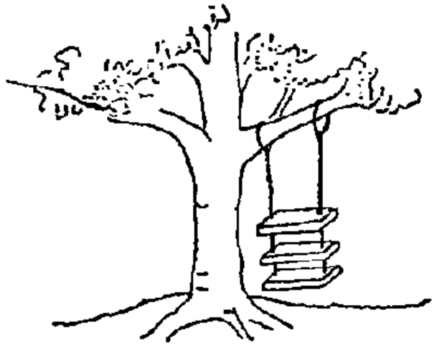
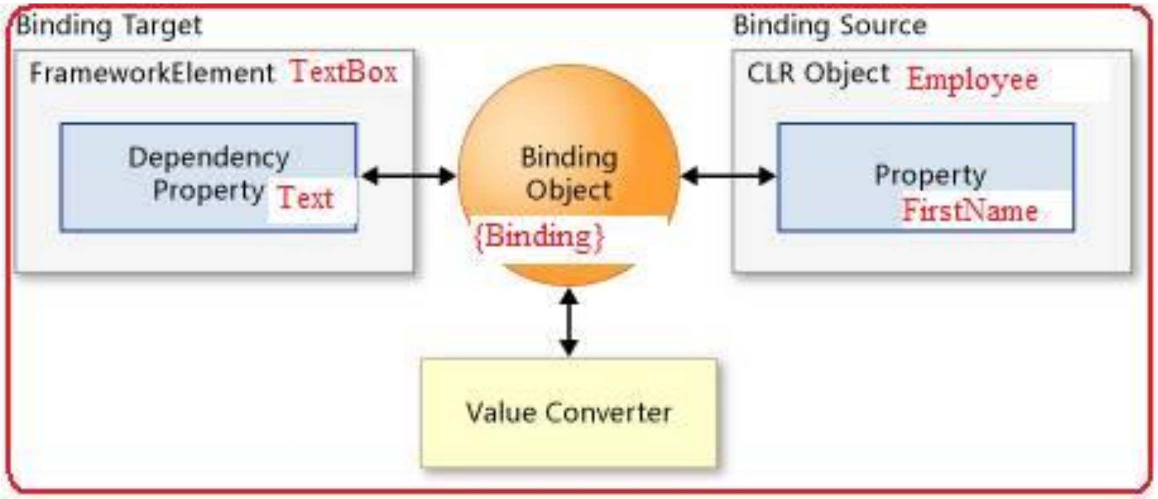
WPF Data Binding

Software Entwicklung



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/

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, typically in a collection
- ‘Bind' your XAML to the data
  - `<Label Content="{Binding Name}" />`

Implement INotifyPropertyChanged

- PropertyChangedEventHandler

```
public event PropertyChangedEventHandler PropertyChanged;
```

- [CallerMemberName]

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

Key Points you need to know

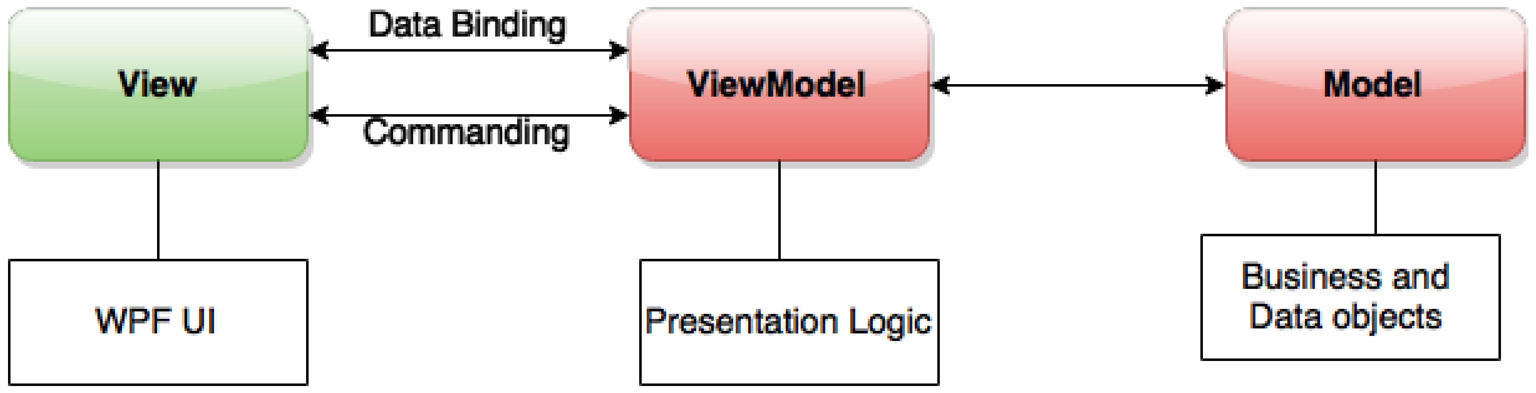
- Interface 'INotifyPropertyChanged’
  - Used to communicate any changes in the data between the GUI and your code
- Use ObservableCollection<>
  - not a List or Dictionary
  - WPF window needs to be able to 'observe' your data
  - WPF controls (including 'Window's) have a 'DataContext’
  - Collection controls have an 'ItemsSource' attribute to bind to

MVVM

Model View ViewModel
like MVC

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

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));
    }
}
```

Observable Collection

- Update an object of your list
  - use ConcreteVM in Observable Collections

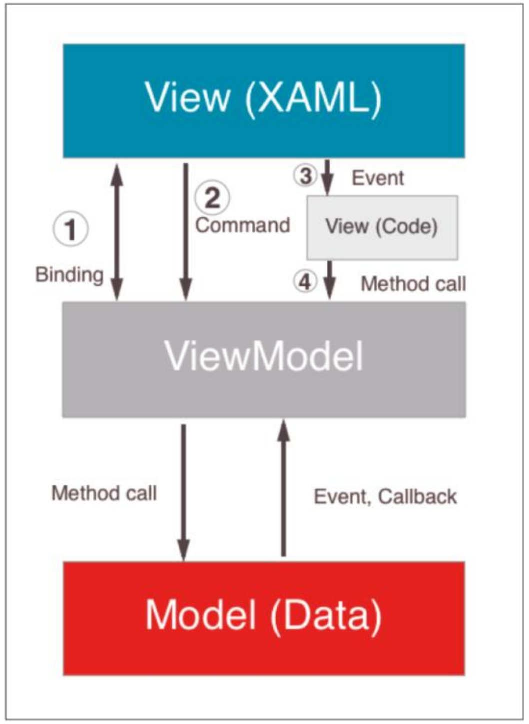
```
class PersonsVM
{
    public ObservableCollection<PersonVM> People { get;
        private set; }

