="{Binding SelectedEmployee.MemberID}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Name}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Department}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Email}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Phone}"></TextBlock>
    <TextBlock Text="{Binding SelectedEmployee.Salary}"></TextBlock>
</StackPanel>

<ComboBox ItemsSource="{Binding Employees}" SelectedItem="{Binding SelectedEmployee}"
      HorizontalAlignment="Left" Margin="177,9,0,0" VerticalAlignment="Top" Width="362"/>
```

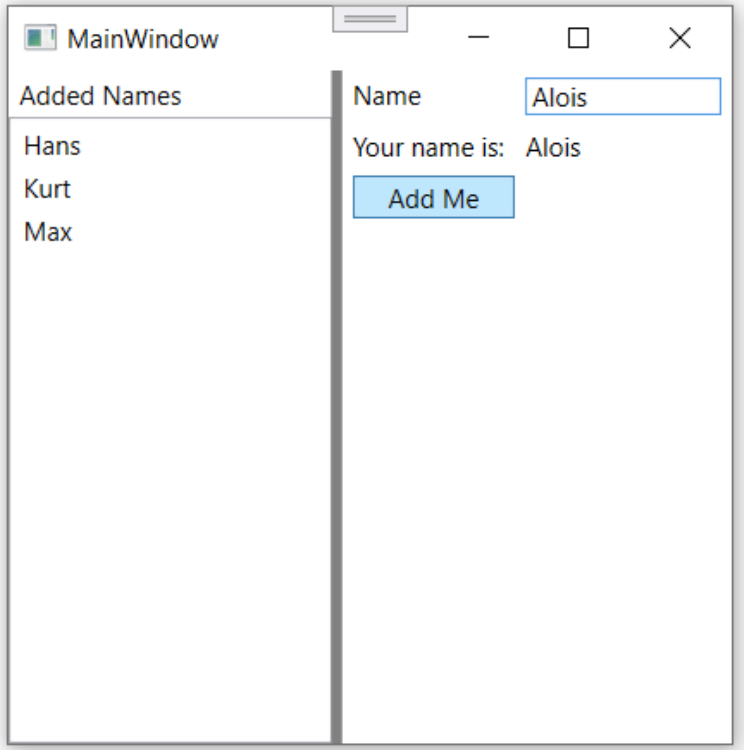
Copy Initialize

```
private ObservableCollection<Employee> InitializeEmployees() {
            ObservableCollection<Employee> employees;
            employees = new ObservableCollection<Employee>();
            employees.Add(new Employee {
                MemberID = 1,
                Name = "John Hancock",
                Department = "IT",
                Phone = "31234743",
                Email = @"John.Hancock@Company.com",
                Salary = "3450.44"
            });
            employees.Add(new Employee {
                MemberID = 2,
                Name = "Jane Hayes",
                Department = "Sales",
                Phone = "31234744",
                Email = @"Jane.Hayes@Company.com",
                Salary = "3700"
            });
            employees.Add(new Employee {
                MemberID = 3,
                Name = "Larry Jones",
                Department = "Marketing",
                Phone = "31234745",
                Email = @"Larry.Jones@Company.com",
                Salary = "3000"
            });
            employees.Add(new Employee {
                MemberID = 4,
                Name = "Patricia Palce ",
                Department = "Secretary",
                Phone = "31234746",
                Email = @"Patricia.Palce@Company.com",
                Salary = "2900"
            });
            employees.Add(new Employee {
                MemberID = 5,
                Name = "Jean L. Trickard ",
                Department = "Director",
                Phone = "31234747",
                Email = @"Jean.L.Tricard@Company.com",
                Salary = "5400"
            });
            return employees;
        }
```

1	John Hancock	IT	31234743	John.Hancock@Company.com	3450.44
2	Jane Hayes	Sales	31234744	Jane.Hayes@Company.com	3700
3	Larry Jones	Marketing	31234745	Larry.Jones@Company.com	3000
4	Patricia Palce	Secretary	31234746	Patricia.Palce@Company.com	2900
5	Jean L. Trickard	Director	31234747	Jean.L.Tricard@Company.com	5400

All together now





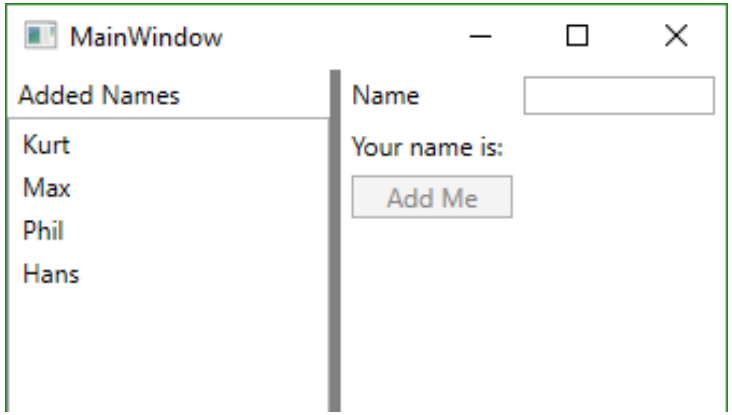
Simple List of Names

Create an inputfield, bind another label to that control. Create an <AddMe> Button. Write a Command, which adds the Name to the ListControl.

Use an Observable Collection and bind it to the ListControl

Simple Window

- Create a WPF Application,
- add some names to a list and show it.



XAML

```
<Window …
        Title="MainWindow" Height="350" Width="350">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="150"/>
            <ColumnDefinition Width="5"/>
            <ColumnDefinition Width="*"/>
        </Grid.ColumnDefinitions>
        <DockPanel>
            <TextBlock Text="Added Names" DockPanel.Dock="Top" Margin="5,3"/>
            <ListBox ItemsSource="{…}">
            </ListBox>
        </DockPanel>
        <GridSplitter Grid.Column="1" VerticalAlignment="Stretch"
                      Width="5" Background="Gray" HorizontalAlignment="Left" />
        <Grid Grid.Column="2">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto"/>
                <ColumnDefinition Width="*"/>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
            </Grid.RowDefinitions>
            <TextBlock Grid.Row="0" Text="Name" Margin="5,3"/>
            <TextBox Grid.Row="0" Grid.Column="1"
                     Text="{Binding …}" Margin="5,3"/>
            <TextBlock Grid.Row="1" Text="Your name is:" Margin="5,3"/>
            <TextBlock Grid.Row="1" Grid.Column="1" Text="{…}" Margin="5,3"/>

            <Button Grid.Row="2" Grid.ColumnSpan="2" HorizontalAlignment="Left„
                    Content="Add Me" Margin="5,3" MinWidth="75" Command="{…}" />
        </Grid>
    </Grid>
</Window>
```

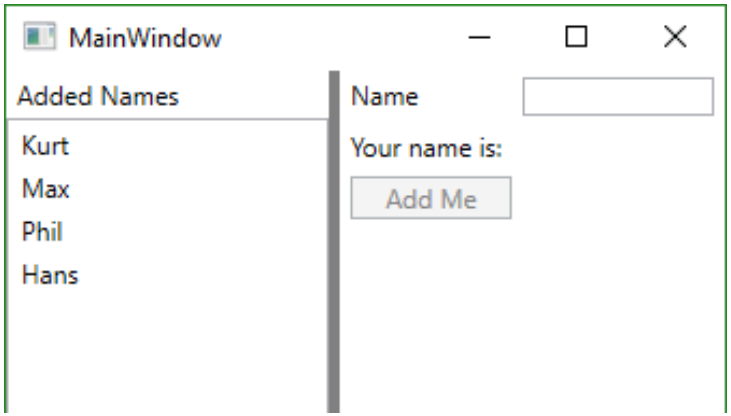
DataContext can be set in the Constructor or in XAML

```
public partial class SimpleWindow : Window
{
    public SimpleWindow()
    {
        InitializeComponent();
        DataContext = new SimpleNamesListVM();
        this.Show();
    }
}
```

ViewModel

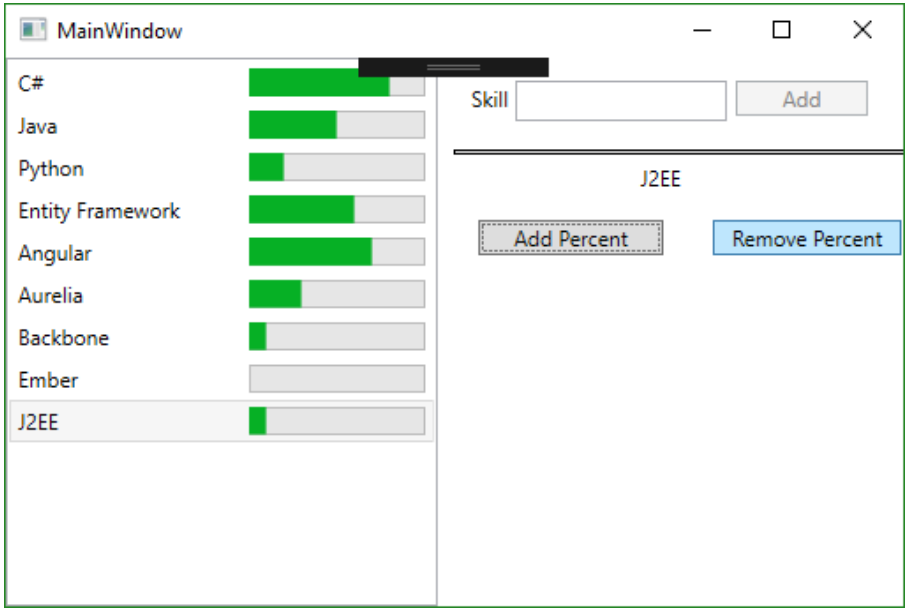
```
public ICommand AddCommand { get; private set; }
public SimpleNamesListVM() {
    AddCommand = new AddNameCommand(this);
}
```

```
public class SimpleNamesListVM : INotifyPropertyChanged {

    string mCurrentName;
    public string CurrentName {
        get { return mCurrentName; }
        set {
            if (value == mCurrentName)
                return;
            mCurrentName = value;
            OnPropertyChanged();
        }
    }
    public ICommand AddCommand { get; private set; }
    public SimpleNamesListVM() {
        AddCommand = new AddNameCommand(this);
    }

    public ObservableCollection<string> AddedNames { get; } = new ObservableCollection<string>();
    public event PropertyChangedEventHandler PropertyChanged;
    void OnPropertyChanged([CallerMemberName] string propertyName = null) {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```



Add Command

```
class AddNameCommand : ICommand {
    SimpleNamesListVM parent;
    public AddNameCommand(SimpleNamesListVM parent) {
        this.parent = parent;
    }

    public bool CanExecute(object parameter) {
        return !string.IsNullOrEmpty(parent.CurrentName);
    }
    public void Execute(object parameter) {
        parent.AddedNames.Add(parent.CurrentName); ;
        parent.CurrentName = null;
    }
    public event EventHandler CanExecuteChanged {
        add => CommandManager.RequerySuggested += value;
        remove => CommandManager.RequerySuggested -= value;
    }
}
```



Skill Manager

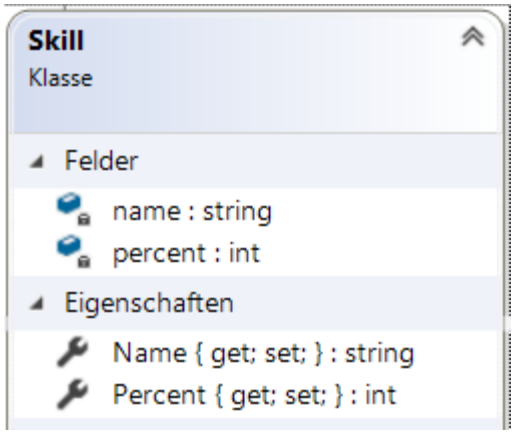
ListBox & Progressbar

Add a Skill to the SkillList (with 0% knowledge)

Increase or Decrease (10%) the percent

Skill & SkillViewModel

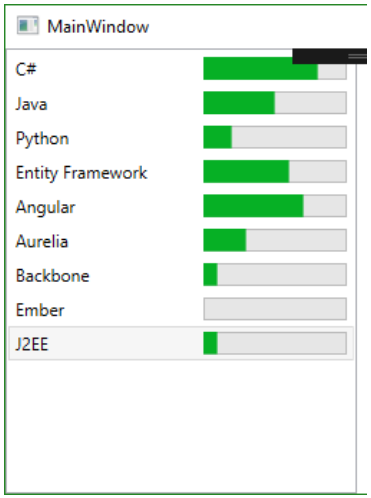
```
private Skill skill;
public Skill Skill...
private Skill selectedSkill=null;
public Skill SelectedSkill...
public ObservableCollection<Skill> Skills ...
public SkillVM()...
public SkillVM(List<Skill> skills)...

public void AddSkills(List<Skill>skills)...
public ICommand AddSkillCommand...
private void AddSkill()...
public ICommand AddPercentageCommand...
private void AddPercentage()...
public ICommand RemovePercentageCommand...
private void RemovePercentage()...
```



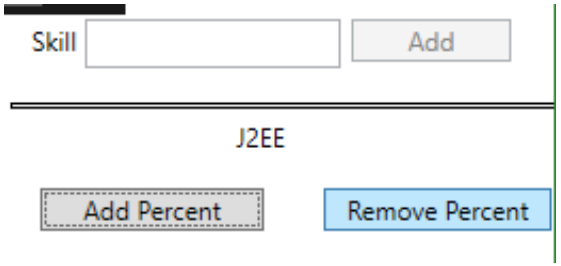

```
<Window.DataContext>
    <local:SkillVM/>
</Window.DataContext>
```

Binding a ListBox to a SkillList

```
<Window.DataContext>
    <local:SkillVM/>
</Window.DataContext>
<Grid>
    <ListBox Name="lb_skills"  HorizontalContentAlignment="Stretch"
             Margin="0,0,264.4,-0.2"
             ItemsSource="{Binding Skills,UpdateSourceTrigger=PropertyChanged}"
             SelectedItem="{Binding SelectedSkill,Mode=TwoWay}" >
        <ListBox.ItemTemplate>
            <DataTemplate>
                <Grid Margin="0,2">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="100" />
                    </Grid.ColumnDefinitions>
                    <TextBlock Text="{Binding Name}" />
                    <ProgressBar Grid.Column="1" Minimum="0" Maximum="100" Value="{Binding Percent}" />
                </Grid>
            </DataTemplate>
        </ListBox.ItemTemplate>
    </ListBox>
```



Binding: Label, TextBox & Button



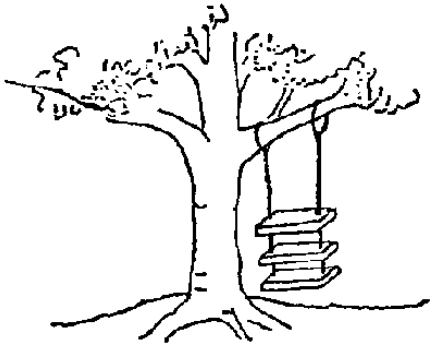

```
<Label Content="Skill" HorizontalAlignment="Left" Margin="259,10,0,0" VerticalAlignment="Top"/>
<TextBox HorizontalAlignment="Left" Height="23" Margin="289,13,0,0" TextWrapping="Wrap"
         Text="{Binding Skill.Name,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"
         VerticalAlignment="Top" Width="120"/>
<Button Content="Add" HorizontalAlignment="Left" Margin="414,13,0,0"
        VerticalAlignment="Top" Width="75"
        Command="{Binding AddSkillCommand}"/>
<Rectangle Fill="#FFF4F4F5" HorizontalAlignment="Left" Height="3" Margin="254,52,0,0"
           Stroke="Black" VerticalAlignment="Top" Width="264"/>
<TextBlock HorizontalAlignment="Left" Margin="360,60,0,0" TextWrapping="Wrap"
           Text="{Binding SelectedSkill.Name}" VerticalAlignment="Top"/>
<Button Content="Add Percent" HorizontalAlignment="Left" Margin="268,92,0,0"
        VerticalAlignment="Top" Width="105"
        Command="{Binding AddPercentageCommand}"/>
<Button Content="Remove Percent" HorizontalAlignment="Left" Margin="401,92,0,0"
        VerticalAlignment="Top" Width="107"
        Command="{Binding RemovePercentageCommand}" />
```

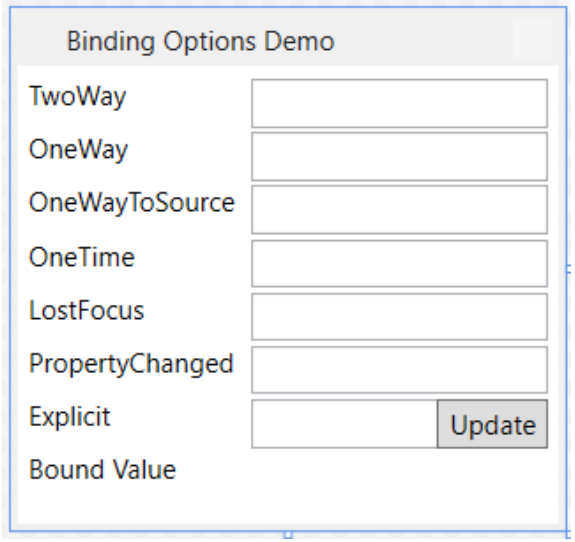
Add Data…

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        List<Skill> skills = new List<Skill>()
        {
            new Skill() {Name = "C#", Percent = 80},
            new Skill() {Name = "Java", Percent = 50},
            new Skill() {Name = "Python", Percent = 20},
            new Skill() {Name = "Entity Framework", Percent = 60},
            new Skill() {Name = "Angular", Percent = 70},
            new Skill() {Name = "Aurelia", Percent = 30},
            new Skill() {Name = "Backbone", Percent = 10},
            new Skill() {Name = "Ember", Percent = 0},
        };
        var skillVm = DataContext as SkillVM;
        skillVm.AddSkills(skills);
    }
}
```

Binding Modes

OneWay, OneWayToSource, TwoWay, OneTime





Hello Bound World

Try the diffrent ways to UpdateSourceTrigger:

http://www.blackwasp.co.uk/WPFBindingOptions.aspx

Create UI

