

MVVM

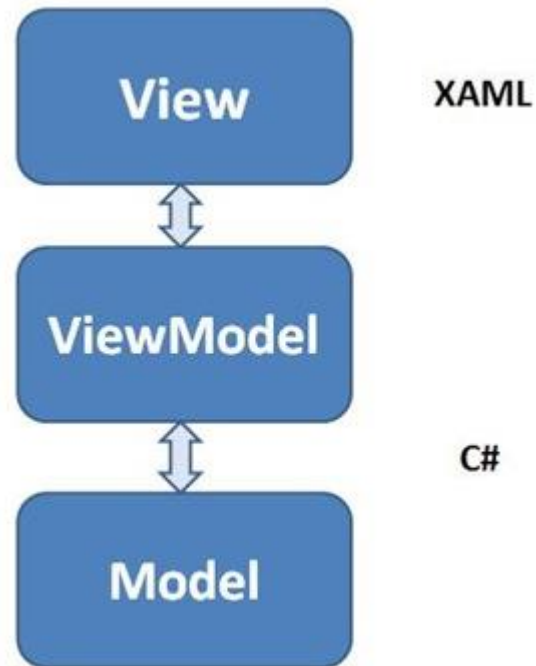
beyond the basics



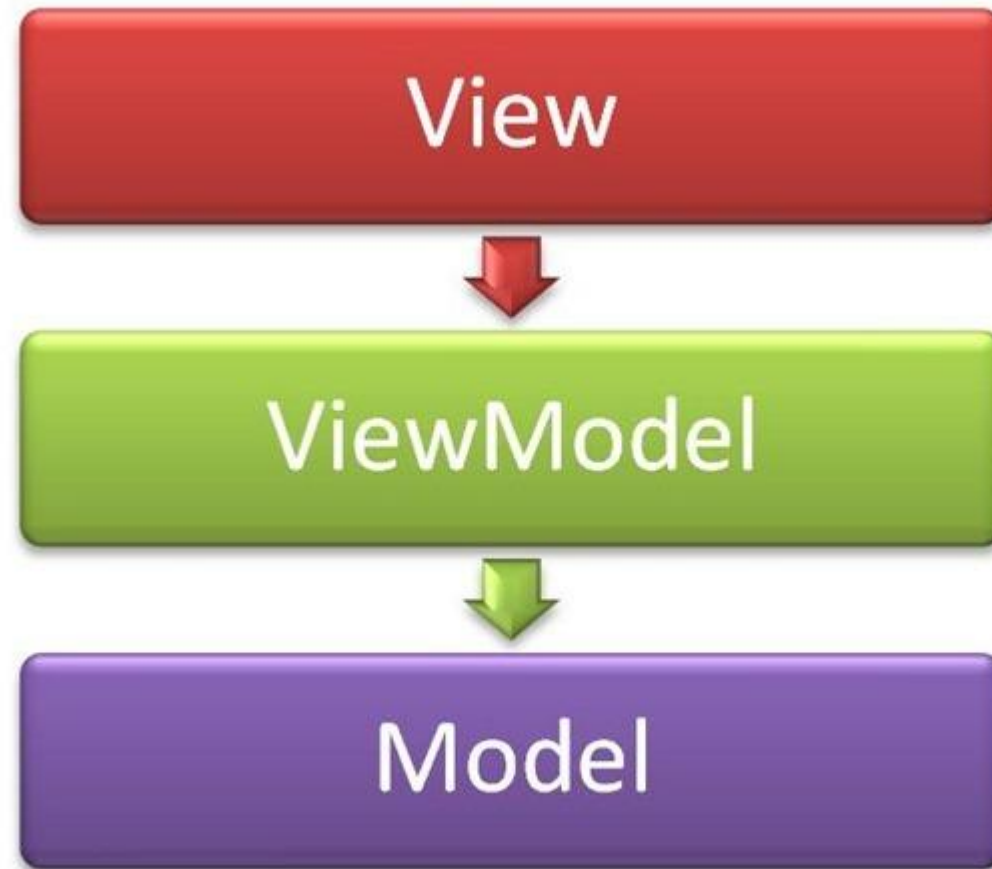
DI Thomas Mutzl
Software Architect



MVVM – simple explained



MVVM – simple explained



MVVM – simple explained



MVVM – simple explained

```
public class Contact
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public DateTime? Birthdate { get; set; }
    public string City { get; set; }
}
```

```
public class ContactViewModel : INotifyPropertyChanged
{
    private List<string> _cities;
    private Contact _contact;

    public ContactViewModel()
    {
        Cities = new List<string> { "Wien", "Neulengbach", "Linz", "Graz", "Salzburg", "Innsbruck", };
        Contact = new Contact { FirstName = "Thomas", LastName = "Mutzl", City = "Neulengbach" };
    }

    public Contact Contact
    {
        get { return _contact; }
        set
        {
            _contact = value;
            RaisePropertyChanged("Contact");
        }
    }

    public List<string> Cities
    {
        get { return _cities; }
        private set
        {
            _cities = value;
            RaisePropertyChanged("Cities");
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;

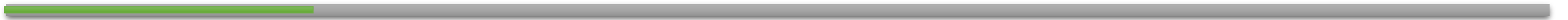
    protected virtual void RaisePropertyChanged(string propertyName)
    {
        var handler = PropertyChanged;
        if (handler != null) handler(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

The screenshot shows a WPF window titled "Kontakt". It contains four controls: a text box for "Vorname" with the value "Thomas", a text box for "Nachname" with the value "Mutzl", a date picker for "Geburtsdatum" showing "15", and a ComboBox for "Stadt" with "Neulengbach" selected. The ComboBox dropdown is open, showing a list of cities: "Neulengbach", "Wien", "Linz", "Graz", "Salzburg", and "Innsbruck".