    public PersonsVM(List<Person> people)
    {
        this.People = new ObservableCollection<PersonVM>(
            people.Select(p => new PersonVM(p))
        );
    }
}
```

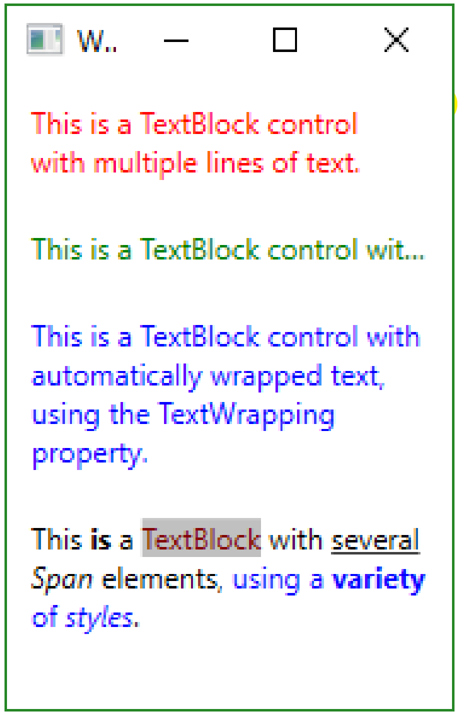
StudentViewModel

```
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();
        }
    }
}
```

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





This is a TextBlock control
with multiple lines of text.

This is a TextBlock control wit...

This is a TextBlock control with automatically wrapped text, using the TextWrapping property.

This **is** a TextBlock with <u>several</u> *Span* elements, using a **variety** of *styles*.

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/

Data Binding on TextBlock

- Add DataContext in Window or Control:

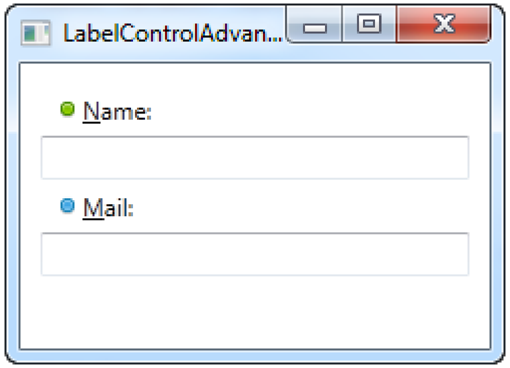
```
<Window.DataContext>
    <local:SkillVM/>
</Window.DataContext>
```

  - Set Binding:

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

  - Set Context and Binding in the Control:

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



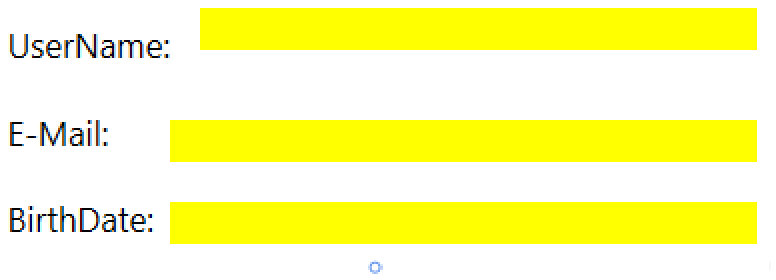
Labels

Content instead of Text Property

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

Label Data Binding

- DataContext auf UserVM setzen
- Label 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"
```

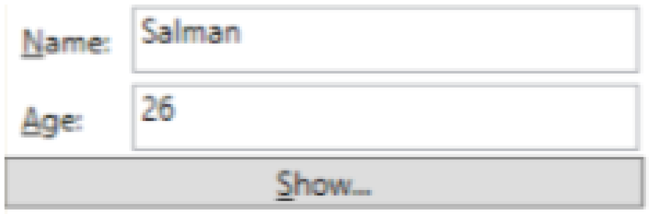
TextBox

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/

Bind Name & Age

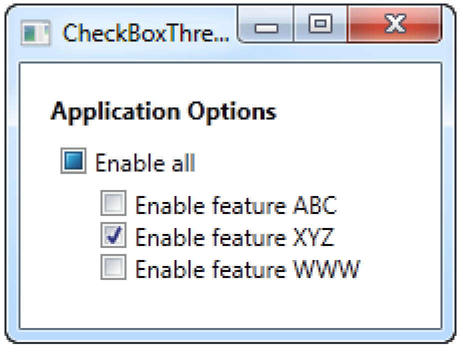


- One Way Binding

```
<Label Name = "nameLabel" Margin = "2">_Name:</Label>

<TextBox Name = "nameText" Grid.Column = "1" Margin = "2"
   Text = "{Binding Name, Mode = OneWay}"/>

<Label Name = "ageLabel" Margin = "2" Grid.Row = "1">_Age:</Label>

<TextBox Name = "ageText" Grid.Column = "1" Grid.Row = "1" Margin = "2"
   Text = "{Binding Age, Mode = OneWay}"/>
```

- Two Way Binding

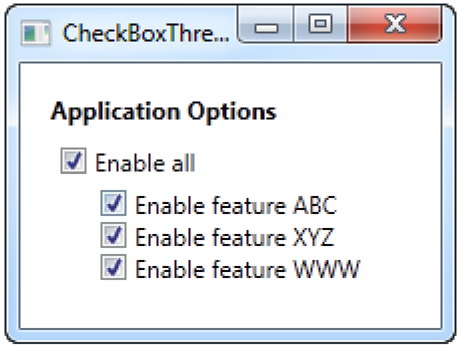
```
<Label Name = "nameLabel" Margin = "2">_Name:</Label>
<TextBox Name = "nameText" Grid.Column = "1" Margin = "2"
   Text = "{Binding Name, Mode = TwoWay}"/>
<Label Name = "ageLabel" Margin = "2" Grid.Row = "1">_Age:</Label>
<TextBox Name = "ageText" Grid.Column = "1" Grid.Row = "1" Margin = "2"
   Text = "{Binding Age, Mode = TwoWay}"/>
```


CheckBoxThre...