```
<Window …
        Title="Binding Options Demo"
        Height="235"
        Width="250"
        ResizeMode="NoResize">
    <Grid Margin="5">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="100"/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="24"/>
            <RowDefinition Height="24"/>
            <RowDefinition Height="24"/>
            <RowDefinition Height="24"/>
            <RowDefinition Height="24"/>
            <RowDefinition Height="24"/>
            <RowDefinition Height="24"/>
            <RowDefinition/>
        </Grid.RowDefinitions>

            <TextBlock>TwoWay</TextBlock>
            <TextBox Grid.Column="1" Height="22"/>

            <TextBlock Grid.Row="1">OneWay</TextBlock>
            <TextBox Grid.Column="1" Grid.Row="1" Height="22"/>

            <TextBlock Grid.Row="2">OneWayToSource</TextBlock>
            <TextBox Grid.Column="1" Grid.Row="2" Height="22"/>

            <TextBlock Grid.Row="3">OneTime</TextBlock>
            <TextBox Grid.Column="1" Grid.Row="3" Height="22"/>

            <TextBlock Grid.Row="4">LostFocus</TextBlock>
            <TextBox Grid.Column="1" Grid.Row="4" Height="22"/>

            <TextBlock Grid.Row="5">PropertyChanged</TextBlock>
            <TextBox Grid.Column="1" Grid.Row="5" Height="22"/>

            <TextBlock Grid.Row="6">Explicit</TextBlock>
            <TextBox Name="Explicit" Grid.Column="1" Grid.Row="6"
                     Height="22" Margin="0 0 49 0"/>
            <Button Grid.Column="1" Grid.Row="6"
                    Height="22" Width="50" HorizontalAlignment="Right"
                    Content="Update"/>

            <TextBlock Grid.Row="7">Bound Value</TextBlock>
            <TextBlock Grid.Column="1" Grid.Row="7" Text="{Binding Text}"/>
    </Grid>
</Window>
```

TestObject

```
public class TestObject : INotifyPropertyChanged {
    string _text = "Hello, world";

    public string Text {
        get { return _text; }
        set {
            _text = value;
            OnPropertyChanged("Text");
        }
    }
    private void OnPropertyChanged(string propertyName) {
        if (PropertyChanged != null)
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }

    public event PropertyChangedEventHandler PropertyChanged;
}
```

Add Bindings

```
<TextBlock>TwoWay</TextBlock>
<TextBox Grid.Column="1" Height="22" Text="{Binding Text, Mode=TwoWay}"/>

<TextBlock Grid.Row="1">OneWay</TextBlock>
<TextBox Grid.Column="1" Grid.Row="1" Height="22" Text="{Binding Text, Mode=OneWay}"/>

<TextBlock Grid.Row="2">OneWayToSource</TextBlock>
<TextBox Grid.Column="1" Grid.Row="2" Height="22" Text="{Binding Text, Mode=OneWayToSource}"/>

<TextBlock Grid.Row="3">OneTime</TextBlock>
<TextBox Grid.Column="1" Grid.Row="3" Height="22" Text="{Binding Text, Mode=OneTime}"/>

<TextBlock Grid.Row="4">LostFocus</TextBlock>
<TextBox Grid.Column="1" Grid.Row="4" Height="22"
        Text="{Binding Text, Mode=TwoWay,UpdateSourceTrigger=LostFocus}"/>

<TextBlock Grid.Row="5">PropertyChanged</TextBlock>
<TextBox Grid.Column="1" Grid.Row="5" Height="22"
        Text="{Binding Text,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"/>
```

HelloBoundWorldWindow

- Code Behind with Button_Click

```
/// <summary>
/// Interaktionslogik für HelloBoundWorldWindow.xaml
/// </summary>
public partial class HelloBoundWorldWindow : Window
{
    public HelloBoundWorldWindow()
    {
        InitializeComponent();
        this.Show();
        DataContext = new TestObject();
    }

    private void Button_Click(object sender, RoutedEventArgs e)
    {
        BindingExpression expr = Explicit.GetBindingExpression(TextBox.TextProperty);
        expr.UpdateSource();
    }
}
```

Bennene die Textbox in der View:

```
<TextBox Name="Explicit" />
```

Binding Modes

One-Way

- transfers values from the ViewModel to the View

One-Way-To-Source

- transfers values from the View to the ViewModel

Two-Way

- transfers values in both directions

One-Time

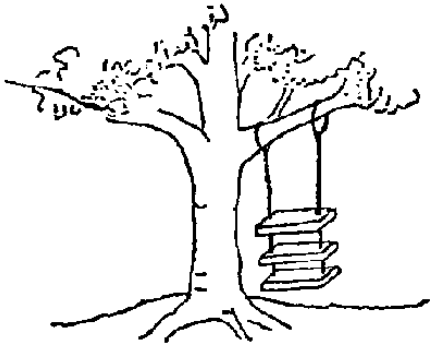
- transfers data from ViewModel to View only when the binding source is set:
  - After this, the binding doesn't monitor changes and doesn't perform any updates, unless the binding source itself is reset.

Update Triggers

- determines when changes made in the control's property are passed back to the data source
  - valid only for one-way and two-way bindings
- Four options defined in the *UpdateSourceTrigger* enumeration:
  - **LostFocus**
    causes changes in the property of a control when the control loses focus
  - **PropertyChanged**
    changes to the information in the control are copied to the source immediately
  - **Explicit**
    changes to the property are not copied automatically
    You must call the data binding's *UpdateSource* method
  - **Default**
    Setting the update trigger to *Default* uses the standard option for the property, controls use different options for the update trigger

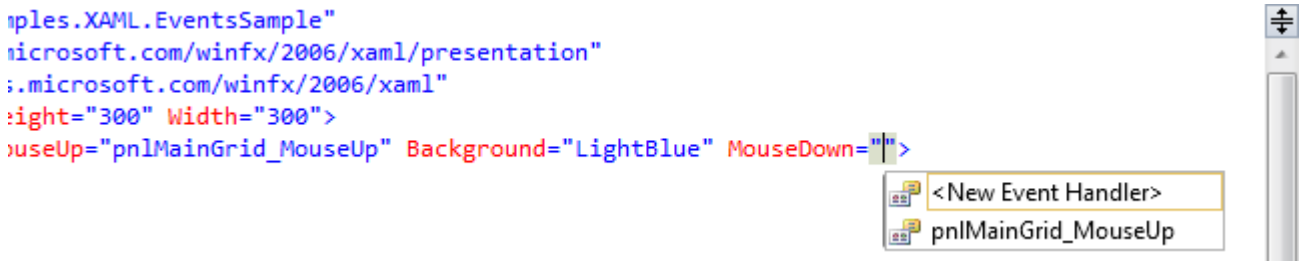
ICommand

Software Entwicklung



Overview

- WPF Basics
  - New Window
  - MessageBox
- WPF Panels
- WPF Data Binding
  - Model View ViewModel (MVVM)
- WPF Controls
  - Events in WPF
  - Control - Button
- WPF ICommand
  - ICommand
  - ConcreteCommand
  - Bind Command to Button – CommandParameter
- IValueConverter
- RelayCommand
- WPF Exercises

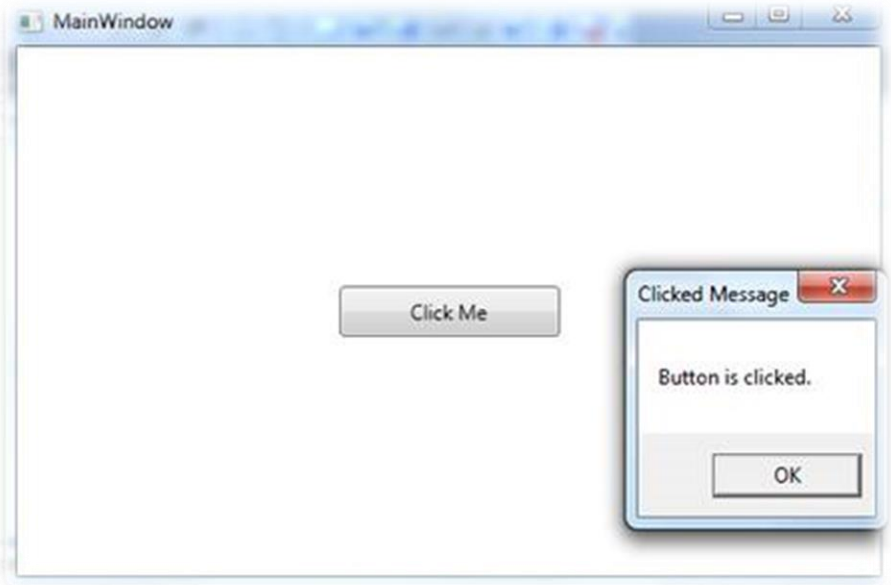


Events

Click, MouseMove, MouseDown, MouseUp …

```
private void pnlMainGrid_MouseUp(object sender, MouseButtonEventArgs e)
{
        MessageBox.Show("You clicked me at " + e.GetPosition(this).ToString());
}
```

https://wpf-tutorial.com/xaml/events-in-xaml/



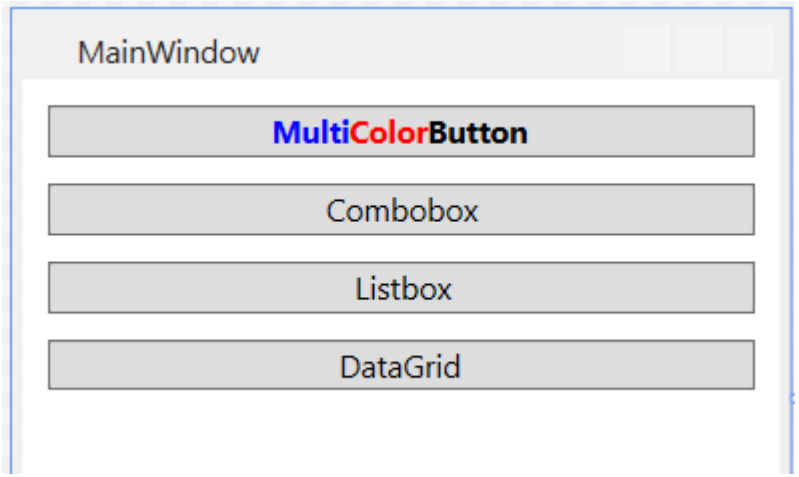
Button

includes all of the functionality required to generate a clickable button

The type inherits from ContentControl, meaning that a button can contain plain text or any other control, including layout controls with their own children.

Button

- Colored Button & Click
- Buttons with Commands

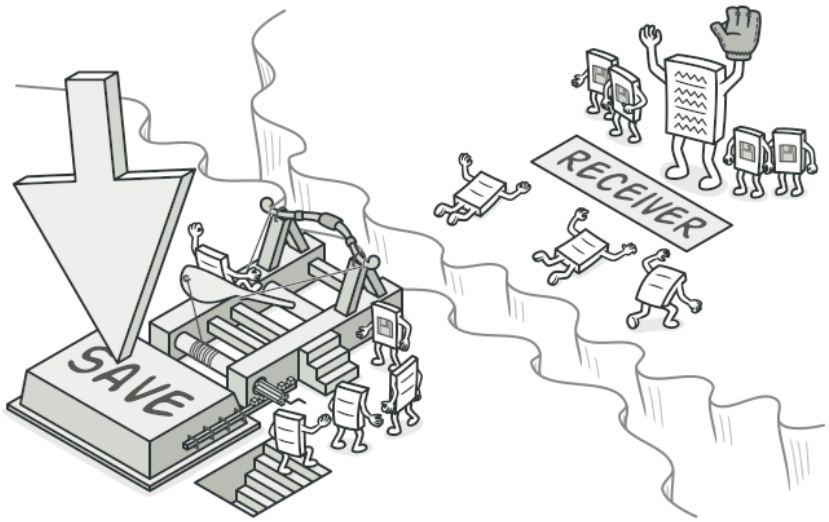



```
<Window.DataContext>
    <local:VM_MainWindowCommands/>
</Window.DataContext>
<StackPanel>
    <!-- Set a method behind a Click-Event in Buttons -->
    <Button Height="20" Margin="10,10,10,5" Click="Button_Click" >
        <Button.FontWeight>Bold</Button.FontWeight>
        <Button.Content>
            <WrapPanel>
                <TextBlock Foreground="Blue">Multi</TextBlock>
                <TextBlock Foreground="Red">Color</TextBlock>
                <TextBlock>Button</TextBlock>
            </WrapPanel>
        </Button.Content>
    </Button>
    <!-- Bind ICommand Properties to Buttons -->
    <Button Height="20" Content="Combobox"  Command="{Binding ShowComboboxCommand}" Margin="10,5,10,5"/>
    <Button Height="20" Content="Listbox"  Command="{Binding ShowListBoxCommand}" Margin="10,5,10,5"/>
    <Button Height="20" Content="DataGrid" Command="{Binding ShowDataGridCommand}" Margin="10,5,10,5"/>
</StackPanel>
```

Button_Click with MessageBox

- Advanced Messagebox

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    MessageBoxResult result = MessageBox.Show("Would you like to greet the world with a " +
        "\"Hello, world\"?", "My App", MessageBoxButton.YesNoCancel);

    switch (result)
    {
        case MessageBoxResult.Yes:
            MessageBox.Show("Hello to you too!", "My App");
            break;
        case MessageBoxResult.No:
            MessageBox.Show("Oh well, too bad!", "My App");
            break;
        case MessageBoxResult.Cancel:
            MessageBox.Show("Nevermind then...", "My App");
            break;
    }
}
```

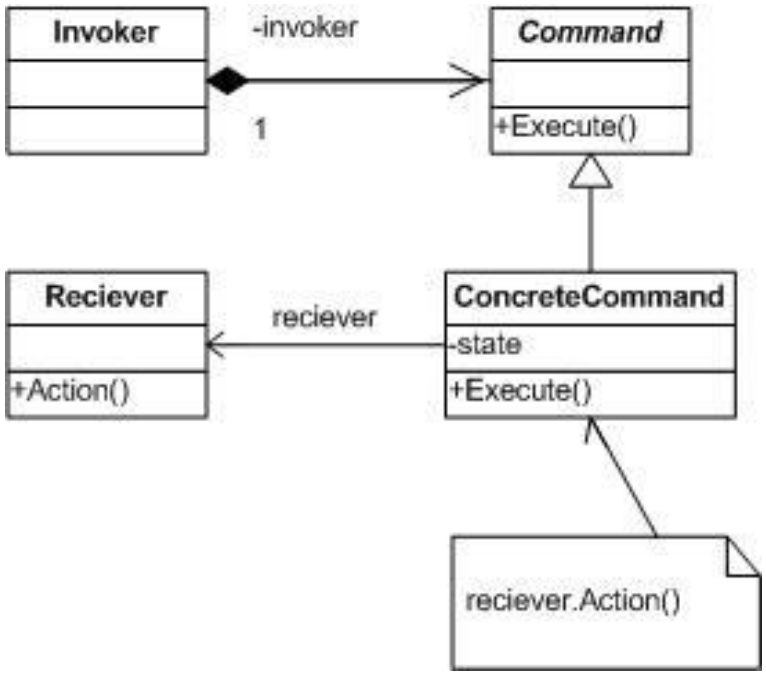


Commands

Not only property can be bound to a Control, Commands (Functions) can be bound to a Button using the ICommand Interface with Execute and CanExecute Methods

Command Pattern

- is a behavioral design pattern
- is used to turn a request into an object which contains all the information about the request



ICommand Interface explained

```
public interface ICommand
{
    //
    // Zusammenfassung:
    //     Tritt ein, wenn Änderungen auftreten, die sich auf die Ausführung des Befehls
    //     auswirken.
    event EventHandler CanExecuteChanged;

    //
    // Zusammenfassung:
    //     Definiert die Methode, die bestimmt, ob der Befehl im aktuellen Zustand ausgeführt
    //     werden kann.
    //
    // Parameter:
    //   parameter:
    //     Vom Befehl verwendete Daten. Wenn der Befehl keine Datenübergabe erfordert, kann
    //     das Objekt auf null festgelegt werden.
    //
    // Rückgabewerte:
    //     true, wenn der Befehl ausgeführt werden kann, andernfalls false.
    bool CanExecute(object parameter);
    //
    // Zusammenfassung:
    //     Definiert die Methode, die aufgerufen wird, wenn der Befehl aufgerufen wird.
    //
    // Parameter:
    //   parameter:
    //     Vom Befehl verwendete Daten. Wenn der Befehl keine Datenübergabe erfordert, kann
    //     das Objekt auf null festgelegt werden.
    void Execute(object parameter);
}
```

Bind Commands

- Set a DataContext to the Window or Control
- Create a Class SpecificCommand, which implements ICommand
- Create a Execute and a CanExcecute Method for the Button

- Bind the Command to the Button

```
<Button Command="{Binding PersonAddCommand}" />
```

