

MVVM

Introduction



- Jan Schweda
 - Development Trainer
@ppedv AG
 - C#, WCF, WPF, Azure
-  jan@familieschweda.de
-  [http://blog.ppedv.de/author/
JanS.aspx](http://blog.ppedv.de/author/JanS.aspx)

Agenda

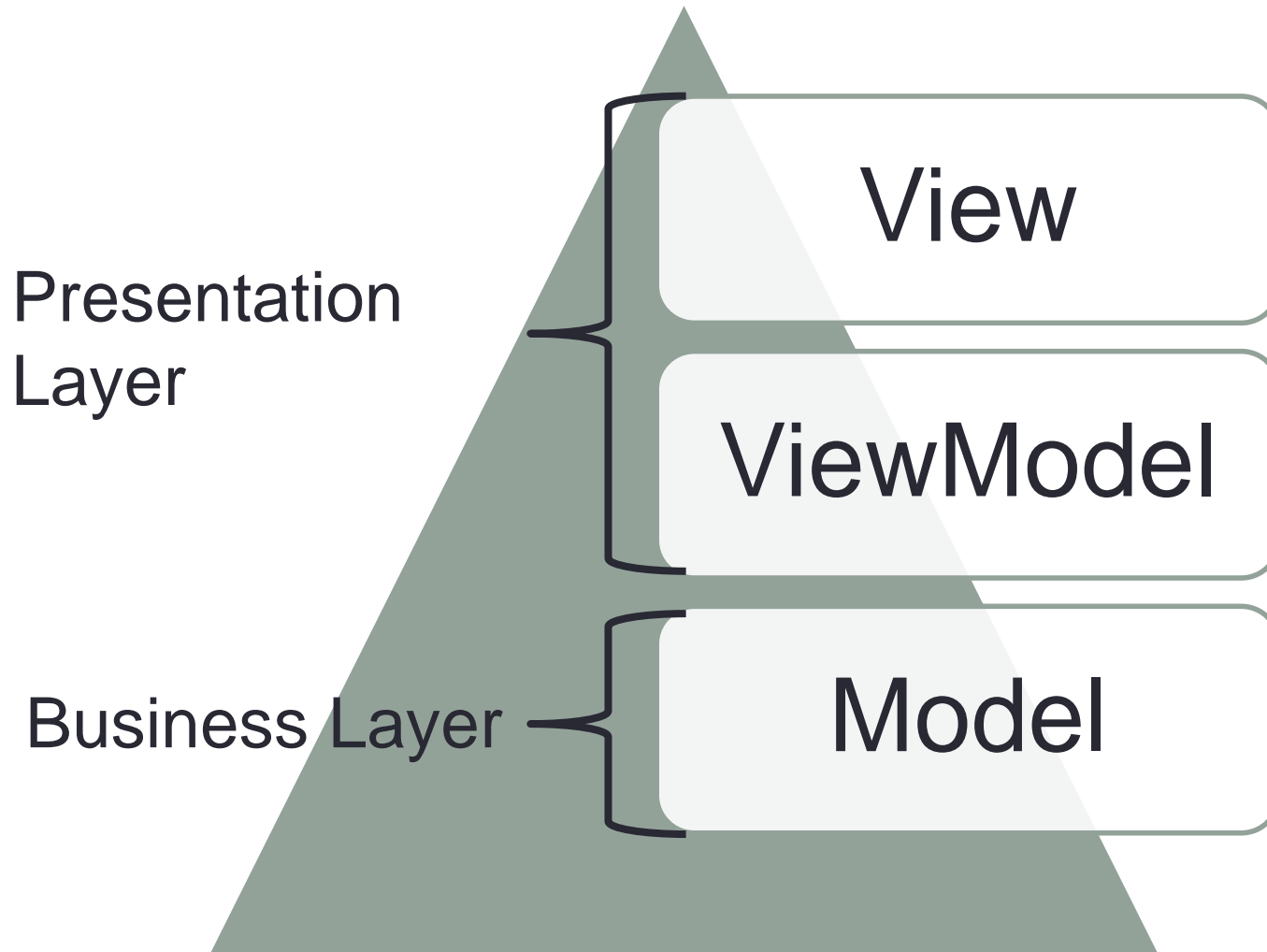
1. Overview
2. Preconditions
3. Model
4. ViewModel
5. View
6. Let's morph our application

OVERVIEW

MVVM Generals

- Presented in 2005
- Supported UI technologies
 - WPF
 - Silverlight
 - Windows 8
- More than 20 frameworks listed on wikipedia