Application Options

■ Enable all

- ☐ Enable feature ABC
- [✓] Enable feature XYZ
- ☐ Enable feature WWW


CheckBoxThre...

Application Options

[✓] Enable all

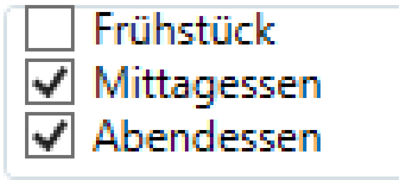
- [✓] Enable feature ABC
- [✓] Enable feature XYZ
- [✓] Enable feature WWW

CheckBox

Select one or multiple choices

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

CheckBox Binding

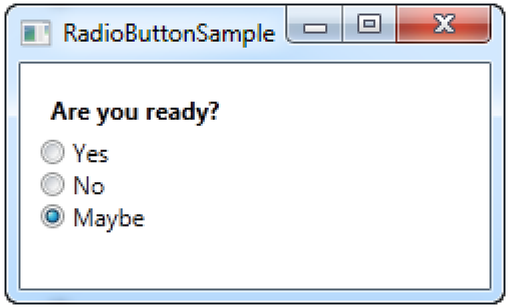


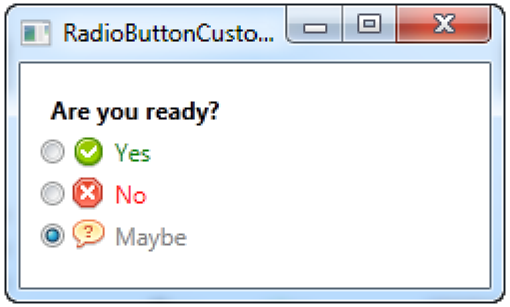
ViewModel

```
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>
```





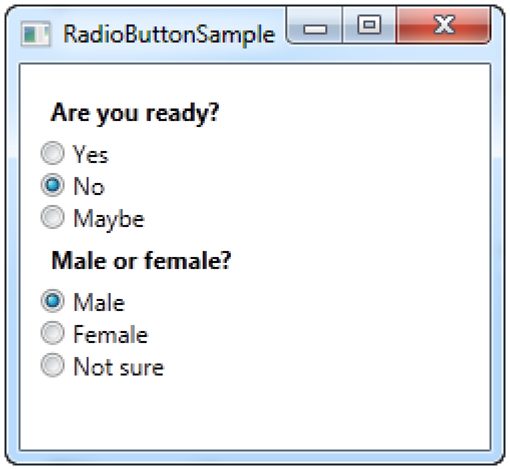
RadioButton

allows you to give your user a list of possible options

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

Radio Button Groups

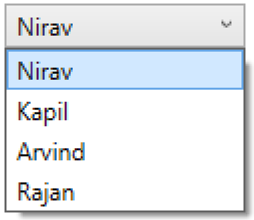
- **GroupName** property allows to specify which radio buttons belong together



Are you ready?
- Yes
- [✓ No]
- Maybe

Male or female?
- [✓ Male]
- Female
- Not sure

```
<Window x:Class="WpfTutorialSamples.Basic_controls.RadioButtonSample"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="RadioButtonSample" Height="230" Width="250">
        <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>
</Window>
```



```
ItemsSource="{Binding Path=Persons}"
SelectedItem="{Binding Path=SPerson}"
DisplayMemberPath="Name"
```

ComboBox

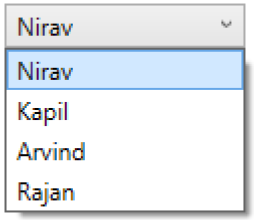
like the ListBox control, but takes up a lot less space, because the list of items is hidden when not needed