AddCommand

- Write an AddPerson Method
- Create a PersonAddCommand

PersonsVM

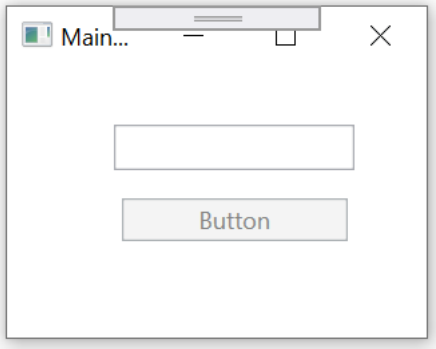
```
public void AddPerson(Person p)
{
    People.Add(new PersonVM(p));
}
public void RemovePerson(Person p)
{
    People.Add(new PersonVM(p));
}

public ICommand PersonAddCommand
{
    get
    {
        return new PersonAddCommand(this);
    }
}
```

```
class PersonAddCommand : ICommand
{
    private PersonsVM personsVM;
    public PersonAddCommand(PersonsVM personsVM)
        { this.personsVM = personsVM; }

    public event EventHandler CanExecuteChanged;
    public bool CanExecute(object parameter)
    {
        return true;
    }

    public void Execute(object parameter)
    {
        personsVM.AddPerson(new Person());
    }
}
```

```
<Button Command="{Binding PersonAddCommand}" />
```

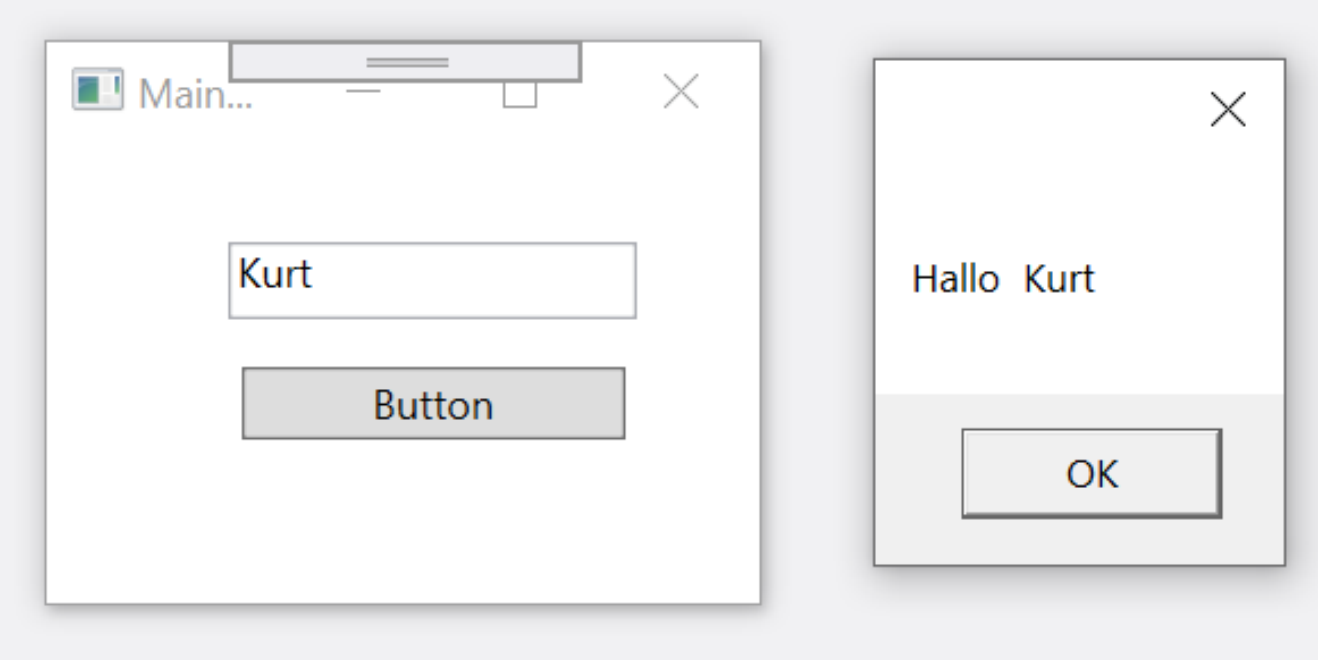
WPF Greet

WPF Command und Data Binding



Greet Example

- Add a Textbox for the Name
- Add a GreetCommand Button to activate a Greeting via Messagebox



AViewModel to Copy

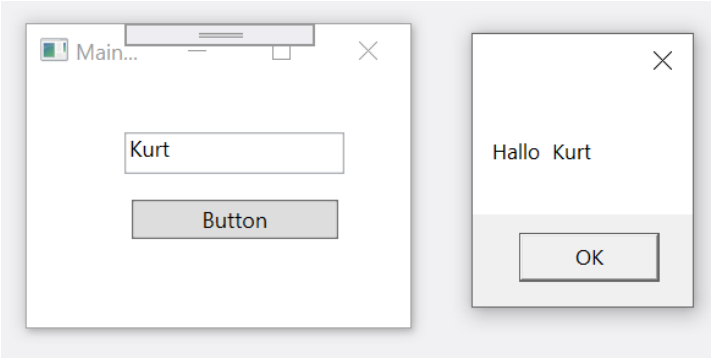
```
abstract class ANotifyPropertyChanged : INotifyPropertyChanged {
    public event PropertyChangedEventHandler PropertyChanged;

    public void OnPropertyChanged([CallerMemberName] string property = null)
        => PropertyChanged(this, new PropertyChangedEventArgs(property));
}
```

```
//To Copy
 abstract class ANotifyPropertyChanged : INotifyPropertyChanged {
        public event PropertyChangedEventHandler PropertyChanged;
        public void OnPropertyChanged([CallerMemberName] string property = null)
            => PropertyChanged(this, new PropertyChangedEventArgs(property));
    }
```

Greet View Model

```
public partial class MainWindow : Window {
    public MainWindow() {
        InitializeComponent();
        DataContext = new GreetVM();
    }
}
```

```
class GreetVM :ANotifyPropertyChanged {

    string name;
    public string Name {
        get { return name; }
        set { name = value;  OnPropertyChanged(); }
    }

    public GreetVM() {

        greetCmd= new ConcreteGreetCommand(this);

    }

    private ICommand greetCmd;
    public ICommand GreetCmd { get { return greetCmd; } }

}
```



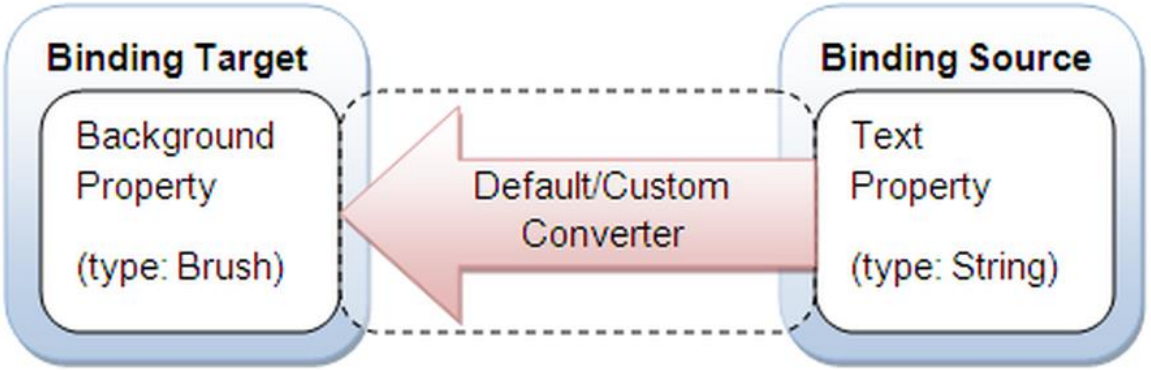

```
<Button Command="{Binding GreetCmd}" Content="Greet"
<TextBox Text="{Binding Name, Mode=TwoWay,
        UpdateSourceTrigger=PropertyChanged}"
```

Add a ConcreteCommand

```
class ConcreteGreetCommand : ICommand {
    private GreetVM vm;

    public event EventHandler CanExecuteChanged {
        add => CommandManager.RequerySuggested += value;
        remove => CommandManager.RequerySuggested -= value;
    }
    public ConcreteGreetCommand(GreetVM vm) {
        this.vm = vm;
    }

    public bool CanExecute(object parameter) {
        return (vm.Name != null && vm.Name != "");
    }

    public void Execute(object parameter) {
        MessageBox.Show("Hallo  " + vm.Name);
    }
}
```

Value Conversion

with IValueConverter



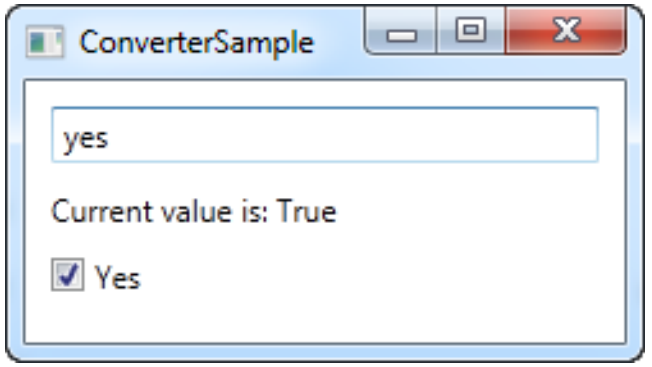
https://www.wpf-tutorial.com/data-binding/value-conversion-with-ivalueconverter/

Overview

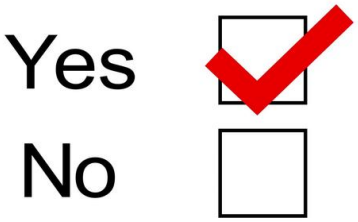
- WPF Basics
  - New Window
  - MessageBox
- WPF Panels
- WPF Data Binding
  - Model View ViewModel (MVVM)
- WPF Controls
  - Events in WPF
  - Control - Button
- WPF ICommand
  - ICommand
  - ConcreteCommand
  - Bind Command to Button – CommandParameter
- IValueConverter
- RelayCommand
- WPF Exercises

Yes No Converter

String to Boolean Converter



Example Yes No Boolean Converter

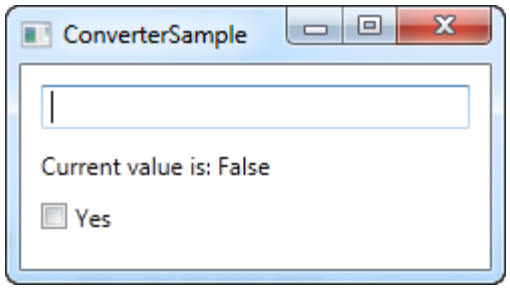


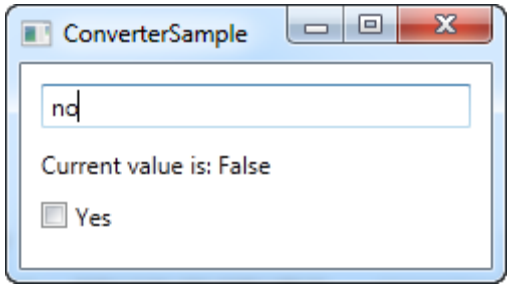
- Write the Converter Class
  - With Convert und ConvertBack Method
- Add the Converter as Resource to the Window
- Write a ViewModel Class, set the DataContext
- Set the Binding
  - Use the Properties form the ViewModel class
  - Use Converter defined as Static Resource
  - Set Parameter if necessary to the Convert Method

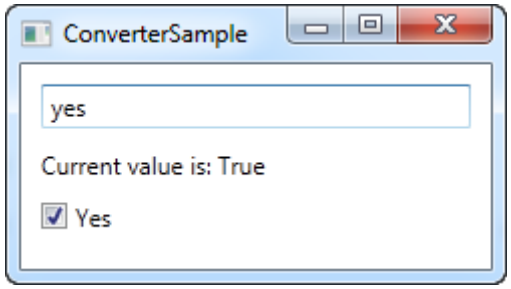
YesNoBooleanConverter Class

```
public class YesNoToBooleanConverter : IValueConverter
{
    2 Verweise
    public object Convert(object value, Type targetType,
        object parameter, System.Globalization.CultureInfo culture)
    {
        switch (value.ToString().ToLower())
        {
            case "yes":
            case "oui":
                return true;
            case "no":
            case "non":
                return false;
        }
        return false;
    }

    2 Verweise
    public object ConvertBack(object value, Type targetType,
        object parameter, System.Globalization.CultureInfo culture)
    {
        if (value is bool)
        {
            if ((bool)value == true)
                return "yes";
            else
                return "no";
        }
        return "no";
    }
}
```

```
using System.Windows.Data;
```

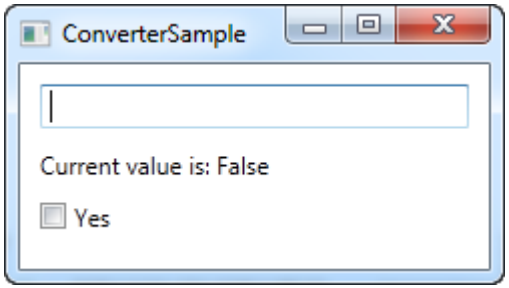


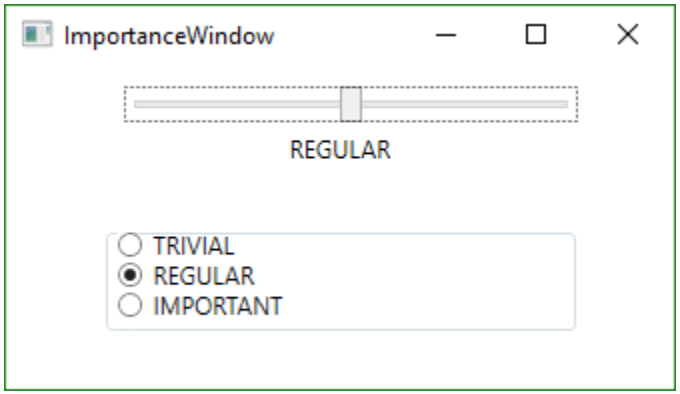




Binding a Static Resource

```
<Window x:Class="WPF_DataCommandBinding.YesNoConverter.YesNoWindow"
        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:WPF_DataCommandBinding.YesNoConverter"
        mc:Ignorable="d"
      Title="ConverterSample" Height="140" Width="250">
    <Window.Resources>
        <local:YesNoToBooleanConverter x:Key="YesNoToBooleanConverter" />
    </Window.Resources>
    <StackPanel Margin="10">
        <TextBox Name="txtValue" />
        <WrapPanel Margin="0,10">
            <TextBlock Text="Current value is: " />
            <TextBlock Text="{Binding ElementName=txtValue, Path=Text,
                Converter={StaticResource YesNoToBooleanConverter}}" />
        </WrapPanel>
        <CheckBox IsChecked="{Binding ElementName=txtValue, Path=Text,
            Converter={StaticResource YesNoToBooleanConverter}}" Content="Yes" />
    </StackPanel>
</Window>
```






```
enum EImportance { TRIVIAL, REGULAR, IMPORTANT};
```

Importance Enum -> Converter

Convert a Enum to a Int and bind it to a Slider

Convert a Enum to an Boolean and bind it to a RaidoButton

XAML ImportanceVM

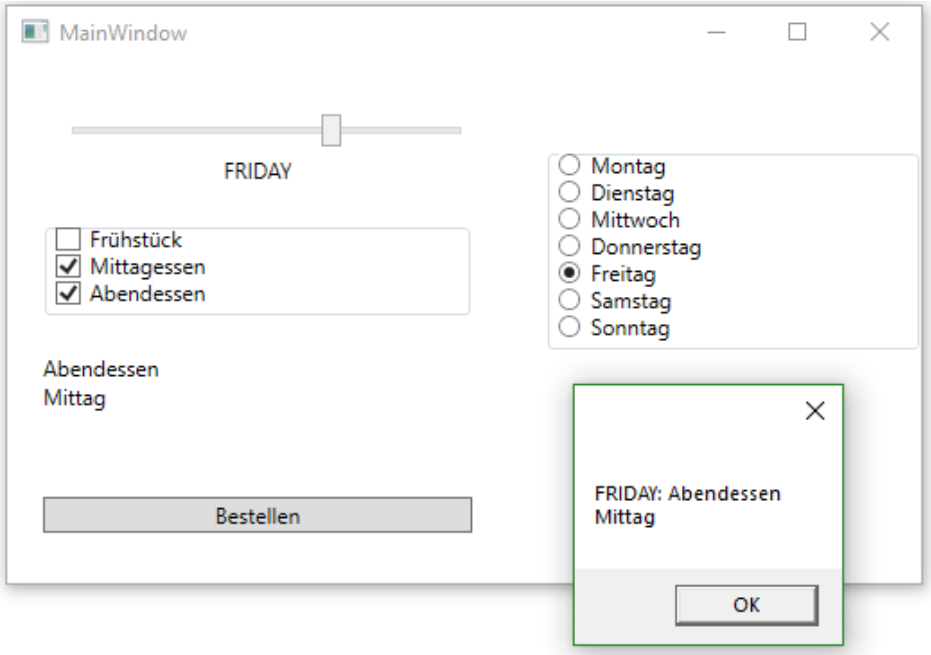
```
class ImportanceViewModel :
BaseViewModel {
   private EImportance importance;
   public EImportance Importance {
       get { return importance; }
       set { importance = value;
             OnPropertyChanged(); }
   }
}
```