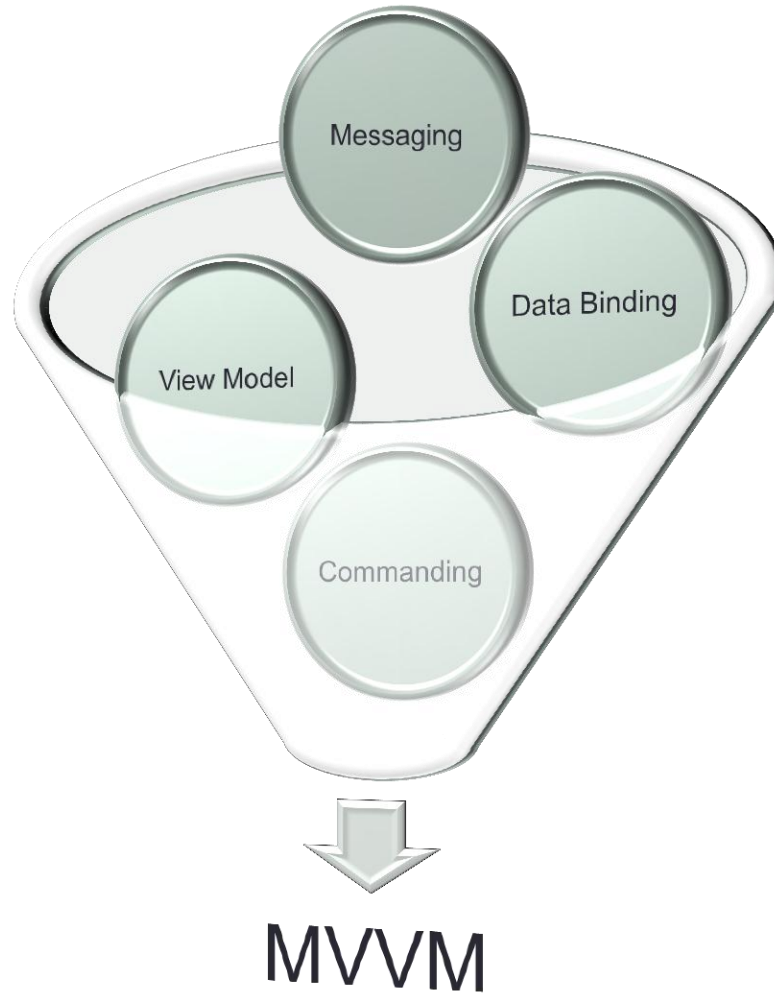
MVVM Structure



MVVM Benefits

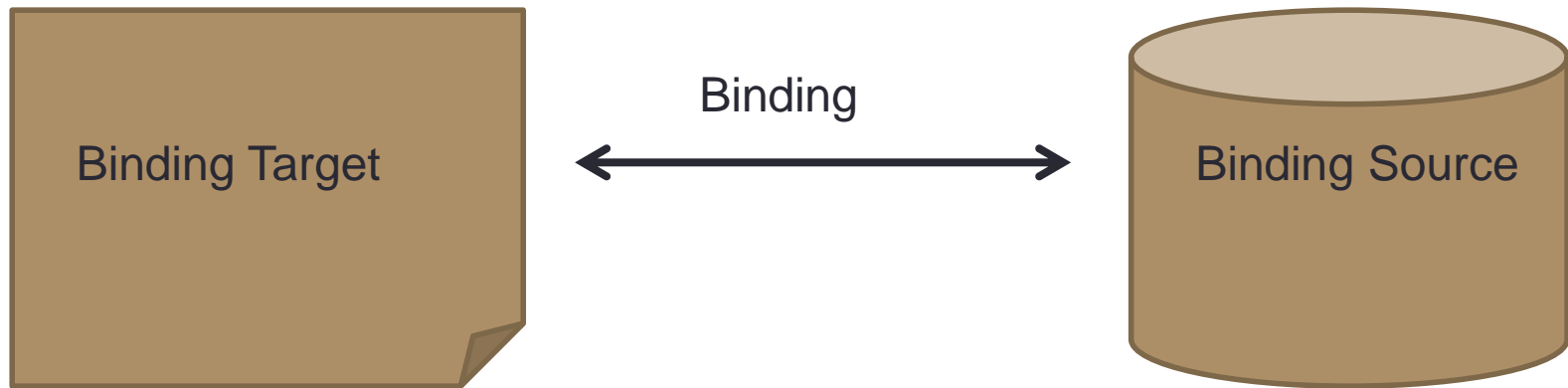
- Separation of Concerns
- Better Testability
- More reusable Code