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

Binding ComboBox

- ViewModel – Personlist & SelectedPerson

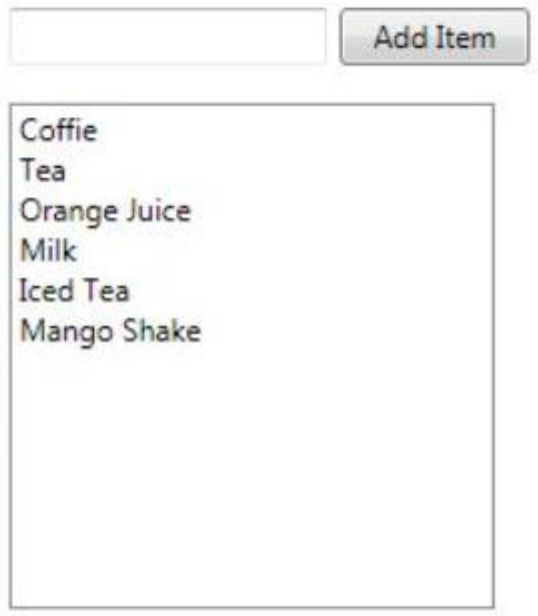
```
private ObservableCollection<Person> _persons;

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



Coffie
Tea
Orange Juice
Milk
Iced Tea
Mango Shake

Listbox

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

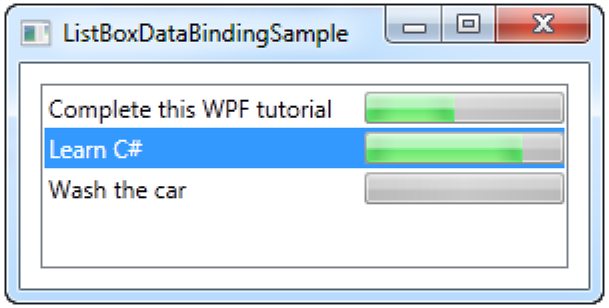
https://wpf-tutorial.com/de/72/listen-steuerelemente/die-listbox/

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

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
- UpdateSourceTrigger:
    - Default, **PropertyChanged**, **LostFocus** and
    - **Explicit**
        - the update has to be pushed manually through to occur,
          using a call to UpdateSource on the Binding

ListBox Binding



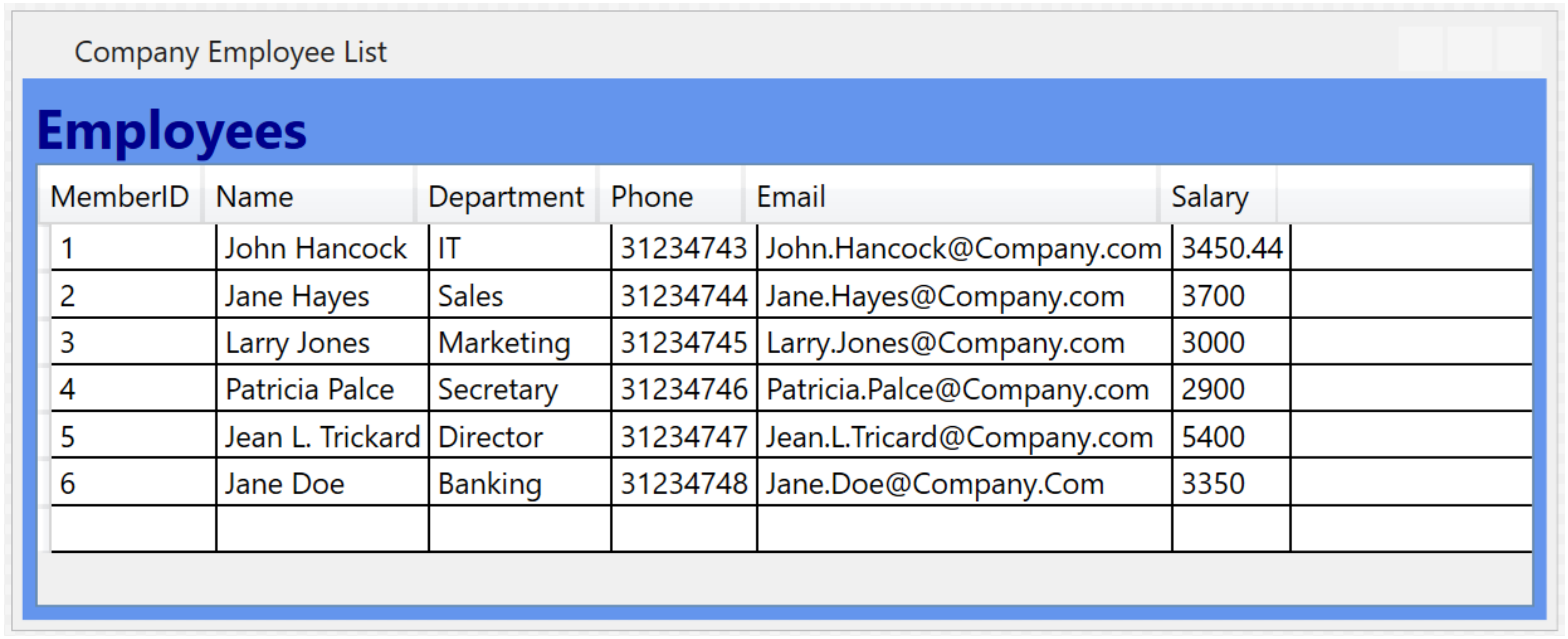
Complete this WPF tutorial
Learn C#
Wash the car