[optional]
Add a ComboBox with the EnumValues ☺

```
<Window.DataContext>
        <local:ImportanceViewModel/>
    </Window.DataContext>
    <Grid>
        <Grid.Resources>
            <local:EnumBooleanConverter x:Key="EnumBooleanConverter" />
            <local:EnumIntConverter x:Key="EnumIntConverter"/>
        </Grid.Resources>
        <StackPanel Margin="0,0,0,0" HorizontalAlignment="Center">
            <Slider IsSnapToTickEnabled="True" HorizontalAlignment="Center"
                Margin="10,10,0,0" VerticalAlignment="Top" Width="227"
                Minimum="0" Maximum="2"
             Value="{Binding
                        Path=Importance,
                        Converter={StaticResource EnumIntConverter},
                        Mode=TWOWAY}"
                TickFrequency="1"/>
            <Label HorizontalAlignment="Center" Height="23" Margin="0,0,0,0"
                 Content="{Binding Path=Importance}" />
            <!--<ComboBox>            </ComboBox> -->
            <TextBlock Height="30"/>
            <GroupBox>
                <StackPanel >
                    <RadioButton IsChecked="{Binding Path=Importance,
                        Converter={StaticResource EnumBooleanConverter},
                        ConverterParameter={x:Static local:EImportance.TRIVIAL}}"
                                 Content="TRIVIAL" GroupName="importance" />
                    <RadioButton IsChecked="{Binding Path=Importance,
                        Converter={StaticResource EnumBooleanConverter},
                        ConverterParameter={x:Static local:EImportance.REGULAR}}"
                                 Content="REGULAR" GroupName="importance" />
                    <RadioButton IsChecked="{Binding Path=Importance,
                        Converter={StaticResource EnumBooleanConverter},
                        ConverterParameter={x:Static local:EImportance.IMPORTANT}}"
                                 Content="IMPORTANT" GroupName="importance" />
                </StackPanel>
            </GroupBox>
        </StackPanel>
    </Grid>
</Window>
```

Converter

```
class EnumBooleanConverter : IValueConverter  {
        public object Convert(object value, Type targetType, object parameter, CultureInfo culture)   {
            //checks if Selection from RadioButtonCheckBoxVM has the same value
            //as the ConverterParameter. Returns true or false
            // return ((Enum)value).HasFlag((Enum)parameter);
            return ((EImportance)value == (EImportance)parameter);
        }

        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)  {
            //If the radiobutton is checked, it returns the ConverterParameter
            //return value.Equals(true) ? parameter : Binding.DoNothing;
             return ((bool)value == true) ?  parameter : null;
        }
}

class EnumIntConverter : IValueConverter {
      public object Convert(object value, Type targetType, object parameter, CultureInfo culture) {
            //return (int)value;
             EImportance e = (EImportance)value;
             return System.Convert.ToInt32(value);
      }

      public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture){
            return (EImportance)System.Convert.ToInt32(value);
      }
}
```



Order Meal EnumConverter Example

Create an enum EWeekday

… bind it to a Slider & some RadioButtons

Order 1 to 3 Meals a Weekday,
show the order in a MessageBox

Binding RadioButton on Enum

- Enum EWeekdayw in VM:

```
private EWeekdays eweekday;

public EWeekdays Weekday {
    get { return eweekday; }
    set {
        eweekday = value;
        OnPropertyChanged();
    }
}
```

- Set a Value Converter

```
<Grid.Resources>
    <local:EnumBooleanConverter x:Key="EnumBooleanConverter" />
</Grid.Resources>
```

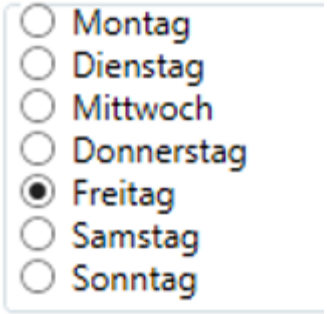
- Binding the RadioButtons

```
<RadioButton Content="Montag" GroupName="weekdays"
             IsChecked="{Binding Path=Weekday,
                         Converter={StaticResource EnumBooleanConverter},
                         ConverterParameter={x:Static local:EWeekdays.MONDAY}}"  />
```

Enum ValueConverter

```
class EnumBooleanConverter : IValueConverter
{
    1-Verweis
    public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
    {
        //checks if Selection from RadioButtonCheckBoxVM has
        //the same value as the ConverterParameter. Returns true or false

        return (EWeekdays)value == (EWeekdays)parameter;
        //return ((Enum)value).HasFlag((Enum)parameter);

    }

    1-Verweis
    public object ConvertBack(object value, Type targetType,
        object parameter, System.Globalization.CultureInfo culture)
    {
        //If the radiobutton is checked, it returns the ConverterParameter
        return value.Equals(true) ? parameter : Binding.DoNothing;
    }
}
```

Bind all RadioButtons

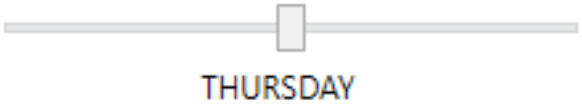



```
<Grid.Resources>
    <local:EnumBooleanConverter x:Key="EnumBooleanConverter" />
</Grid.Resources>
```

```
<GroupBox>
    <StackPanel >
        <RadioButton IsChecked="{Binding Path=Weekday, Converter={StaticResource EnumBooleanConverter},
        ConverterParameter={x:Static local:EWeekdays.MONDAY}}" Content="Montag" GroupName="weekdays" />
        <RadioButton IsChecked="{Binding Path=Weekday, Converter={StaticResource EnumBooleanConverter},
        ConverterParameter={x:Static local:EWeekdays.TUESDAY}}" Content="Dienstag" GroupName="weekdays" />
        <RadioButton IsChecked="{Binding Path=Weekday, Converter={StaticResource EnumBooleanConverter},
        ConverterParameter={x:Static local:EWeekdays.WEDNESDAY}}" Content="Mittwoch" GroupName="weekdays" />
        <RadioButton IsChecked="{Binding Path=Weekday, Converter={StaticResource EnumBooleanConverter},
        ConverterParameter={x:Static local:EWeekdays.THURSDAY}}" Content="Donnerstag" GroupName="weekdays" />
        <RadioButton IsChecked="{Binding Path=Weekday, Converter={StaticResource EnumBooleanConverter},
        ConverterParameter={x:Static local:EWeekdays.FRIDAY}}" Content="Freitag" GroupName="weekdays" />
        <RadioButton IsChecked="{Binding Path=Weekday, Converter={StaticResource EnumBooleanConverter},
        ConverterParameter={x:Static local:EWeekdays.SATURDAY}}" Content="Samstag" GroupName="weekdays" />
        <RadioButton IsChecked="{Binding Path=Weekday, Converter={StaticResource EnumBooleanConverter},
        ConverterParameter={x:Static local:EWeekdays.SUNDAY}}" Content="Sonntag" GroupName="weekdays" />
    </StackPanel>
</GroupBox>
```

Slider & Enum

- Slider




```
public class SliderVM:AViewModel
{
    private EWeekdays eweekday;

    public EWeekdays EWeekday
    {
        get
        {
            return eweekday;
        }
        set
        {
            eweekday = value;
            CallPropertyChanged("EWeekday");
        }
    }
    public SliderVM()
    {
        eweekday = EWeekdays.FR;
    }
}
```

- with Converter

```
public class EnumIntConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        return (int)value;
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        return (EWeekdays)System.Convert.ToInt32(value);
    }
}
```

XAML with Slider & Enum Binding

- EnumIntConverter

```
    <Window.DataContext>
        <local:SliderVM></local:SliderVM>
    </Window.DataContext>
    <Grid>
        <Grid.Resources>
            <local:EnumIntConverter x:Key="EnumIntConverter"></local:EnumIntConverter>
        </Grid.Resources>
        <Slider IsSnapToTickEnabled="True" HorizontalAlignment="Left"
                Margin="10,10,0,0" VerticalAlignment="Top" Width="497"
                Minimum="0" Maximum="6"
                Value="{Binding
                          Path=EWeekday,
                          Converter={StaticResource EnumIntConverter},
                          Mode=TwoWay}"
                TickFrequency="1"/>
        <TextBox HorizontalAlignment="Left" Height="23" Margin="10,33,0,0" TextWrapping="Wrap"
                 Text="{Binding Path=EWeekday}" VerticalAlignment="Top" Width="497"/>
    </Grid>
</Window>
```

RelayCommand

ICommand with Generic Delegates

delegate

lamda =>

Func<>

Action<>

predicate<>

Genereric Delegates

- Func<>
  - returns a value
- Action<>
  - no return value (void)
- Predicate<>
  - returns a bool

delegate

lamda =>

Func<>

Action<>

predicate<>

RelayCommand - Simple

```
public class RelayCommand : ICommand
{
    private Predicate<object> canExecute;
    private Action<object> execute;

    public RelayCommand(Action<object> execute, Predicate<object> canExecute) {
        this.canExecute = canExecute;
        this.execute = execute;
    }
    public bool CanExecute(object parameter)  => canExecute(parameter);
    public void Execute(object parameter)     => execute(parameter);

    public event EventHandler CanExecuteChanged {
        add => CommandManager.RequerySuggested += value;
        remove => CommandManager.RequerySuggested -= value;
    }
}
```

Constructor

- with Action & Predicate

CanExecuteChanged

- Update the Button:
  - CanExecute returns a new value
  - Button gets activated or deactivated

  - Keep the EventHandler updated:
    Add to the CommandClass

```
public event EventHandler CanExecuteChanged {
    add { CommandManager.RequerySuggested += value; }
    remove { CommandManager.RequerySuggested -= value; }
}
```

Bind Commands with RelayCommand

- Set a DataContext to the Window or Control
- Use a BaseClass RelayCommand for all ICommand
- Create a Execute and a CanExcecute Method for the Button
- Create a new instance of `RelayCommand(Excecute,CanExecute)` for the Button

ICommand

- Bind a method to a control using the command attribute
  - Works the same way as Content and ItemsSource

    Except

    -> bind it to a *property* that returns an ICommand

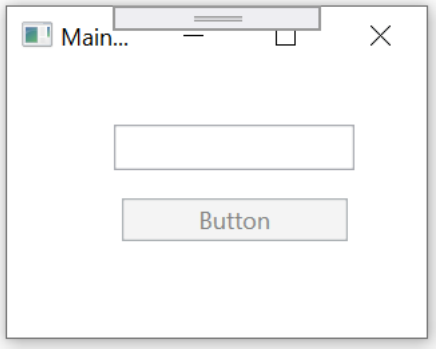
- implement a class 'RelayCommand' that implements ICommand then use it
  - ICommand requires two methods:
    - bool CanExecute
    - void Execute

Using RelayCommand

```
class VM_MainWindowCommands
{
    0 Verweise
    public ICommand ShowComboboxCommand {
        get { return new RelayCommand(o => NewComboboxCommand(), o => true); }
    }
    1-Verweis
    private void NewComboboxCommand() {
        WindowComboBox w = new WindowComboBox();
        w.Show();
    }
    0 Verweise
    public ICommand ShowListBoxCommand {
        get { return new RelayCommand(o => NewListBoxWindow(), o => true); }
    }
    1-Verweis
    private void NewListBoxWindow() {
        WindowListbox w = new WindowListbox();
        w.Show();
    }
    0 Verweise
    public ICommand ShowDataGridCommand {
        get { return new RelayCommand(o => NewDataGridWindow(), o => true); }
    }
    1-Verweis
    private void NewDataGridWindow() {
        WindowDataGrid w = new WindowDataGrid();
        w.Show();
    }
}
```

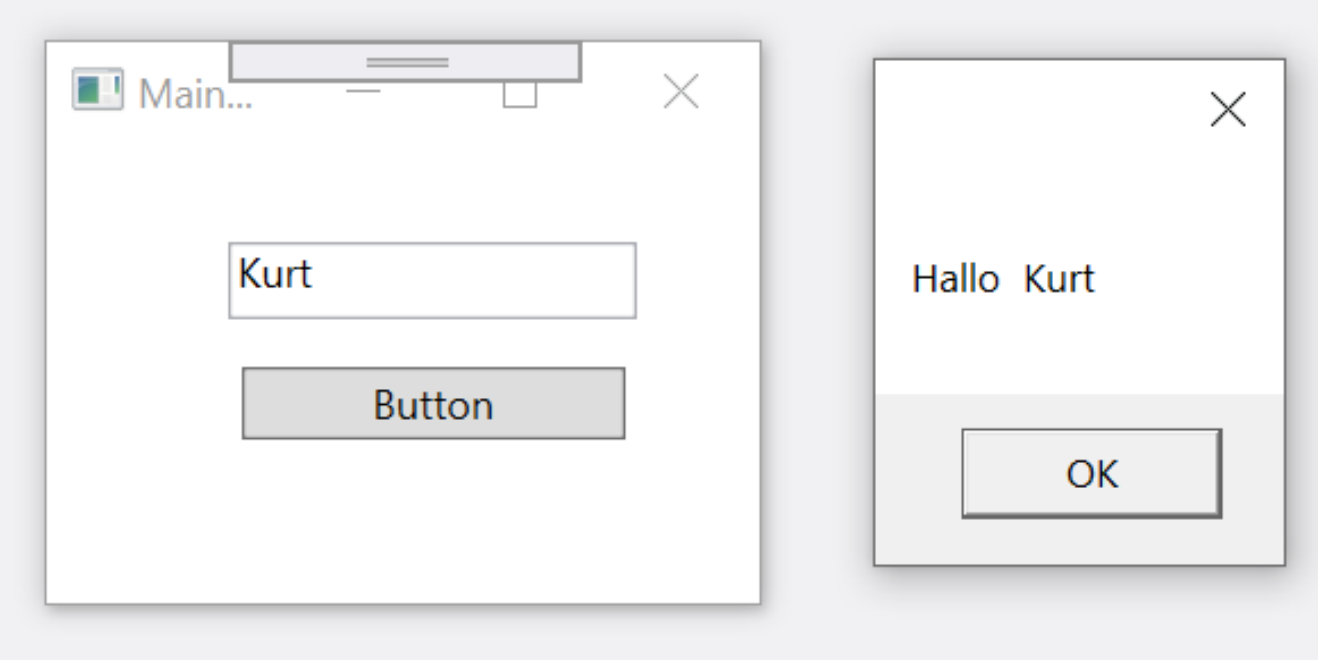
WPF Greet

WPF Command und Data Binding



Greet Example

- Add a Textbox for the Name
- Add a GreetCommand Button to activate a Greeting via Messagebox



RelayCommand to Copy

```
public class RelayCommand : ICommand    {
        private Predicate<object> canExecute;
        private Action<object> execute;
        public event EventHandler CanExecuteChanged {
            add => CommandManager.RequerySuggested += value;
            remove => CommandManager.RequerySuggested -= value;
        }
        public RelayCommand(Action<object> execute, Predicate<object> canExecute){
            this.canExecute = canExecute;
            this.execute = execute;
        }
        public bool CanExecute(object parameter) => canExecute(parameter);
        public void Execute(object parameter)    => execute(parameter);
    }
```

AViewModel to Copy

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

    public void OnPropertyChanged([CallerMemberName] string property = null)
        => PropertyChanged(this, new PropertyChangedEventArgs(property));
}
```

```
//To Copy
 abstract class ANotifyPropertyChanged : INotifyPropertyChanged {
        public event PropertyChangedEventHandler PropertyChanged;
        public void OnPropertyChanged([CallerMemberName] string property = null)
            => PropertyChanged(this, new PropertyChangedEventArgs(property));
    }
```

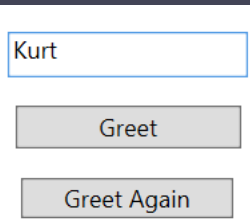
Greet View Model

```
class GreetVM :ANotifyPropertyChanged {

    string name;
    public string Name {
        get { return name; }
        set { name = value;  OnPropertyChanged(); }
    }

    public GreetVM() {
        greetCmd=new RelayCommand(
            //Action for Execute
            o => MessageBox.Show("Hallo  " + Name),
            //Predicate for CanExecute
            x => Name != null && Name != "");
    }

    private ICommand greetCmd;
    public ICommand GreetCmd { get { return greetCmd; } }

}
```

```
public partial class MainWindow : Window {
    public MainWindow() {
        InitializeComponent();
        DataContext = new GreetVM();
    }
}
```

```
<Button Command="{Binding GreetCmd}" Content="Greet"
<TextBox Text="{Binding Name, Mode=TwoWay,
        UpdateSourceTrigger=PropertyChanged}"
```

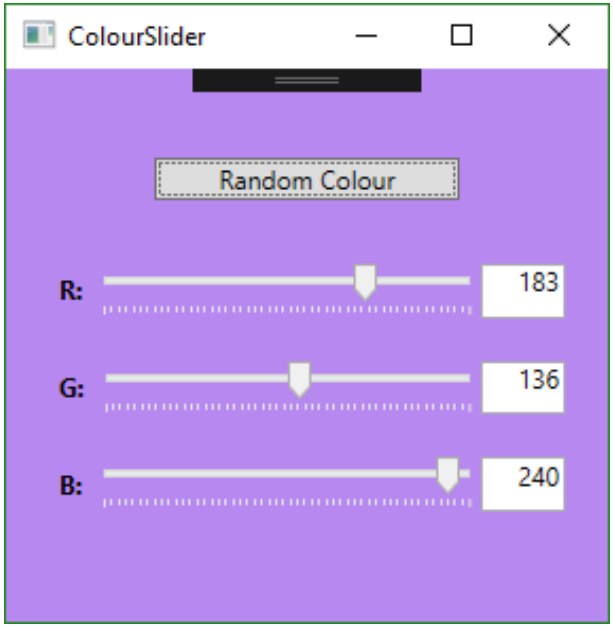
Add a ConcreteCommand




```
class ConcreteGreetCommand : ICommand {
    private GreetVM vm;

    public event EventHandler CanExecuteChanged {
        add => CommandManager.RequerySuggested += value;
        remove => CommandManager.RequerySuggested -= value;
    }
    public ConcreteGreetCommand(GreetVM vm) {
        this.vm = vm;
    }

    public bool CanExecute(object parameter) {
        return (vm.Name != null && vm.Name != "");
    }

    public void Execute(object parameter) {
        MessageBox.Show("Hallo  " + vm.Name);
    }
}
```