```
<Grid DataContext="{StaticResource ContactViewMo"
<TextBox Text="{Binding Contact.FirstName, Mo"
<TextBox Text="{Binding Contact.LastName, Mo"
<DatePicker SelectedDate="{Binding Contact.Bir"
<ComboBox ItemsSource="{Binding Cities}"
            SelectedItem="{Binding Contact.City, Mode=TwoWay}" />
</Grid>
```

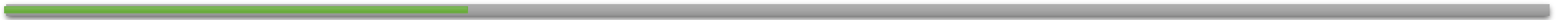
MVVM – basics

- Model – View – ViewModel
- Implementierung von INPC (INotifyPropertyChanged)
- Frameworks
 - MVVM Light
 - Caliburn Micro
 - Prism
 - ...



Properties mit NotifyPropertyChangedInvoker

- Code Snippets
- CallerMemberNameAttribute
- ReSharper
- Aspekt Orientierte Programmierung



Fody

- NuGet Package PropertyChanged.Fody
- <https://github.com/Fody/PropertyChanged>
- Injecting code into IL

```
1 reference
public string FirstName { get; set; }
1 reference
public string LastName { get; set; }

0 references
public string FullName
{
    get
    {
        return string.Format("{0}, {1}", LastName, FirstName);
    }
}
```



```
public string FirstName
{
    [CompilerGenerated]
    get
    {
        return this.<FirstName>k__BackingField;
    }
    [CompilerGenerated]
    set
    {
        if (!string.Equals(this.<FirstName>k__BackingField, value, StringComparison.Ordinal))
        {
            this.<FirstName>k__BackingField = value;
            this.RaisePropertyChanged("FullName");
            this.RaisePropertyChanged("FirstName");
        }
    }
}
```


ViewModelLocator

- Klasse mit Properties für jedes ViewModel
- Ermöglicht Verwendung von Dependency Injection

```
1 reference
public class ViewModelLocator
{
    0 references
    public ViewModelLocator()
    {
        ServiceLocator.SetLocatorProvider(() => SimpleIoc.Default);

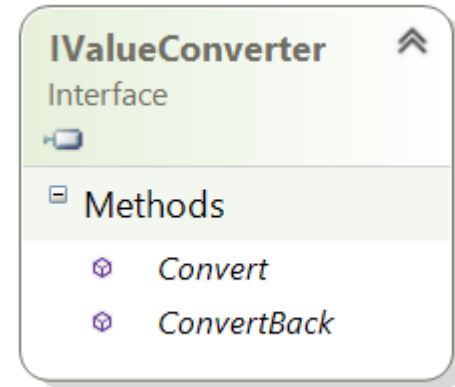
        SimpleIoc.Default.Register<MainViewModel>();
    }
    0 references
    public MainViewModel MainViewModel { get { return ServiceLocator.Current.GetInstance<MainViewModel>(); } }
}
```

- Statische Ressource `<viewModels:ViewModelLocator x:Key="Locator" />`
- Deklaratives Mapping von View zu ViewModel

```
DataContext="{Binding MainViewModel, Source={StaticResource Locator}}"
```

ValueConverter

- Konvertiert von einem Datentyp in einen beliebigen anderen Datentyp (und zurück)
- Converter implementiert IValueConverter
- Ermöglicht Trennung von UI-Aspekten (z.B. Visibility, Color) von Business Logik



Observable Collection

- Hinzufügen oder Löschen von Elementen einer Liste
 - Statt `List<T>`
Verwendung von `ObservableCollection<T>`
 - Implementiert `INotifyCollectionChanged`
-

Commands

- ButtonBase nicht nur Click Event sondern auch Command Property

```
namespace System.Windows.Controls.Primitives
```

```
{
```

```
    ...public abstract class ButtonBase : ContentControl
```

```
    {
```

```
        ...public ICommand Command { get; set; }
```

```
        ...public event RoutedEventHandler Click;
```

```
    }
```

```
}
```

```
<Button Click="ButtonBase_OnClick"
```

```
Command="{Binding DemoCommand}"/>
```

- ICommand
- Keine Implementierung in .NET BCL
- RelayCommand, DelegateCommand, ...

Command ohne Button

- Codebehind

```
((MainViewModel)DataContext).DemoCommand.Execute(null);
```

- EventToCommandBehavior



DependentRelayCommand

- Erweiterung zum `RelayCommand`
- Automatisches `RaiseCanExecuteChanged()` wenn sich Properties ändern, welche Einfluss auf den Return-Value von `CanExecute()` haben

```
AddColorCommand  
= new DependentRelayCommand(AddColor, CanAddColor, this, "NewColor");
```

```
SaveCommand  
= new DependentRelayCommand(Save, CanSave, this, () => FirstName, () => LastName);
```

- Als NuGet Package verfügbar.



MVVM Light DependentRelayCommand (PCL)

Extends the `RelayCommand` of the MVVM Light Toolkit. Fires the `CanExecuteChanged` event for the command automatically if a `PropertyChanged` event of a dependent pro...



www.mutzl.com



thomas@mutzl.com



[/thomas.mutzl](https://www.facebook.com/thomas.mutzl/)



[@mutzl](https://twitter.com/mutzl)

Danke!



DI Thomas Mutzl
Software Architect