```
<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}" />
                </Grid>
            </DataTemplate>
        </ListBox.ItemTemplate>
    </ListBox>
```

DataGrid

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

DataGrid Employee

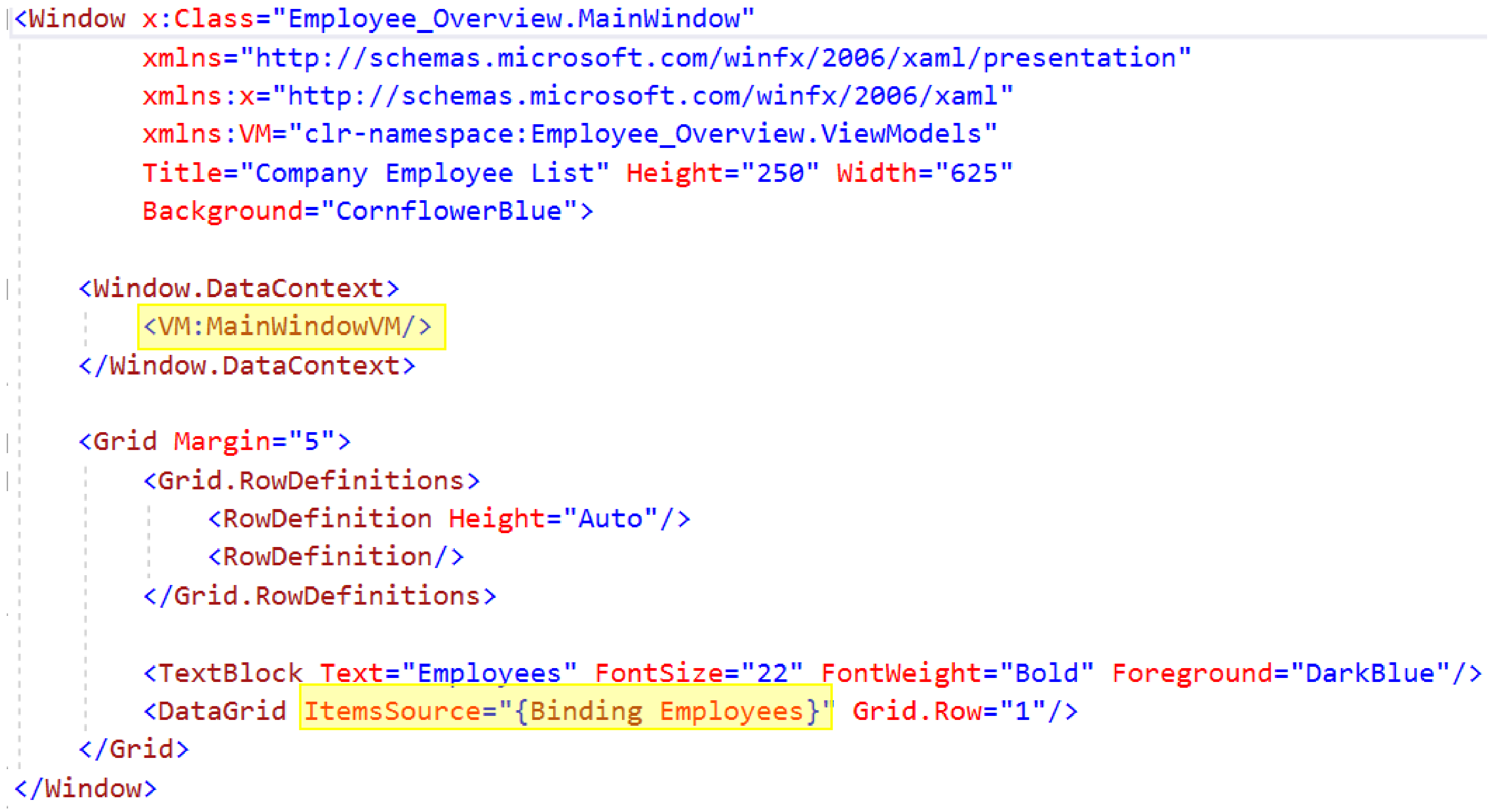


Company Employee List

Employees

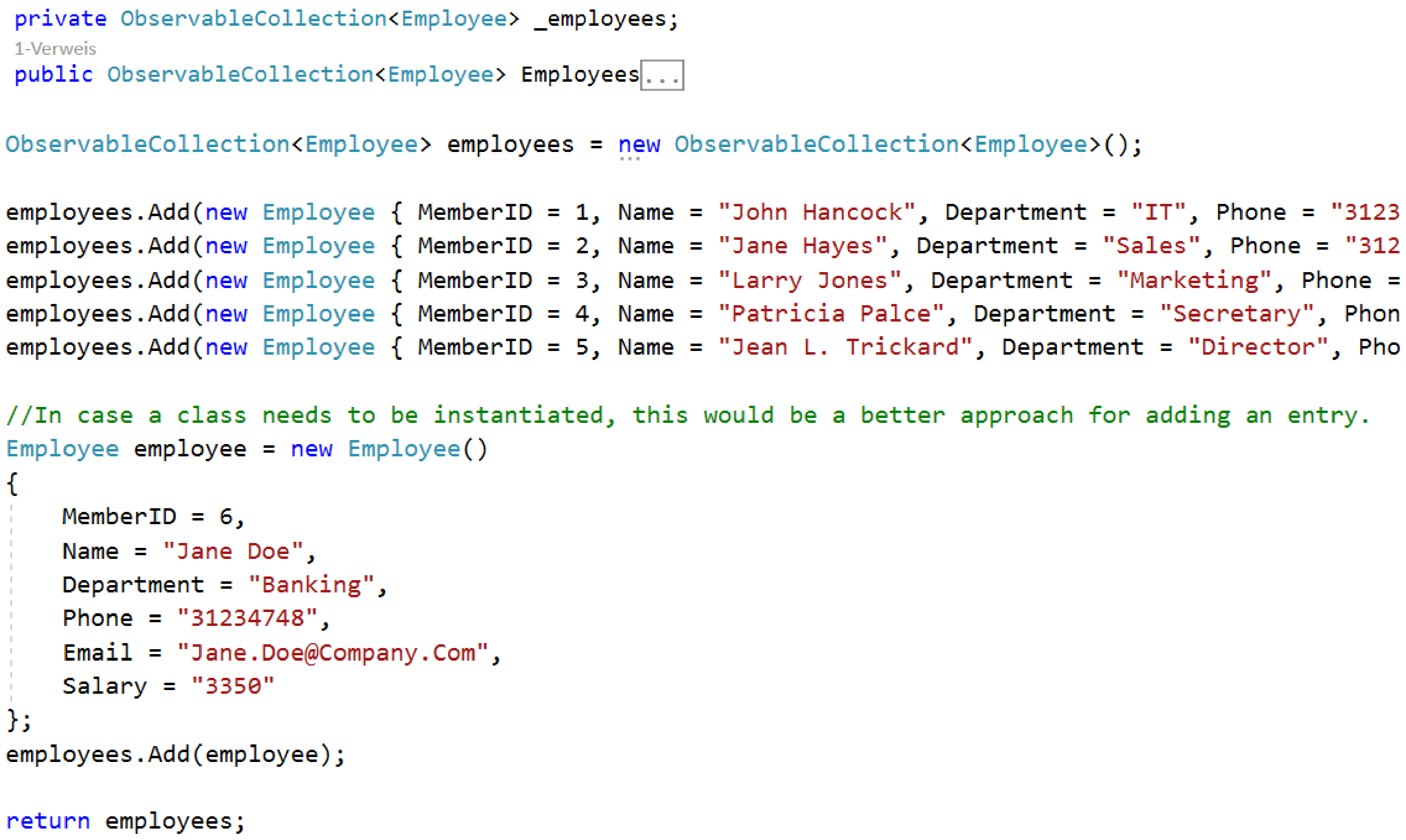
MemberID	Name	Department	Phone	Email	Salary
1	John Hancock	IT	31234743	John.Hancock@Company.com	3450.44
2	Jane Hayes	Sales	31234744	Jane.Hayes@Company.com	3700
3	Larry Jones	Marketing	31234745	Larry.Jones@Company.com	3000
4	Patricia Palce	Secretary	31234746	Patricia.Palce@Company.com	2900
5	Jean L. Trickard	Director	31234747	Jean.L.Tricard@Company.com	5400
6	Jane Doe	Banking	31234748	Jane.Doe@Company.Com	3350

Data Binding im XAML



```
<Window x:Class="Employee_Overview.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:VM="clr-namespace:Employee_Overview.ViewModels"
        Title="Company Employee List" Height="250" Width="625"
        Background="CornflowerBlue">