```
<Button Command="{Binding GreetCmd}" Content="Greet"
<TextBox Text="{Binding Name, Mode=TwoWay,
        UpdateSourceTrigger=PropertyChanged}"
<Button Content="Greet Again" Command="{Binding GreetConcreteCmd}"
```

```
class GreetVM :ANotifyPropertyChanged {

    string name;
    public string Name ...

    public GreetVM() {
        greetCmd=new RelayCommand(
            //Action for Execute
            o => MessageBox.Show("Hallo  " + Name),
            //Predicate for CanExecute
            x => Name != null && Name != "");
        greetConcreteCmd = new ConcreteGreetCommand(this);
    }

    private ICommand greetCmd;
    public ICommand GreetCmd {
        get { return greetCmd; } }

    private ICommand greetConcreteCmd;
    public ICommand GreetConcreteCmd {
        get { return greetConcreteCmd; } }

}
```



BackgroundColor Sliders

Create 3 Sliders to set the Background-Color

Xaml – Slider for RedValue

- Add 3 Slider to the GUI

```
    Title="ColourSlider" Height="300" Width="300">
<Window.DataContext>
    <local:SliderVM></local:SliderVM>
</Window.DataContext>
<StackPanel Margin="10" VerticalAlignment="Center">
    <Button x:Name="button" Content="Random Colour" Margin="60,0,60,20"
            Command="{Binding RandomColour}" />
    <DockPanel VerticalAlignment="Center" Margin="10">
        <Label DockPanel.Dock="Left" FontWeight="Bold" Content="R:"/>
        <TextBox Text="{Binding RedValue, UpdateSourceTrigger=PropertyChanged}"
                 DockPanel.Dock="Right" TextAlignment="Right" Width="40" />
        <Slider Maximum="255" TickPlacement="BottomRight" TickFrequency="5"
                Value="{Binding RedValue,Mode=TwoWay}" IsSnapToTickEnabled="True"
                x:Name="slider_red" ValueChanged="ColorSlider_ValueChanged" />
    </DockPanel>
```

Slider for Green & Blue Values

```
    <DockPanel VerticalAlignment="Center" Margin="10">
        <Label DockPanel.Dock="Left" FontWeight="Bold" Content="G:"/>
        <TextBox Text="{Binding GreenValue, UpdateSourceTrigger=PropertyChanged}"
                 DockPanel.Dock="Right" TextAlignment="Right" Width="40" />

        <Slider Maximum="255" TickPlacement="BottomRight" TickFrequency="5"
                Value="{Binding GreenValue,Mode=TwoWay}" IsSnapToTickEnabled="True"
                x:Name="slider_green" ValueChanged="ColorSlider_ValueChanged" />
    </DockPanel>
    <DockPanel VerticalAlignment="Center" Margin="10">
        <Label DockPanel.Dock="Left" FontWeight="Bold" Content="B:"/>
        <TextBox Text="{Binding BlueValue, UpdateSourceTrigger=PropertyChanged}"
                 DockPanel.Dock="Right" TextAlignment="Right" Width="40" />
        <Slider Maximum="255" TickPlacement="BottomRight" TickFrequency="5"
                Value="{Binding BlueValue,Mode=TwoWay}" IsSnapToTickEnabled="True"
                x:Name="slider_blue" ValueChanged="ColorSlider_ValueChanged" />
    </DockPanel>
</StackPanel>
```

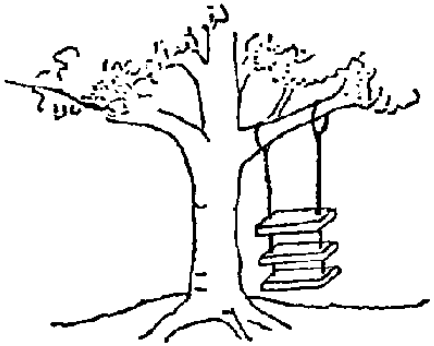
SliderViewModel

```
class SliderVM:INotifyPropertyChanged
{
    private int redValue;
    public int RedValue...
    private int greenValue;
    public int GreenValue...

    private int blueValue;
    public int BlueValue...

    public RelayCommand RandomColour
    {
        get
        {
            return new RelayCommand(
                o =>
                {
                    Random rnd = new Random();
                    RedValue = rnd.Next(1, 255);
                    GreenValue = rnd.Next(1, 255);
                    BlueValue = rnd.Next(1, 255);
                },
                o => true
            );
        }
    }
```

- Use BaseVM instead of IProperyNotifyChanged
- Set DataContext
- Bind the Properties and the Command to the UI

WPF Examples

Software Entwicklung



WPF Animal Example

Use your own Topic

Excercise

- Use the given Topic instead of „Animal“
- Write 5 classes and a baseclass
- Design 5 TabItems with diffent Panels for your derived classes
- Implement ToCsv, static ToFile static FromFile

Tab Control in MainWindow

- Use different panels in each TabItem


```
<TabControl >
    <TabItem Header="Katze">
        <StackPanel Orientation="Horizontal"...>
    </TabItem>
    <TabItem Header="Hund">
        <Canvas></Canvas>
    </TabItem>
    <TabItem Header="Pferd">
        <WrapPanel></WrapPanel>
    </TabItem>
    <TabItem Header="Kuh">
        <DockPanel></DockPanel>
    </TabItem>
    <TabItem Header="Hase">
        <Grid>
            <Grid.ColumnDefinitions...>
            <Grid.RowDefinitions...>
        </Grid>
    </TabItem>
</TabControl>
```

Create Classes

- Create a baseclass and 5 derived classes
- Each derived class has 5 properties,
  set some of them in the baseclass
- Create for each derived class an enumtype

- Write the following
- Methods: ToString, ToCsv
- Static Methods: ToFile & FromFile

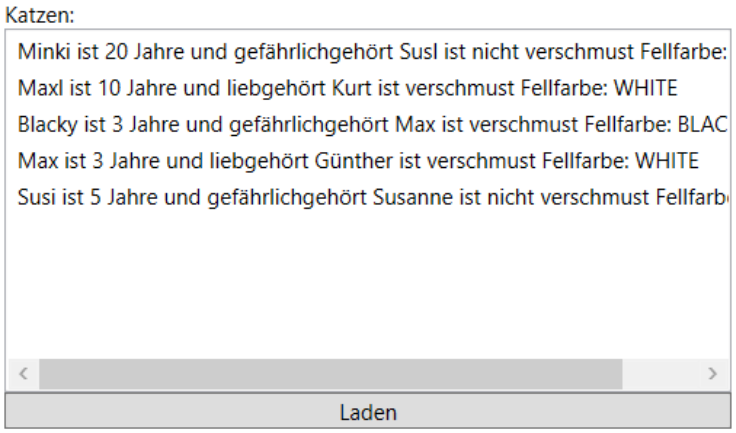
Class AAnimal & Cat

- Write 5 classes which are derived from your baseclass

```
public class AAnimal
{
    string name;
    int age;
    bool dangerous;
    public string Name { get { return name; } set { name = value; } }
    public int Age { get { return age; } set { age = value; } }
    public bool IsDangerous { get { return dangerous; } set { dangerous = value; } }
    public override string ToString()
    {
        return $"{Name};{Age};{IsDangerous};";
    }
}
public enum ECatColor { WHITE, BLACK, TIGER };
public class Cat : AAnimal
{
    string owner;
    public string Owner { get { return owner; } set { owner = value; } }
    bool pussycat;
    public bool IsPussyCat { get { return pussycat; } set { pussycat = value; } }

    public override string ToString()
    {
        return base.ToString() + $"{Owner};{IsPussyCat};";
    }

    public string ToCsv()
    {
        return this.ToString();
    }

    public static void ToFile(string path, Cat cat) { }
    public static Cat[] FromFile(string path) { return null; }
}
```

ToString & ToCsv

```
public override string ToString()
{
    string pussy = IsPussyCat ? "verschmust" : "nicht verschmust";
    return base.ToString() + $"gehört {Owner} ist {pussy} Fellfarbe: {Color}";
}
public string ToCsv()
{
    return base.ToCsv() + $"{Owner};{IsPussyCat};{Color}";
}
```




```
Minki;20;True;Susl;False;TIGER
Maxl;10;False;Kurt;True;WHITE
Blacky;3;True;Max;True;BLACK
Max;3;False;Günther;True;WHITE
Susi;5;True;Susanne;False;WHITE
```

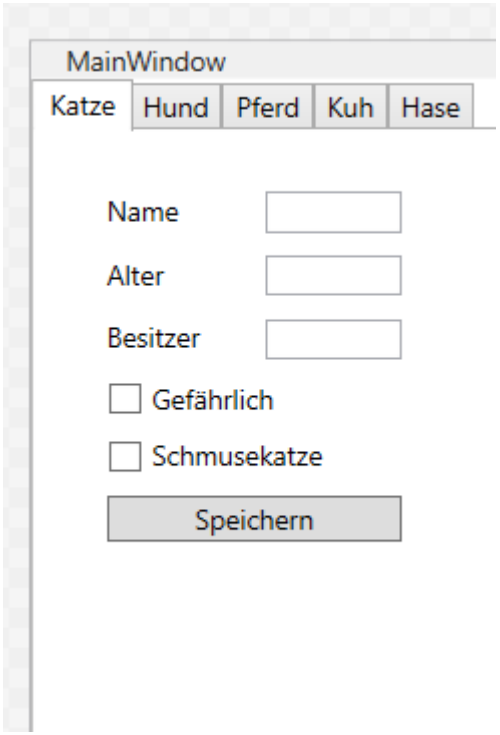
Read & Write from/to File

```
public static string Path {get; private set;}
public static void ToFile(CatVM cat)
{
    StreamWriter sw = new StreamWriter(Path, true);
    sw.WriteLine(cat.ToCsv());
    sw.Close();
}
public static List<CatVM> FromFile()
{
    StreamReader sr = new StreamReader(Path);
    List<CatVM> cats = new List<CatVM>();
    while (sr.Peek() != -1)
    {
        CatVM c = new CatVM();
        string line = sr.ReadLine();
        string[] word = line.Split(';');
        c.Name = word[0];
        c.Age = Int32.Parse(word[1]);
        c.IsDangerous = Boolean.Parse(word[2]);
        c.owner = word[3];
        c.IsPussyCat = Boolean.Parse(word[4]);
        c.Color = (ECatColor) Enum.Parse<ECatColor>(word[5]);
        cats.Add(c);
    }
    return cats;
}
```

```
Minki;20;True;Susl;False;TIGER
Maxl;10;False;Kurt;True;WHITE
Blacky;3;True;Max;True;BLACK
Max;3;False;Günther;True;WHITE
Susi;5;True;Susanne;False;WHITE
```

Design your View

- Create a view for each derived class




```
<TabControl >
    <TabItem Header="Katze">
        <StackPanel Orientation="Horizontal">
            <StackPanel Margin="20">
                <StackPanel Orientation="Horizontal">
                    <TextBlock Width="50" Text="Name" TextWrapping="Wrap" Margin="10,5"/>
                    <TextBox  Text="" Width="60" TextWrapping="Wrap" Margin="10,5"/>
                </StackPanel>
                <StackPanel Orientation="Horizontal">
                    <TextBlock Width="50" Text="Alter" TextWrapping="Wrap" Margin="10,5"/>
                    <TextBox  Text="" Width="60"  TextWrapping="Wrap" Margin="10,5"/>
                </StackPanel>
                <StackPanel Orientation="Horizontal">
                    <TextBlock Width="50" Text="Besitzer" TextWrapping="Wrap" Margin="10,5"/>
                    <TextBox  Text="" Width="60"  TextWrapping="Wrap" Margin="10,5"/>
                </StackPanel>
                <StackPanel Orientation="Horizontal">
                    <CheckBox IsChecked="False" Margin="10,5">
                        <TextBlock Text="Gefährlich" />
                    </CheckBox>
                </StackPanel>
                <StackPanel Orientation="Horizontal">
                    <CheckBox IsChecked="False" Margin="10,5">
                        <TextBlock Text="Schmusekatze" />
                    </CheckBox>
                </StackPanel>
                <Button Content="Speichern" Margin="10,5"/>
            </StackPanel>
            <!--...
        </StackPanel>
    </TabItem>
    <TabItem Header="Hund">
        <Canvas></Canvas>
    </TabItem>
```

ToDo

- Öffne den Foliensatz WPF Example New und ergänze deine View um das Databinding.
- Implementiere das INotifyPropertyChanged in der Basisklasse
- Ändere jede abgeleitete Klasse auf eine VM-Klasse und
- Rufe in jedem Set die OnPropertyChanged auf
- Setze in jedem Tab den DataContext
- Setze das Binding in der XAML-Datei für alle Properties
- Implementiere die SaveCommand-Klassen und rufe darin die static ToFile auf
- Erstelle ein ICommand-Property in der jeweiligen VM Klasse und instantiiere die SaveCommand-Klasse.
- Erstelle in jedem Tab ein Listcontrol (nutze je Tab unterschiedliche)
    - DataGrid
    - ListView
    - ListBox
    - ComboBox
- Ergänze einen Button zum Laden
- Ergänze ein Bild und zeige dieses Bild in jedem Tab an

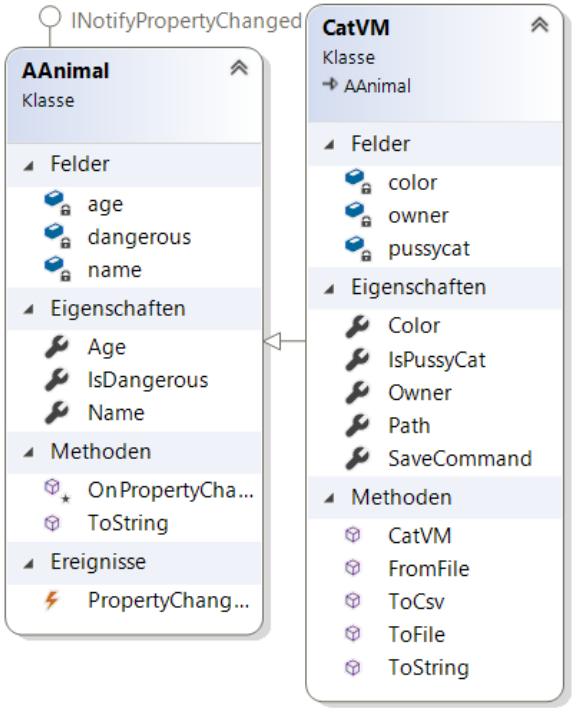
Add the DataBinding

```
public class AAnimal : INotifyPropertyChanged
{
    string name;
    int age;
    bool dangerous;
    public string Name {
        get { return name; }
        set { name = value; OnPropertyChanged(); } }
    public int Age {
        get { return age; }
        set { age = value; OnPropertyChanged(); } }
    public bool IsDangerous {
        get { return dangerous; }
        set { dangerous = value; OnPropertyChanged(); } }
    public override string ToString()...

    public event PropertyChangedEventHandler PropertyChanged;

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

}
```

- Implement INotifyProperty Changed

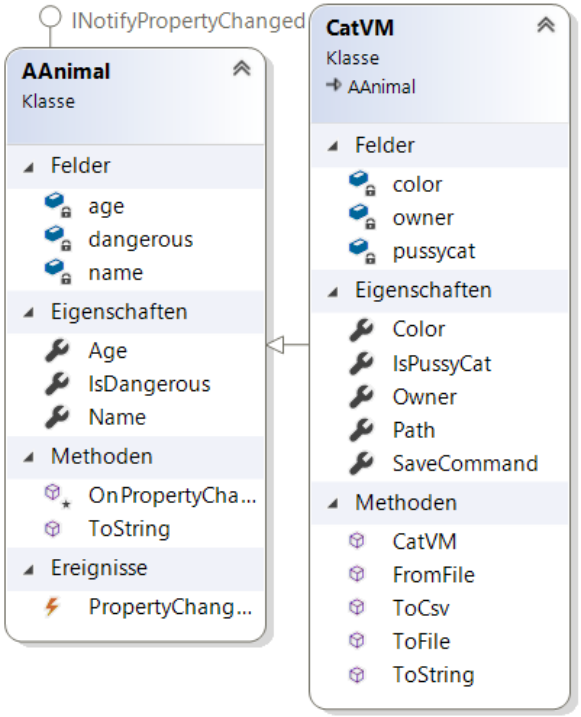
Add OnPropertyChanged

```
public class CatVM : AAnimal
{
    string owner;
    public string Owner {
        get { return owner; }
        set { owner = value; OnPropertyChanged(); } }
    bool pussycat;
    public bool IsPussyCat {
        get { return pussycat; }
        set { pussycat = value; OnPropertyChanged(); } }
    ECatColor color;
    public ECatColor Color {
        get { return color; }
        set { color = value; OnPropertyChanged(); }
    }
    public override string ToString()...
    public string ToCsv()...