MVVM Elements



HOW DOES IT WORK

DataBinding



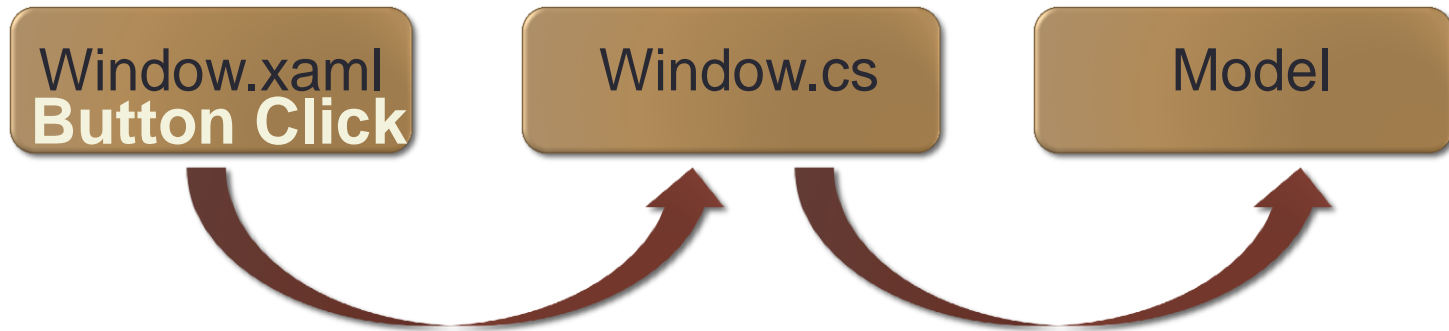
INotifyPropertyChanged

```
public interface INotifyPropertyChanged
{
    event PropertyChangedEventHandler PropertyChanged;
}

void XXX_PropertyChanged(object sender, PropertyChangedEventArgs e)
{
    // e.PropertyName;
}
```

Commanding

- Event Handling



- Commanding



ButtonBase

```
public abstract class ButtonBase : ContentControl
{
    public ICommand Command { get; set; }
    public object CommandParameter { get; set; }
}
```

ICommand

```
public interface ICommand
{
    event EventHandler CanExecuteChanged;
    bool CanExecute(object parameter);
    void Execute(object parameter);
}
```

Using Commanding

```
<Window.CommandBindings>  
<CommandBinding Command="Close,,  
    Executed="CommandBinding_Executed,,  
    CanExecute="CommandBinding_CanExecute"/>  
</Window.CommandBindings>  
<Button Command="Close">Klick mich</Button>
```

```
private void CB_Executed(object sender, ExecutedRoutedEventArgs e)  
{  
    // execute here  
}
```

```
private void CB_CanExecute(object sender, CanExecuteRoutedEventArgs e)  
{  
    e.CanExecute = true;  
}
```

InvokeCommandAction Behavior

- Command functionality for non ButtonBase based controls
- Located in System.Windows.Interactivity.dll

<Window

xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity">

<i:Interaction.Triggers>

<i:EventTrigger eventName="Loaded">

<i:InvokeCommandAction Command="Close"/>

</i:EventTrigger>

</i:Interaction.Triggers>

</Window>

MODEL

Model

- Don't change if you don't need to

VIEWMODEL

ViewModel Class

- Implement INotifyPropertyChanged
- Create Property for Model
- Create Additional Properties

VIEW

What to avoid in the View

- Don't use Name or x:Name
- Don't use Eventhandler
- Decide between IValueConverter and additional Properties

Connecting ViewModel and View

- Bind ViewModel to a View
 - Declaratively in XAML
 - In Code behind
 - Using advanced approaches

Binding the ViewModel declaratively

```
<Window x:Class="MvvmSample.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:vm="clr-namespace:MvvmSample.ViewModels">
<Window.Resources>
    <vm:MainViewModel x:Key="vm"/>
</Window.Resources>
    <Grid DataContext="{Binding Source={StaticResource vm}}">
        ...
    </Grid>
</Window>
```



Binding the ViewModel in Code

```
void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    MainPageViewModel vm = new MainPageViewModel();
    this.DataContext = vm;
}
```

Questions



Let's transform an application