    <Window.DataContext>
        <VM:MainWindowVM/>
    </Window.DataContext>

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

        <TextBlock Text="Employees" FontSize="22" FontWeight="Bold" Foreground="DarkBlue"/>
        <DataGrid ItemsSource="{Binding Employees}" Grid.Row="1"/>
    </Grid>
</Window>
```

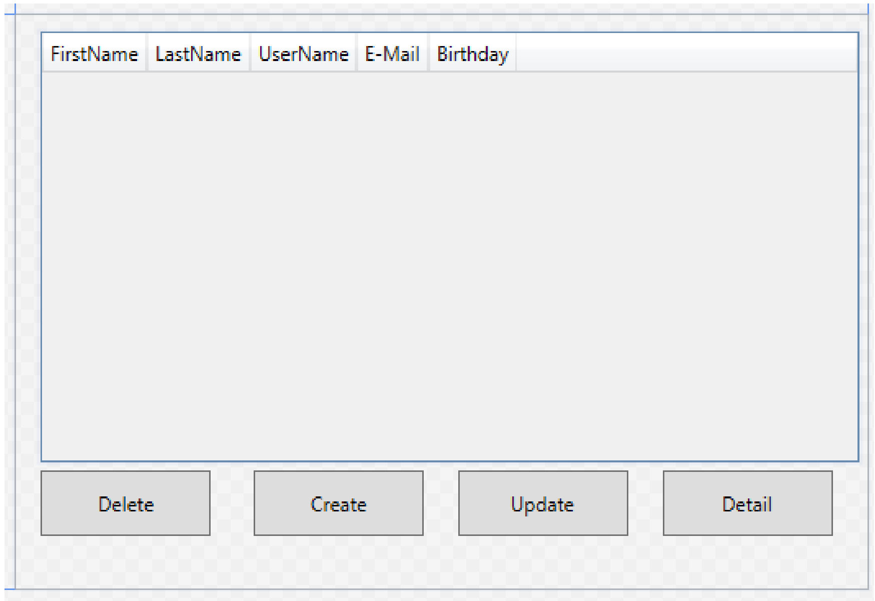
MainViewModel



```
private ObservableCollection<Employee> _employees;
1-Verweis
public ObservableCollection<Employee> Employees...

ObservableCollection<Employee> employees = new ObservableCollection<Employee>();

employees.Add(new Employee { MemberID = 1, Name = "John Hancock", Department = "IT", Phone = "3123
employees.Add(new Employee { MemberID = 2, Name = "Jane Hayes", Department = "Sales", Phone = "312
employees.Add(new Employee { MemberID = 3, Name = "Larry Jones", Department = "Marketing", Phone =
employees.Add(new Employee { MemberID = 4, Name = "Patricia Palce", Department = "Secretary", Phon
employees.Add(new Employee { MemberID = 5, Name = "Jean L. Trickard", Department = "Director", Pho

//In case a class needs to be instantiated, this would be a better approach for adding an entry.
Employee employee = new Employee()
{
    MemberID = 6,
    Name = "Jane Doe",
    Department = "Banking",
    Phone = "31234748",
    Email = "Jane.Doe@Company.Com",
    Salary = "3350"
};
employees.Add(employee);

return employees;
```

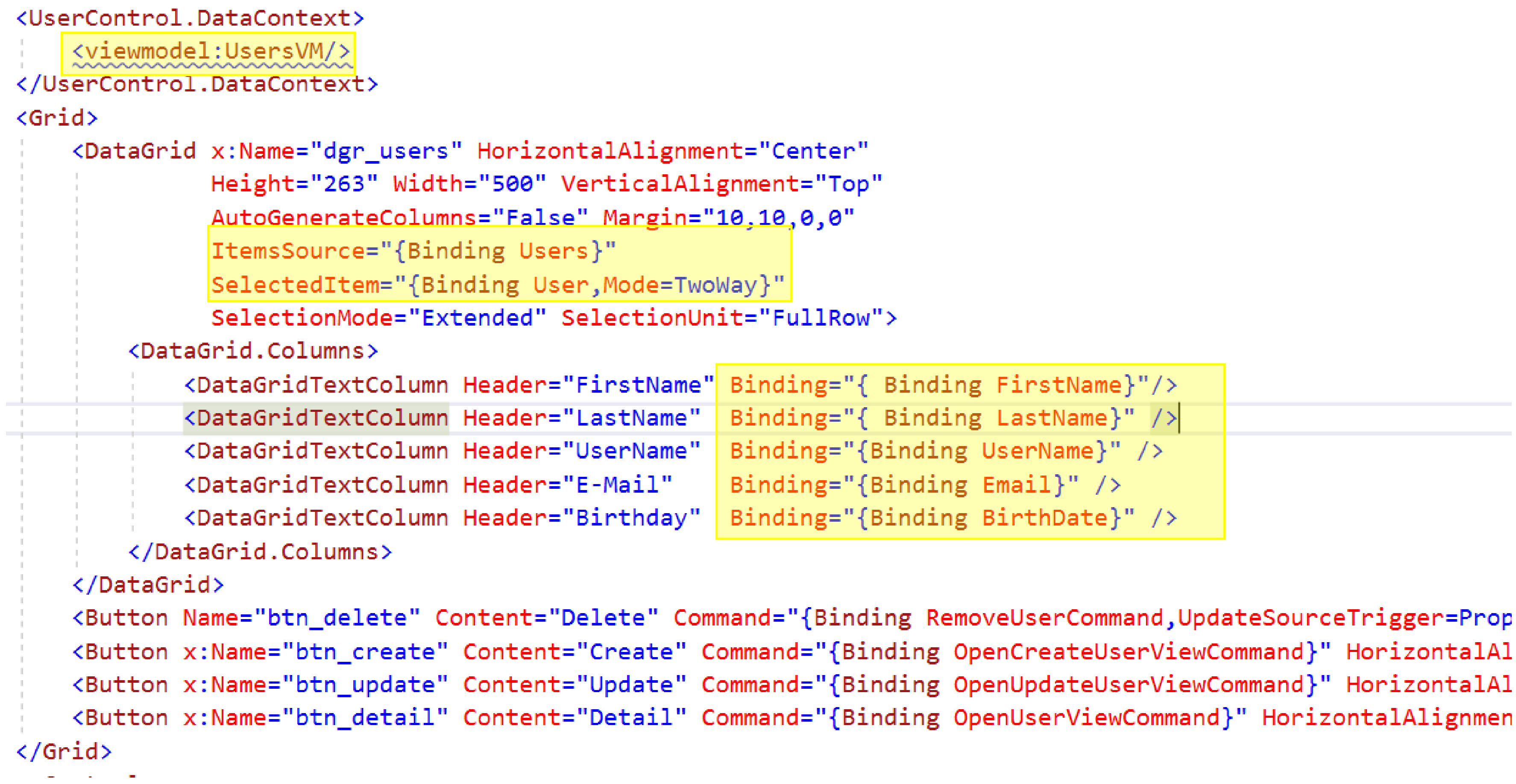
DataGrid User

- Users