    public CatVM()...
    public static string Path {get; private set;}
    public static void ToFile(CatVM cat)...
    //{Name};{Age};{IsDangerous};{Owner};{IsPussyCat};
    public static List<CatVM> FromFile()...
}
```

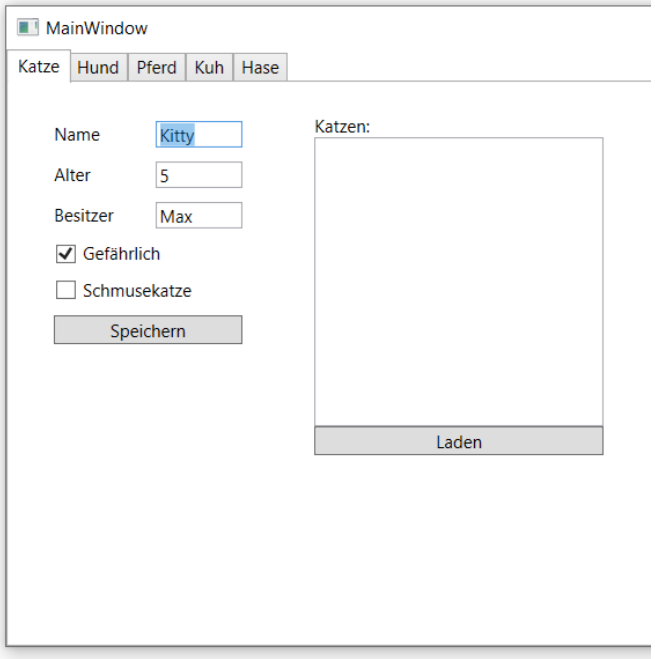


Binding im XAML

```
<TabItem Header="Katze" DataContext="CatVM">
    <StackPanel Orientation="Horizontal">
        <StackPanel Margin="20">
            <StackPanel Orientation="Horizontal">
                <TextBlock Width="50" Text="Name"
                <TextBox  Text="{Binding Name}" Width="60"
            </StackPanel>
            <StackPanel Orientation="Horizontal">
                <TextBlock Width="50" Text="Alter"
                <TextBox  Text="{Binding Age}" Width="60"
            </StackPanel>
            <StackPanel Orientation="Horizontal">
                <TextBlock Width="50" Text="Besitzer"
                <TextBox  Text="{Binding Owner}" Width="60"
            </StackPanel>
            <StackPanel Orientation="Horizontal">
                <CheckBox IsChecked="{Binding IsDangerous}"
                    <TextBlock Text="Gefährlich" />
                </CheckBox>
            </StackPanel>
            <StackPanel Orientation="Horizontal">
                <CheckBox IsChecked="{Binding IsPussyCat}"
                    <TextBlock Text="Schmusekatze" />
                </CheckBox>
            </StackPanel>
```



Save to CSV




```
Mizi;2;False;Max;True;WHITE
Minka;2;False;Max;True;TIGER
Minki;3;False;Mike;True;WHITE
Blacky;4;False;Susi;False;BLACK
Kitty;5;False;Max;False;WHITE
```

- Use values to create a cat
- Add cat to the list
- Clear properties after clicking >Save<

Add Command Binding

```
public class CatVM : AAnimal
{
    #region Properties
    string owner;
    public string Owner { ... }
    bool pussycat;
    public bool IsPussyCat { ... }
    ECatColor color;
    public ECatColor Color { ... }


    public ICommand SaveCommand { get; private set; }

    #endregion

    #region Methods
    public CatVM()
    {
        Path = @"..\..\..\cats.csv";
        SaveCommand = new CmdSave(this);


    }
```

```
public class CmdSave : ICommand{
    CatVM vm;
    public CmdSave(CatVM vm){
        this.vm = vm;
    }

    public bool CanExecute(object parameter){
        return (vm.Name != null
            && vm.Age != 0
            && vm.Owner != null);
    }

    public void Execute(object parameter) {
        CatVM cat = new CatVM() {
            Name = vm.Name, Owner = vm.Owner,
            Age = vm.Age, Color = vm.Color,
            IsDangerous = vm.IsDangerous,
            IsPussyCat = vm.IsPussyCat};
        CatVM.ToFile(cat);
        //reset values to default
        vm.Name = String.Empty;
        vm.Age = 0;
        vm.Owner = String.Empty;
        vm.IsDangerous = false;
        vm.IsPussyCat = false;
    }
    public event EventHandler CanExecuteChanged {
        add => CommandManager.RequerySuggested += value;
        remove => CommandManager.RequerySuggested -= value;
    }
}
```

```
<Window.DataContext>
    <local:CatVM/>
</Window.DataContext>
<TabControl >
    <TabItem Header="Katze">
        <StackPanel Orientation="Horizontal">
            <StackPanel Margin="20">
                <StackPanel Orientation="Horizontal"...>
                <StackPanel Orientation="Horizontal"...>
                <StackPanel Orientation="Horizontal"...>
                <StackPanel Orientation="Horizontal"...>
                <StackPanel Orientation="Horizontal"...>
                <Button Content="Speichern" Command="{Binding SaveCommand}" Margin="10,5"/>
            </StackPanel>
```

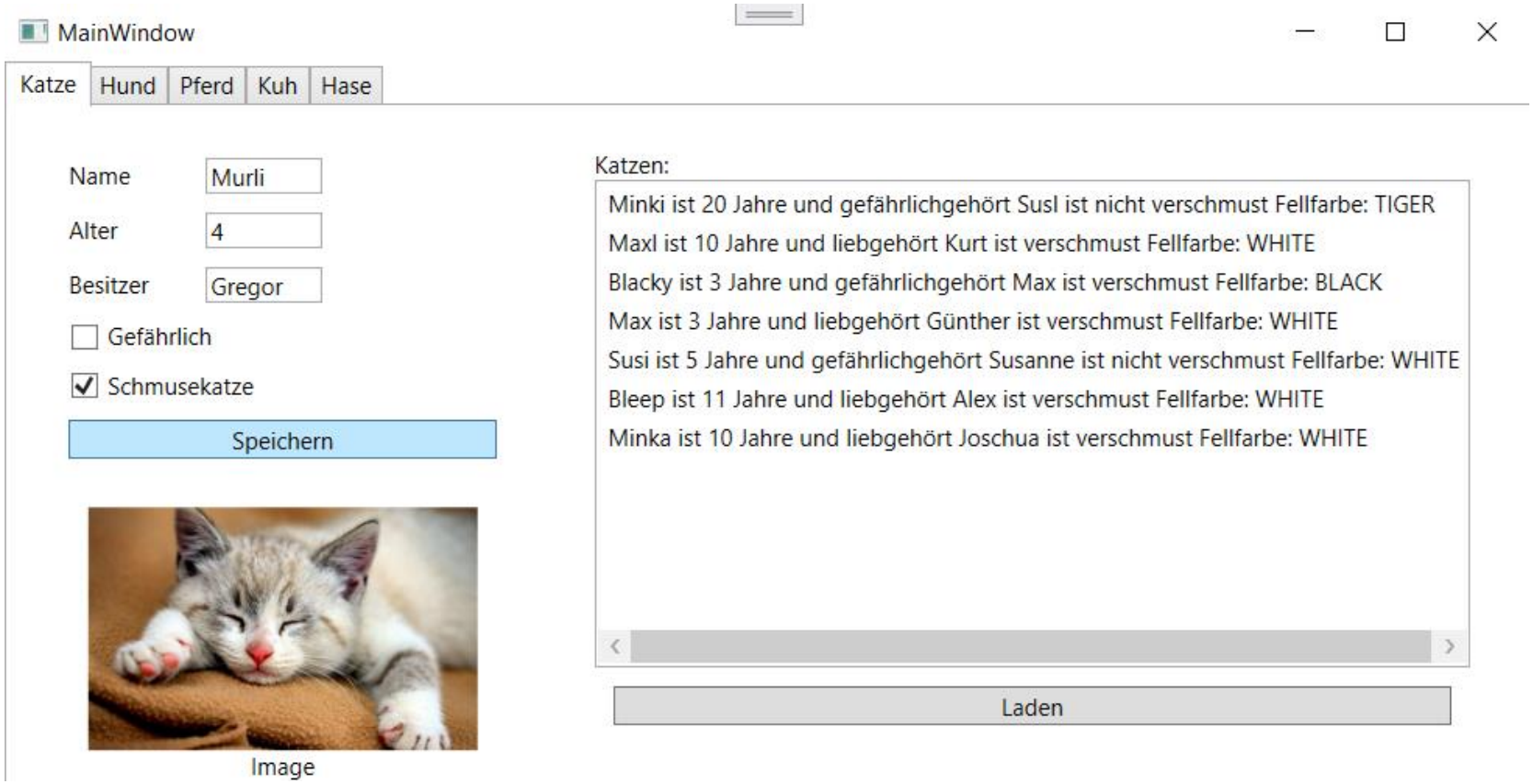
Add List Controls

```
public class CatVM : AAnimal
{
    #region Properties
    string owner;
    public string Owner { ... }
    bool pussycat;
    public bool IsPussyCat { ... }
    ECatColor color;
    public ECatColor Color { ... }

    public ObservableCollection<CatVM> List { get; private set; }
    public ICommand SaveCommand { get; private set; }
    public ICommand LoadCommand { get; private set; }
    #endregion

    #region Methods
    public CatVM()
    {
        Path = @"..\..\..\cats.csv";
        SaveCommand = new CmdSave(this);
        LoadCommand = new CmdLoad(this);
        List = new ObservableCollection<CatVM>();
    }
```

```
<StackPanel Margin="20" >
    <TextBlock Text="Katzen:"/>
    <ListBox ItemsSource="{Binding List}" Width="450" Height="250"/>
    <Button Content="Laden" Command="{Binding LoadCommand}" Margin="10"/>
</StackPanel>
```

```
public class CmdLoad : ICommand {
    private CatVM vm;
    public CmdLoad(CatVM vm){
        this.vm = vm;
    }
    public bool CanExecute(object parameter){
        return File.Exists(CatVM.Path); ;
    }
    public void Execute(object parameter) {
        vm.List.Clear();
        List<CatVM> cats = CatVM.FromFile();
        foreach (var item in cats)
        {
            vm.List.Add(item);
        }
    }
    public event EventHandler CanExecuteChanged {
        add => CommandManager.RequerySuggested += value;
        remove => CommandManager.RequerySuggested -= value;
    }
}
```

Test your application



Add your ENUMS

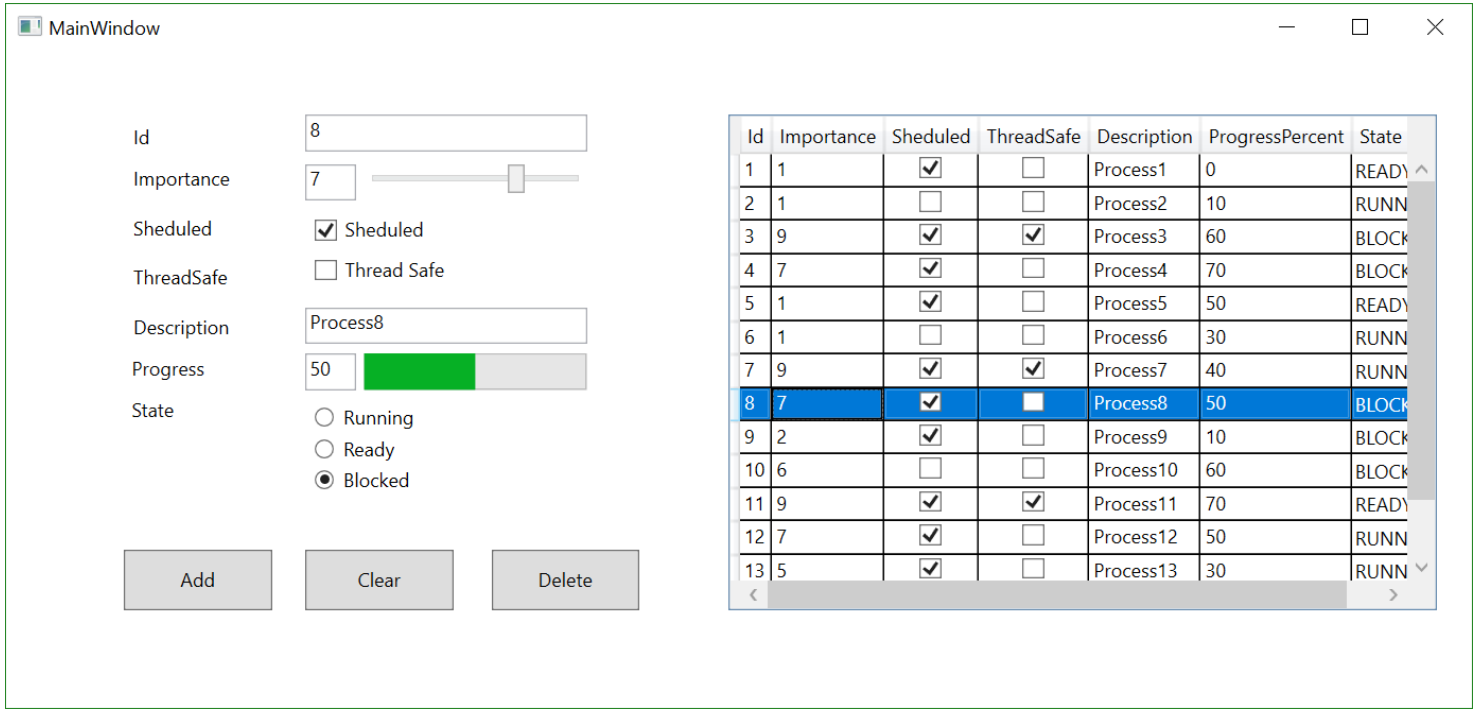
- using converter for different controls

- Slider: Int to Enum
- Radiobutton: Boolean to Enum
- Image: StringPath to Enum
- Textblock: String to Enum

Image

Black
White
Tiger

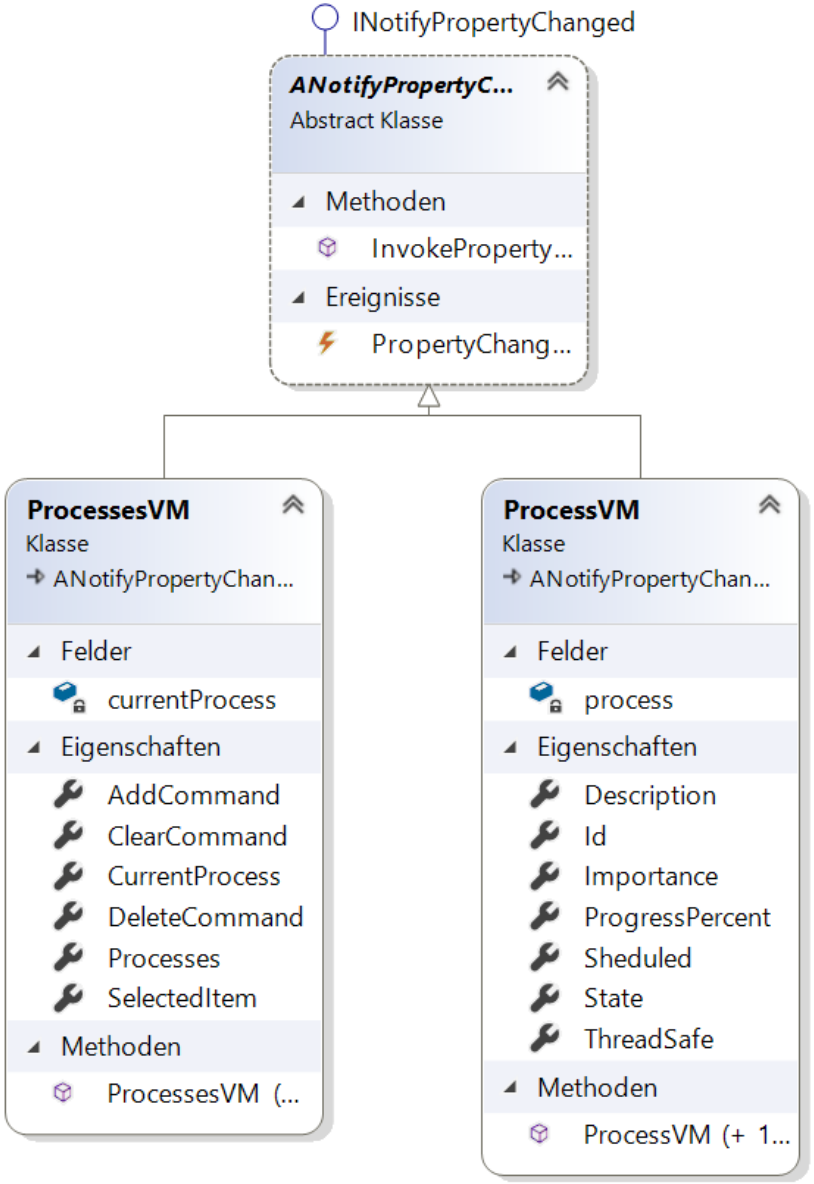
White

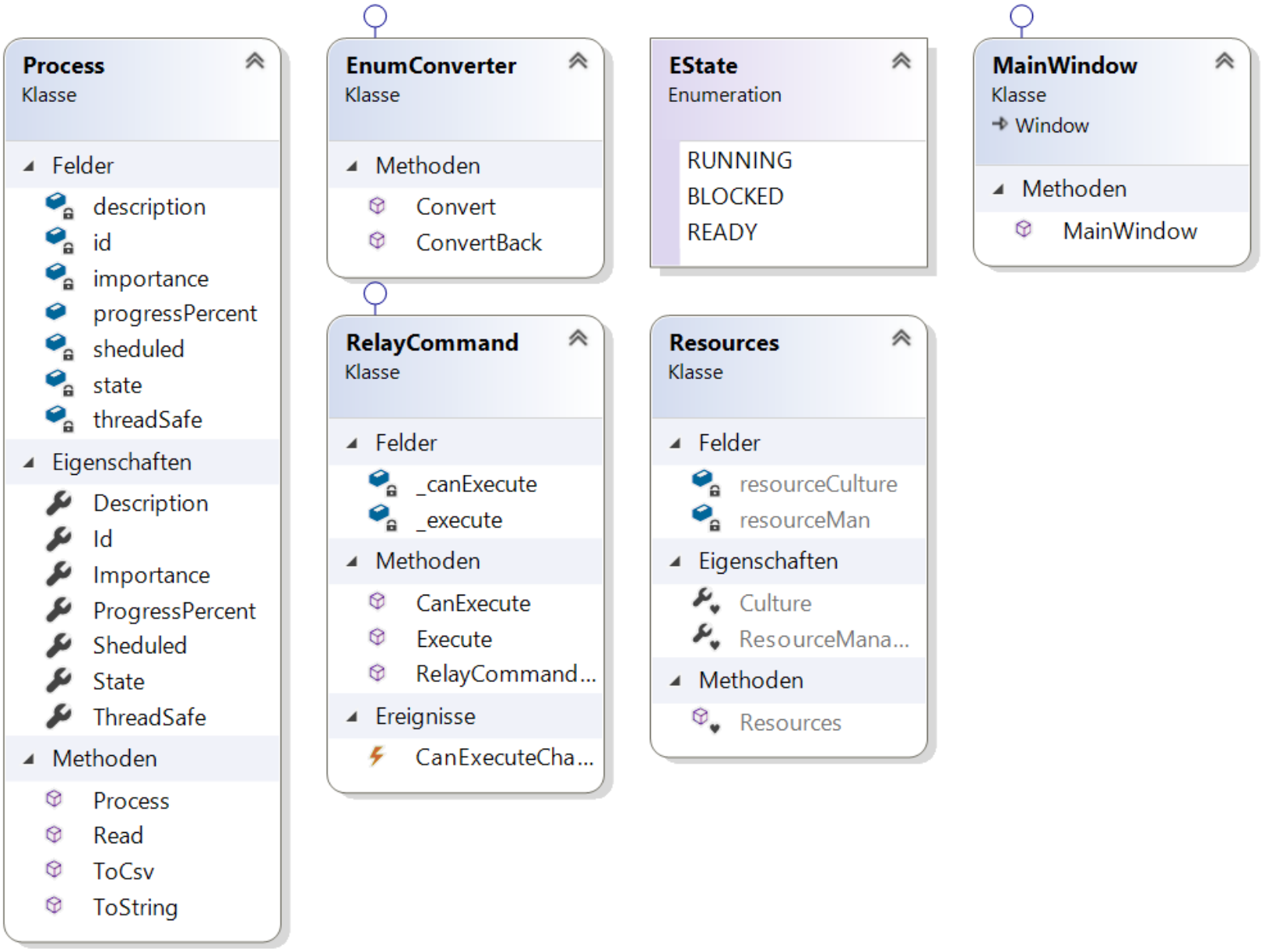


Process WPF

Load Processes from CSV, Select and Edit,
Clear Fields & Add new Process to List

ProcessVM & ProcessesVM




Process
Klasse
Felder
description
id
importance
progressPercent
sheduled
state
threadSafe
Eigenschaften
Description
Id
Importance
ProgressPercent
Sheduled
State
ThreadSafe
Methoden
Process
Read
ToCsv
ToString
EnumConverter
Klasse
Methoden
Convert
ConvertBack
RelayCommand
Klasse
Felder
_canExecute
_execute
Methoden
CanExecute
Execute
RelayCommand...
Ereignisse
CanExecuteCha...
EState
Enumeration
RUNNING
BLOCKED
READY
Resources
Klasse
Felder
resourceCulture
resourceMan
Eigenschaften
Culture
ResourceMana...
Methoden
Resources
MainWindow
Klasse
Window
Methoden
MainWindow

Process

```
public class Process
   {
       private int id;

       //Value [1-10]
       private int importance;
       private bool sheduled;
       private string description;
       private bool threadSafe;
       public int progressPercent;
       private EState state;


       public int Id { get => id; set => id = value; }
       public int Importance { get => importance; set => importance = value; }
       public bool Sheduled { get => sheduled; set => sheduled = value; }
       public bool ThreadSafe { get => threadSafe; set => threadSafe = value; }
       public string Description { get => description; set => description = value; }
       public int ProgressPercent { get => progressPercent; set => progressPercent = value; }
       public EState State { get => state; set => state = value; }

       public Process(int id, int importance, bool sheduled,
           bool threadsafe, string desc, int progress, EState state)
       {
           this.id = id;
           this.importance = importance;
           this.sheduled = sheduled;
           this.threadSafe = threadsafe;
           this.description = desc;
           this.progressPercent = progress;
           this.state = state;
       }
```

CSV Read Write

```
public string ToCsv()
{
    return $"{Id};{Importance};{Sheduled};{ThreadSafe};{Description};{ProgressPercent};{State};" ;
}
1-Verweis
public static List<Process> Read()
{
    List<Process> processes = new List<Process>();
    StreamReader sr = new StreamReader("process.csv");
    string line;
    string[] words;


    while ((line = sr.ReadLine()) != null)
    {
        words = line.Split(';');

        processes.Add(new Process(Convert.ToInt32(words[0]),
            Convert.ToInt32(words[1]), Convert.ToBoolean(words[2]),
            Convert.ToBoolean(words[3]),
            words[4], Convert.ToInt32(words[5]),
            (EState)Enum.Parse(typeof(EState), words[6])));
    }
    return processes;
}
```

XAML

Copy Code

```
<Grid>
        <Label Content="Id" HorizontalAlignment="Left" Margin="75,41,0,0" VerticalAlignment="Top"/>
        <TextBox HorizontalAlignment="Left" Height="23" Margin="188,40,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="177"/>

        <Label Content="Importance" HorizontalAlignment="Left" Margin="75,67,0,0" VerticalAlignment="Top"/>
        <TextBox HorizontalAlignment="Left" Height="23" Margin="188,71,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="32"/>
        <Slider  HorizontalAlignment="Left" Margin="225,71,0,0" VerticalAlignment="Top" Width="140" Height="22"/>


        <Label Content="ThreadSafe" HorizontalAlignment="Left" Margin="75,129,0,0" VerticalAlignment="Top"/>
        <CheckBox Content="Thread Safe" HorizontalAlignment="Left" Margin="193,130,0,0" VerticalAlignment="Top"/>
        <Label Content="Sheduled" HorizontalAlignment="Left" Margin="75,98,0,0" VerticalAlignment="Top"/>
        <CheckBox Content="Sheduled" HorizontalAlignment="Left" Margin="193,105,0,0" VerticalAlignment="Top"/>

        <Label Content="Description" HorizontalAlignment="Left" Margin="75,160,0,0" VerticalAlignment="Top"/>
        <TextBox HorizontalAlignment="Left" Height="23" Margin="188,161,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="177"/>

        <Label HorizontalAlignment="Left" Margin="95,234,0,0" VerticalAlignment="Top"/>

        <Label Content="Progress" HorizontalAlignment="Left" Margin="74,186,0,0" VerticalAlignment="Top"/>
        <ProgressBar HorizontalAlignment="Left" Height="23" Margin="225,190,0,0" VerticalAlignment="Top" Width="140"/>
        <TextBox HorizontalAlignment="Left" Height="23" Margin="188,190,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="32"/>

        <Label Content="State" HorizontalAlignment="Left" Margin="74,212,0,0" VerticalAlignment="Top"/>
        <RadioButton Content="Running" HorizontalAlignment="Left" Margin="193,223,0,0" VerticalAlignment="Top"/>
        <RadioButton Content="Ready" HorizontalAlignment="Left" Margin="193,243,0,0" VerticalAlignment="Top"/>
        <RadioButton Content="Blocked" HorizontalAlignment="Left" Margin="193,262,0,0" VerticalAlignment="Top"/>

        <Button Content="Add" HorizontalAlignment="Left" Height="38" Margin="74,313,0,0" VerticalAlignment="Top" Width="93"/>
        <Button Content="Clear" HorizontalAlignment="Left" Height="38" Margin="188,313,0,0" VerticalAlignment="Top" Width="93" />
        <Button Content="Delete" HorizontalAlignment="Left" Height="38" Margin="305,313,0,0" VerticalAlignment="Top" Width="93"/>
        <DataGrid HorizontalAlignment="Left" Height="311" Margin="454,40,0,0" VerticalAlignment="Top" Width="445" />

    </Grid>
```

Solution

```
class ProcessVM : ANotifyPropertyChanged
{
    private Process process;

    public int Id{
        get => this.process.Id;
        set { this.process.Id = value; InvokePropertyChanged(); }
    }
    public int Importance{
        get => this.process.Importance;
        set { this.process.Importance = value; InvokePropertyChanged();}
    }
    public bool Sheduled{
        get => this.process.Sheduled;
        set{ this.process.Sheduled = value; InvokePropertyChanged();}
    }
    public bool ThreadSafe{
        get => this.process.ThreadSafe;
        set{ this.process.ThreadSafe = value; InvokePropertyChanged(); }
    }
    public string Description
    {
        get => this.process.Description;
        set{this.process.Description = value;InvokePropertyChanged();}
    }
    public int ProgressPercent{
        get => this.process.ProgressPercent;
        set{this.process.ProgressPercent = value;InvokePropertyChanged();}
    }
    public EState State{
        get => this.process.State;
        set{this.process.State = value;InvokePropertyChanged();}
    }

    public ProcessVM() : this(  new Process(0, 0, false, false, null, 0, EState.BLOCKED)) { }
    public ProcessVM(Process process){
        this.process = process;
    }
}
```

ProcessesVM

```
class ProcessesVM : ANotifyPropertyChanged
{
    public ObservableCollection<ProcessVM> Processes { get; private set; }

    public RelayCommand AddCommand...
    public RelayCommand ClearCommand...
    public RelayCommand DeleteCommand...

    public ProcessVM SelectedItem...
    private ProcessVM currentProcess = new ProcessVM();
    public ProcessVM CurrentProcess...

    public ProcessesVM() : this(Process.Read()) { }
    public ProcessesVM(IEnumerable<Process> processes)...
}
```

Selected Item vs CurrentProcess

```
public ProcessVM SelectedItem
{
    get => Processes.Contains(CurrentProcess) ? CurrentProcess : null;
    set => CurrentProcess = value == null ? new ProcessVM() : value;
}
private ProcessVM currentProcess = new ProcessVM();
public ProcessVM CurrentProcess
{
    get => this.currentProcess;
    set
    {
        this.currentProcess = value;
        InvokePropertyChanged();
        InvokePropertyChanged("SelectedItem");
    }
}
```

Change one Item of a list, the OC doesn‘t get that something changed, therefore use a OC<T> where T:INotifyPropertyChanged to recognize the change

Solution

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

    public void InvokePropertyChanged([CallerMemberName] string property = null)
        => PropertyChanged(this, new PropertyChangedEventArgs(property));
}
```

```
class ProcessesVM : ANotifyPropertyChanged
{
    public ObservableCollection<ProcessVM> Processes { get; private set; }

    public RelayCommand AddCommand
        => new RelayCommand(
            o => {
                Processes.Add(CurrentProcess);
                SelectedItem = null;
            }
        );
    public RelayCommand ClearCommand
        => new RelayCommand(
            o => SelectedItem = null
        );
    public RelayCommand DeleteCommand
        => new RelayCommand(
            o => Processes.Remove(CurrentProcess)
        );

    public ProcessVM SelectedItem
    {
        get => Processes.Contains(CurrentProcess) ? CurrentProcess : null;
        set => CurrentProcess = value == null ? new ProcessVM() : value;
    }
    private ProcessVM currentProcess = new ProcessVM();
    public ProcessVM CurrentProcess
    {
        get => this.currentProcess;
        set
        {
            this.currentProcess = value;
            InvokePropertyChanged();
            InvokePropertyChanged("SelectedItem");
        }
    }

    public ProcessesVM() : this(Process.Read()) { }
    public ProcessesVM(IEnumerable<Process> processes)
    {
        Processes = new ObservableCollection<ProcessVM>(
            processes.Select(p => new ProcessVM(p))
        );
    }
}
```

BoolEnumConverter

```
public class EnumConverter : IValueConverter
{
    public object Convert(object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        return ((EState)value) == ((EState)parameter);
    }

    public object ConvertBack(object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        if ((bool)value)
            return (EState)parameter;
        else
            return Binding.DoNothing;
    }
}
```

Binding UI Element

```
<Label Content="Id" HorizontalAlignment="Left" Margin="75,41,0,0" VerticalAlignment="Top"/>
<TextBox HorizontalAlignment="Left" Height="23" Margin="188,40,0,0" TextWrapping="Wrap"
        Text="{Binding CurrentProcess.Id}" VerticalAlignment="Top" Width="177"/>

<Label Content="Importance" HorizontalAlignment="Left" Margin="75,67,0,0" VerticalAlignment="Top"/>
<TextBox HorizontalAlignment="Left" Height="23" Margin="188,71,0,0" TextWrapping="Wrap"
        Text="{Binding CurrentProcess.Importance}" VerticalAlignment="Top" Width="32"/>
<Slider HorizontalAlignment="Left" Margin="225,71,0,0" VerticalAlignment="Top" Width="140" Height="22"
        Value="{Binding CurrentProcess.Importance}"/>


<Label Content="ThreadSafe" HorizontalAlignment="Left" Margin="75,129,0,0" VerticalAlignment="Top"/>
<CheckBox Content="Thread Safe" HorizontalAlignment="Left" Margin="193,130,0,0" VerticalAlignment="Top"
         IsChecked="{Binding CurrentProcess.ThreadSafe}"/>
<Label Content="Sheduled" HorizontalAlignment="Left" Margin="75,98,0,0" VerticalAlignment="Top"/>
<CheckBox Content="Sheduled" HorizontalAlignment="Left" Margin="193,105,0,0" VerticalAlignment="Top"
         IsChecked="{Binding CurrentProcess.Sheduled}"/>

<Label Content="Description" HorizontalAlignment="Left" Margin="75,160,0,0" VerticalAlignment="Top"/>
<TextBox HorizontalAlignment="Left" Height="23" Margin="188,161,0,0" TextWrapping="Wrap"
        Text="{Binding CurrentProcess.Description}" VerticalAlignment="Top" Width="177"/>

<Label HorizontalAlignment="Left" Margin="95,234,0,0" VerticalAlignment="Top"/>
```

Converter Binding

```
<Window.DataContext>
    <local:ProcessesVM />
</Window.DataContext>
<Window.Resources>
    <local:EnumConverter x:Key="EnumConverter"/>
</Window.Resources>
```

```
<RadioButton Content="Running" HorizontalAlignment="Left" Margin="193,223,0,0" VerticalAlignment="Top">
    <RadioButton.IsChecked>
        <Binding Path="CurrentProcess.State"
                 Converter="{StaticResource EnumConverter}"
                 ConverterParameter="{x:Static local:EState.RUNNING}" />
    </RadioButton.IsChecked>
</RadioButton>
<RadioButton Content="Ready" HorizontalAlignment="Left" Margin="193,243,0,0" VerticalAlignment="Top">
    <RadioButton.IsChecked>
        <Binding Path="CurrentProcess.State"
                 Converter="{StaticResource EnumConverter}"
                 ConverterParameter="{x:Static local:EState.READY}" />
    </RadioButton.IsChecked>
</RadioButton>
<RadioButton Content="Blocked" HorizontalAlignment="Left" Margin="193,262,0,0" VerticalAlignment="Top">
    <RadioButton.IsChecked>
        <Binding Path="CurrentProcess.State"
                 Converter="{StaticResource EnumConverter}"
                 ConverterParameter="{x:Static local:EState.BLOCKED}" />
    </RadioButton.IsChecked>
</RadioButton>
```

ICommand & RelayCommand...

```
public class RelayCommand : ICommand
{
    readonly Func<Boolean> _canExecute;
    readonly Action _execute;

    0 Verweise
    public RelayCommand(Action execute) : this(execute, null) { }
    2 Verweise
    public RelayCommand(Action execute, Func<Boolean> canExecute) {
        if (execute == null)
            throw new ArgumentNullException("execute");
        _execute = execute;
        _canExecute = canExecute;
    }
    public event EventHandler CanExecuteChanged...
    2 Verweise
    public Boolean CanExecute(Object parameter) {
        return _canExecute == null ? true : _canExecute();
    }
    2 Verweise
    public void Execute(Object parameter){
        _execute();
    }
}
```

CanExecute

- if something changes, the button has to activate and deactivate itself
- therefore the Eventhandler CanExecuteChanged has to add or remove a value

```
public event EventHandler CanExecuteChanged {
    add => CommandManager.RequerySuggested += value;
    remove => CommandManager.RequerySuggested -= value;
}
```

Using RelayCommand

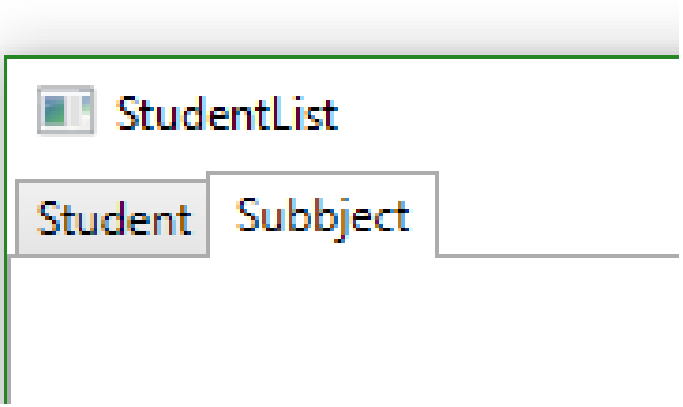
ICommand
as
Returntype

would be
even better
…

```
public RelayCommand AddCommand
    => new RelayCommand(
        o => {
            Processes.Add(CurrentProcess);
            SelectedItem = null;
        }
    );
public RelayCommand ClearCommand
    => new RelayCommand(
        o => SelectedItem = null
    );
public RelayCommand DeleteCommand
    => new RelayCommand(
        o => Processes.Remove(CurrentProcess)
    );
```