```
public partial class User
{
        1-Verweis
        public int UserId { get; set; }
        15 Verweise
        public string FirstName { get; set; }
        15 Verweise
        public string LastName { get; set; }
        14 Verweise
        public string Email { get; set; }
        14 Verweise
        public string UserName { get; set; }
        14 Verweise
        public DateTime BirthDate { get; set; }
}
```

XAML DataGrid

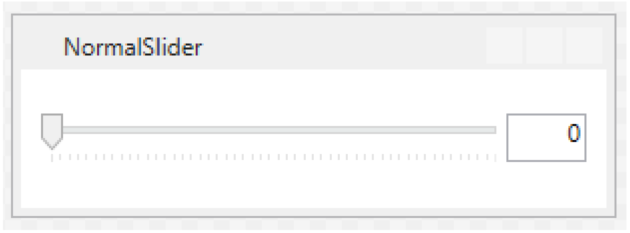


```
<UserControl.DataContext>
    <viewmodel:UsersVM/>
</UserControl.DataContext>
<Grid>
    <DataGrid x:Name="dgr_users" HorizontalAlignment="Center"
              Height="263" Width="500" VerticalAlignment="Top"
              AutoGenerateColumns="False" Margin="10,10,0,0"
              ItemsSource="{Binding Users}"
              SelectedItem="{Binding User,Mode=TwoWay}"
              SelectionMode="Extended" SelectionUnit="FullRow">
        <DataGrid.Columns>
            <DataGridTextColumn Header="FirstName" Binding="{ Binding FirstName}"/>
            <DataGridTextColumn Header="LastName"  Binding="{ Binding LastName}" />
            <DataGridTextColumn Header="UserName"  Binding="{Binding UserName}" />
            <DataGridTextColumn Header="E-Mail"    Binding="{Binding Email}" />
            <DataGridTextColumn Header="Birthday"  Binding="{Binding BirthDate}" />
        </DataGrid.Columns>
    </DataGrid>
    <Button Name="btn_delete" Content="Delete" Command="{Binding RemoveUserCommand,UpdateSourceTrigger=Prop
    <Button x:Name="btn_create" Content="Create" Command="{Binding OpenCreateUserViewCommand}" HorizontalAl
    <Button x:Name="btn_update" Content="Update" Command="{Binding OpenUpdateUserViewCommand}" HorizontalAl
    <Button x:Name="btn_detail" Content="Detail" Command="{Binding OpenUserViewCommand}" HorizontalAlignmen
</Grid>
```

Slider

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

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" />
```

ProgressBar

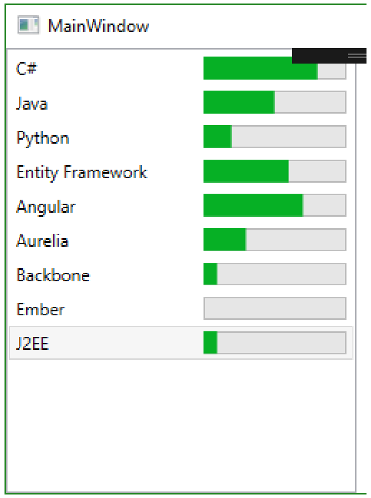
https://www.wpf-tutorial.com/misc-controls/the-progressbar-control/

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

ProgressBar

- Minimum, Maximum & Value with Binding

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

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



C#
Java
Python
Entity Framework
Angular
Aurelia
Backbone
Ember
J2EE

Binding Modes

OneWay, OneWayToSource, TwoWay, OneTime

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

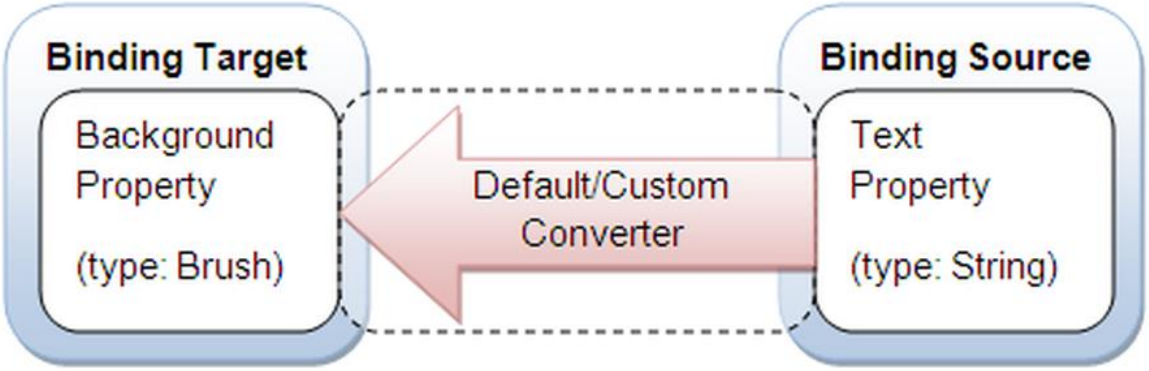
- transfers values in both directions

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



Value Conversion

with IValueConverter

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

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

When to use a value converter…

- frequently used with data bindings
  - numeric value showing zero values for the negative numbers

  - CheckBox based on a string like "yes" or "no" instead of a Boolean value

  - Binding an enum value to a control convert it to an integer or boolean

Value Converter

- implements the IValueConverter interface
- Interface IValueConverter provides two object level conversion methods:
    - Convert
        - changing values from ViewModel to View
    - ConvertBack
        - changing values back from View to ViewModel

Convert Method

```
class IntToStringConverter : IValueConverter
{
    public object Convert(object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        int i = System.Convert.ToInt32(value);
        switch(i)
        {
            case 1:
                return "ONE";
            case 2:
                return "TWO";
            case 3:
                return "THREE";
            default:
                return "A LOT";
        }
    }
```

Convert Back Method

```
public object ConvertBack(object value, Type targetType,
    object parameter, CultureInfo culture)
{
     string s = (string)value;
     switch (s)
     {
          case "ONE":
               return 1;
          case "TWO":
               return 2;
          case "THREE":
               return 3;
          default:
               return Int32.MaxValue;
     }
}
```

Using the Converter as

- Using Resources…
- binding the TextBox to the Slider

```
<Window.Resources>
    <local:IntToStringConverter x:Key="MyConverter" />
</Window.Resources>
[...]
<TextBox Text="{Binding ElementName=sldValue, Path=Value,
   Converter={StaticResource MyConverter}}" />
<Slider Name="sldValue"  Minimum="1" TickFrequency="1"
   Maximum="3" IsSnapToTickEnabled="True" />
```

- What‘s a Resource?!