Command Binding

```
<Button Content="Add"
        Command="{Binding AddCommand}"/>
<Button Content="Clear"
        Command="{Binding ClearCommand}" />
<Button Content="Delete"
        Command="{Binding DeleteCommand}"/>
<DataGrid Name="dgrProcesses"
          ItemsSource="{Binding Processes}" IsReadOnly="True"
          SelectedItem="{Binding SelectedItem}" />
```

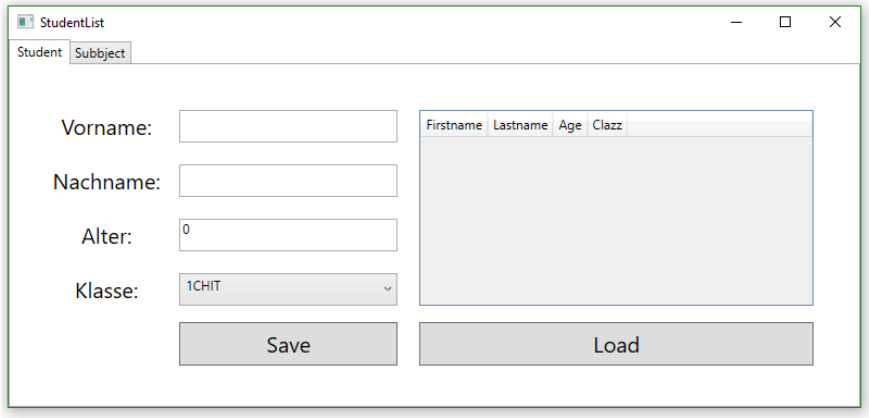


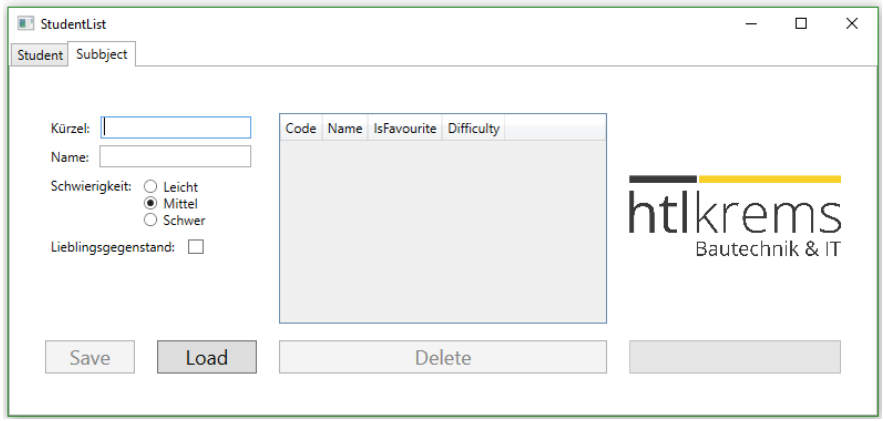
Student & Subject Example

Use a TabControl to manage Students & Subjects

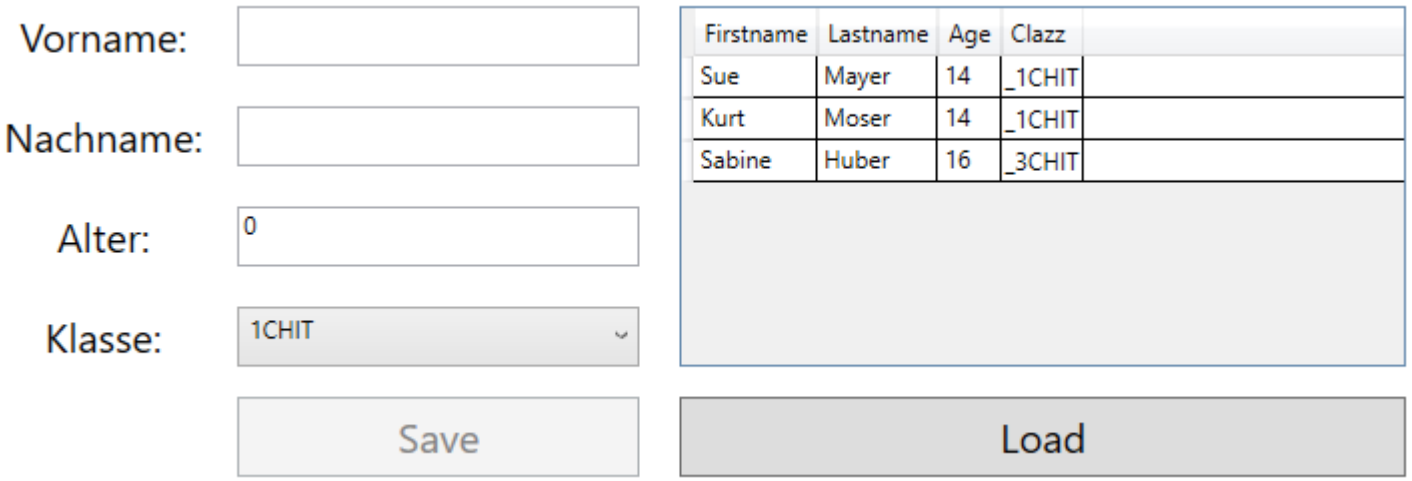
TabControl

- TabItem Student
  - Set the DataContext for TabItem Student to StudentVM
- TabItem Subject
  - Set the DataContext for TabItem Subject to SubjectVM






```
<Grid>
    <TabControl  >
        <TabItem Header="Student" DataContext="StudentViewModel">
            <Grid Margin="20 30" ...>
        </TabItem>
        <TabItem Header="Subbject" DataContext="SubjectViewModel">
            <Grid Margin="20 30"...>
        </TabItem>
    </TabControl>
</Grid>
```



Studenten

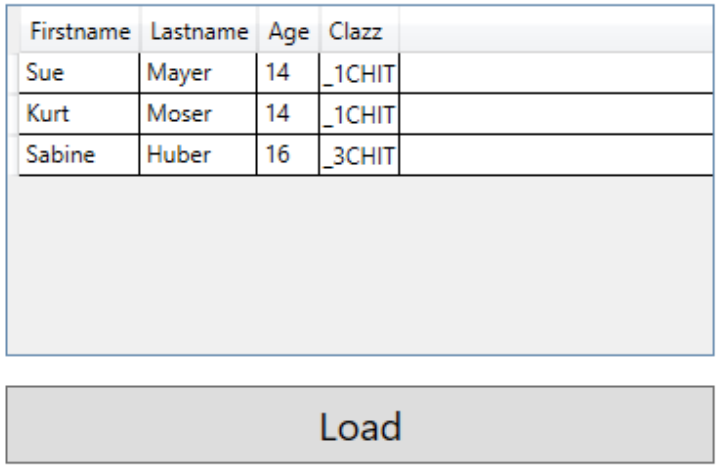
Label, TextBox, Button, ComboBox, ListBox

Student

- Create a new Form, enter FirstName, LastName, Age and his/her Class like 1Bhit, 2Bhit, 3Bhit, …
- Use Label and TextBox for the input
- Use an Enum & a ComboBox for the Class

- Save all Students in an ObservableCollection
- Show all Students in a DataGrid

Student – Load from csv file



- Use a ObservableCollection „Students“ in the ViewModel
  - Add a Student
- Save in a csv-file
  - Use a to csv-method in the student
- Load the csv-file
  - Load all students from the csv to a the ObservableCollection „Students“

```
<Button Grid.Column="1" Grid.Row="4" Margin="0 5" FontSize="20"
        Command="{Binding student.AddCommand}">Save</Button>
<DataGrid Grid.Column="2" Grid.RowSpan="4" Margin="20 10" CanUserAddRows="False"
          ItemsSource="{Binding student.StudentCollection, Mode=OneWay}" />
<Button Grid.Column="2" Grid.Row="4" Margin="20 5" FontSize="20"
        Command="{Binding student.LoadCommand}">Load</Button>
```

StudentViewModel - Command

```
public class ViewModel : INotifyPropertyChanged
{




    public ICommand AddCommand { get; private set; }
    public ICommand LoadCommand { get; private set; }

    public ViewModel()
    {


        AddCommand = new AddCommand(this);
        LoadCommand = new LoadCommand(this);
    }
```

LoadCommand

```
public ICommand AddCommand { get; private set; }
public ICommand LoadCommand { get; private set; }

public ViewModel()
{
    AddCommand = new AddCommand(this);
    LoadCommand = new LoadCommand(this);
```

```
class LoadCommand:ICommand
{
    ViewModel parent;
    public event EventHandler CanExecuteChanged;

    public LoadCommand(ViewModel parent)...

    public bool CanExecute(object parameter)
    {
        return true;
    }

    public void Execute(object parameter)
    {
        parent.LoadFromCSV();
    }
}
```

```
<Button Grid.Column="2" Grid.Row="4" Margin="20 5" FontSize="20"
        Command="{Binding student.LoadCommand}">Load</Button>
```

Class Student

```
public enum EClass {_1CHIT, _2CHIT, _3CHIT, _4CHIT, _5CHIT }

public class Student
{
    public string Firstname { get; set; }
    public string Lastname { get; set; }
    public int Age { get; set; } = 0;
    public EClass Clazz { get; set; }

    public Student(string firstname, string lastname, int age, EClass clazz)
    {
        this.Firstname = firstname;
        this.Lastname = lastname;
        this.Age = age;
        this.Clazz = clazz;
    }
    public Student(){  }
    public string ToCsv()...
    public void LoadFromCSVLine()...
}
```

- Properties & Constructor
- ToCsv / ReadCsv

Access a csv file

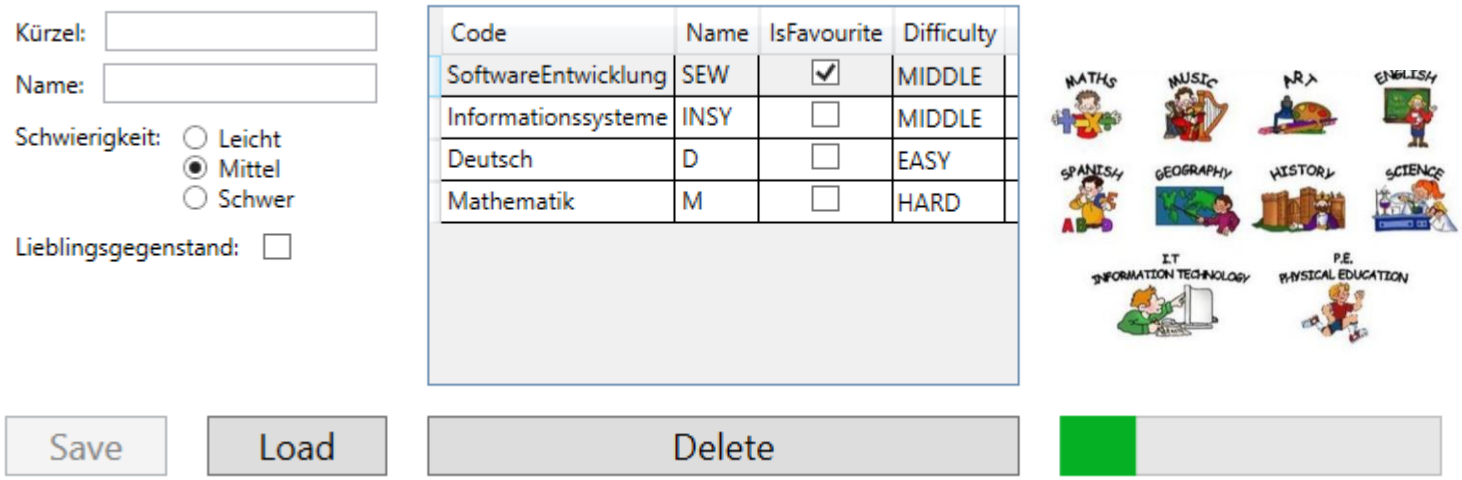
```
public void Save(string path, Student s)
{
    StreamWriter sw = new StreamWriter("students.csv", true);
    sw.WriteLine(s.ToCSV());
    sw.Close();
}

public List<Student> Read(string path)
{
    StreamReader sr = new StreamReader(path);
    List<Student> students = new List<Student>();
    string line;
    while ((line = sr.ReadLine()) != null)
    {
        string[] splittedLine = line.Split(';');
        students.Add(new Student(splittedLine[0],
            splittedLine[1],
            Convert.ToInt32(splittedLine[2]),
            (Classes)Enum.Parse(typeof(Classes),
            splittedLine[3])));
    }
    sr.Close();
    return students;
}
```

Add File.Exits(Path) condition to CanExecute to the Load Method,

can only be executed, if the File exists

Student ViewModel

```
public class StudentViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;
    public ObservableCollection<Student> StudentCollection { get; private set; }
    private Student currentStudent;
    public string Path { get; set; } = "Students.csv";

    public ICommand AddCommand { get; private set; }
    public ICommand LoadCommand { get; private set; }

    public StudentViewModel()...

    public string CurrentFirstname...

    public string CurrentLastname...

    public int CurrentAge...

    public int CurrentClazz...

    public Student CurrentStudent...

    private void OnPropertyChanged(string Property)...

    public void SaveInCSV()...

    public void LoadFromCSV()...
}
```



Manage Subjects

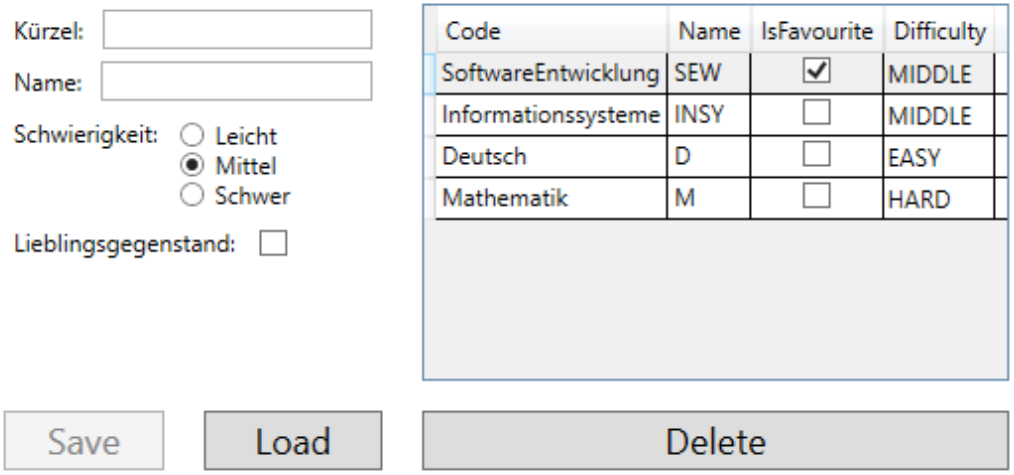
DataGrid, RadioButton, CheckBox, PictureBox

Subject

```
public enum EDifficulty { EASY, MIDDLE, HARD }

public class Subject
{
    public string Code { get; set; }
    public string Name { get; set; }
    public bool IsFavourite { get; set; }
    public EDifficulty Difficulty { get; set; }

    public Subject()
    {
        Code = "";
        Name = "";
        IsFavourite = false;
        Difficulty = EDifficulty.MIDDLE;
    }

    public string ToCsv()...
    public void ReadFromCSV()...
}
```

ObservableCollection Subjects

- Save Subject to a ObservableCollection
- Delete selected Subject from Collection



- Save all Subjects to csv file
- Load all Subjects from csv file

Add Picture & ProgressBar

- PictureBox: SchoolLogo

- Add a ProgressBar
  - Amount of saved Subjects (max 20)

Read Subjects

```
public static List<Subject> ReadSubjects(string path)
{
    List<Subject> subjects = new List<Subject>();
    StreamReader sr = new StreamReader(path);
    string line;
    while ((line = sr.ReadLine()) != null)
    {
        string[] splittedLine = line.Split(';');
        subjects.Add(new Subject(splittedLine[0],
            splittedLine[1],
            (EDifficulty)Enum.Parse(typeof(EDifficulty),
            splittedLine[2]),
            Convert.ToBoolean(splittedLine[3])));
    }
    sr.Close();
    return subjects;

}
```

Save Subjects

```
public static void WriteSubjects(List<Subject> subjects, string path) {
    StreamWriter sw = new StreamWriter(path);
    foreach (Subject subject in subjects) {
        sw.WriteLine(subject.ToCSV());
    }
    sw.Close();
}

private string ToCSV() {
    return $"{this.Contraction};{this.Name};{(int)this.Difficulty};{this.FavoriteSubject}";
}

public override string ToString()
{
    StringBuilder sb = new StringBuilder();
    sb.Append($"{Name} ({Contraction}) is {Difficulty.ToString().ToLower()} and ");
    if (!FavoriteSubject) { sb.Append("not ");}
    sb.Append("one of my favorite subjects");
    return sb.ToString();
}
```

SubjectViewModel

```
public class SubjectViewModel:INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;
    private ObservableCollection<Subject> subjectCollection;
    private Subject current;
    public ICommand SaveCommand { get; set; }
    public ICommand DeleteCommand { get; set; }
    public ICommand LoadCommand { get; set; }
    private int selectedIndex;
    public string Path { get; set; }

    public SubjectViewModel()...
    private void OnPropertyChanged([CallerMemberName]string Property = null)...
    public ObservableCollection<Subject> SubjectCollection...
    public string CurrentCode...
    public string CurrentName...
    public bool CurrentFavourite...
    public bool CurrentEasy...
    public bool CurrentMiddle...
    public bool CurrentHard...
    public Subject Current...
    public void AddCurrent()...
    public int CollectionCount...
    public int SelectedIndex...
    public void SaveAsCSV()...
    public void LoadFromCSV()...
    public void RemoveAt(int index)...
}
```