Use the Converter as Static Resource

https://www.wpf-tutorial.com/wpf-application/resources/

Simple Static Resources

- Declare a Static Resource

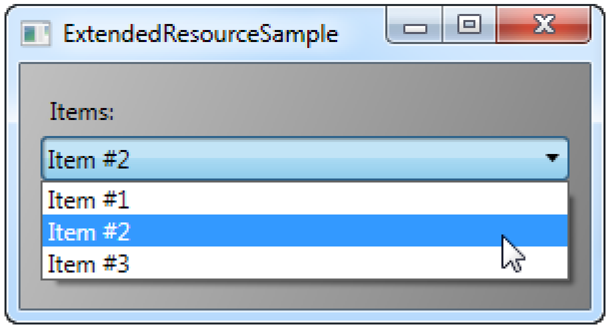
```
<StackPanel>
    <StackPanel.Resources>
        <local:Person x:Key="MyPerson" />
    </StackPanel.Resources>

    <TextBox Text="{Binding Source={StaticResource
       ResourceKey=MyPerson}, Path=Name}" />
</StackPanel>
```

- Bind on a Static Resource

```
"{Binding Source={StaticResource MyPerson}, Path=Name}"
```

Resource & ComboBox Example

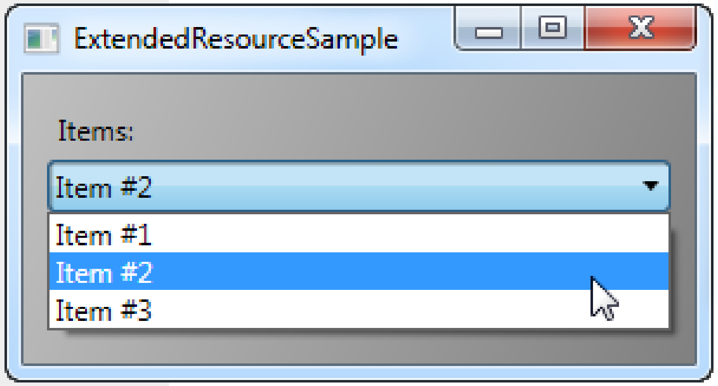
- Implement a Resource
  - for the Title „Items:“
  - for the ComboBoxItems:
    - „Item #1“
    - „Item #2“
    - „Item #3“



https://www.wpf-tutorial.com/wpf-application/resources/

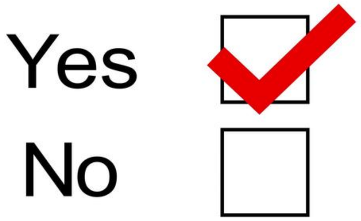
Static Resource

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

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

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



Yes No Converter

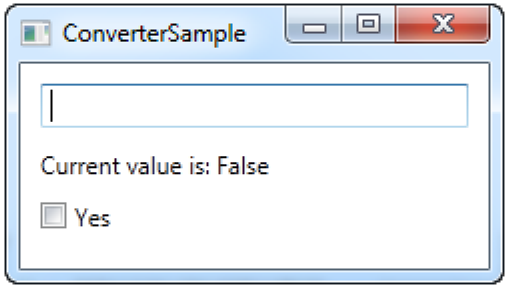
Example Yes No Boolean Converter



- 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

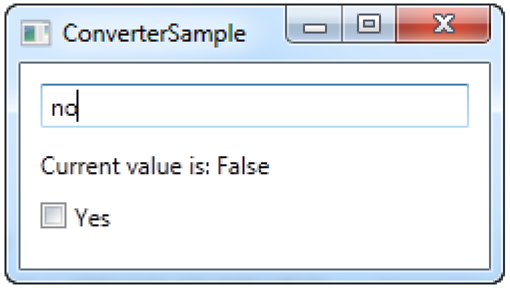
YesNoBooleanConverter Class

```
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";
    }
}
```


ConverterSample

Current value is: False

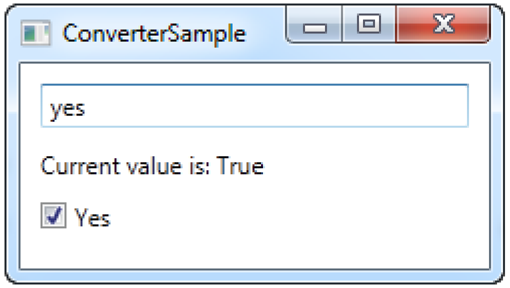
☐ Yes


ConverterSample

no

Current value is: False

☐ Yes


ConverterSample

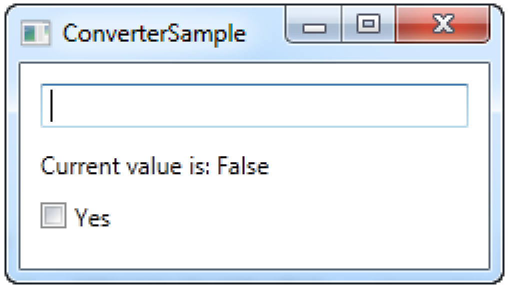
yes

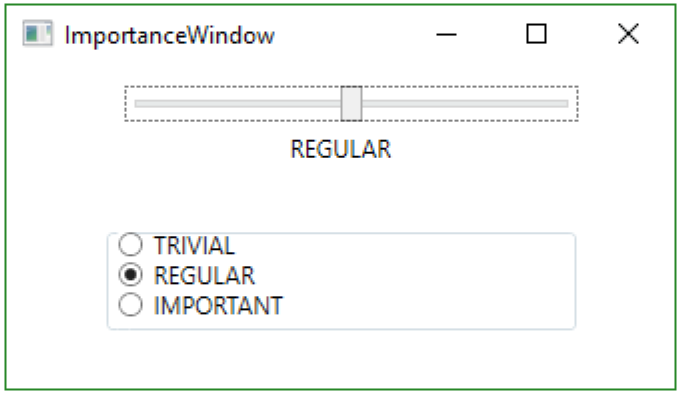
Current value is: True

☑ Yes

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);
      }
}
```

Additional Information Blackwasp

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