

.NET Training Slides



Rainer Stropek
software architects gmbh

MVVM
mit WPF und C#

Web: <http://www.timecockpit.com>
Mail: rainer@timecockpit.com
Twitter: @rstropek

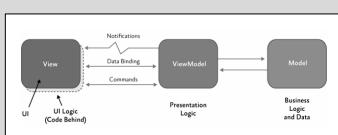


time cockpit
Saves the day.

Inhalt

Wenn Programme mit WPF über den Status "Hello World" hinaus wachsen, braucht man ein Konzept für gute Gliederung des Codes, Testbarkeit und Codewiederverwendung. Mit dem MVVM (Model-View-ViewModel) Pattern hat sich im Lauf der Jahre ein Architekturmuster entwickelt, das eine Lösung für diese Themen verspricht. Die WPF-Bibliothek von Microsoft unterstützt MVVM optimal.

- ▶ Vorstellung von MVVM anhand eines **durchgängigen Beispiels**
- ▶ UI-Design (View), Geschäftsobjekte und -Logik (Model) und UI-Logik (ViewModel) richtig trennen und dadurch ein besser strukturiertes und leichter **testbares Programm** erhalten
- ▶ WPF **Data Binding** einsetzen, um View an ViewModel und Model zu binden
- ▶ typische Fällen bei der Umsetzung von MVVM und **Tipps**, wie man sie umgeht



MVVM
Model-View-ViewModel

Architecture

UI Development Framework Prism Helpful when applying MVVM

View

- ▶ Uses declarative language XAML
XML Application Markup Language
For details see separate slide deck
- ▶ Code-behind is (nearly) empty
Call to `InitializeComponent`
Set `DataContext` to `ViewModel`
UI-Logic that cannot be expressed in XAML efficiently
- ▶ Prism-Views are derived from `Control/UserControl`
Option: Use Data Templates

ViewModel

- ▶ No direct reference to the view
- ▶ Offers properties for data binding
IPropertyChanged vs. Dependency Properties
Use `ICommand` for actions that the user can invoke (e.g. using a button)
- ▶ Might contain validation logic
`IDataErrorInfo`, `INotifyDataErrorInfo`
- ▶ Re-use the VM across different platforms
Tip: Consider using a portable class library

Model

- ▶ Data classes and business logic
Client-side domain model
Code to support data access
- ▶ Tip: Make your model data-binding-friendly
`IPropertyChanged`, `IDataErrorInfo`, `ObservableCollection<T>`, etc.
No Dependency Properties
- ▶ Tip: If possible generate your client-side model or re-use existing code

Data Binding

Connecting View and ViewModel

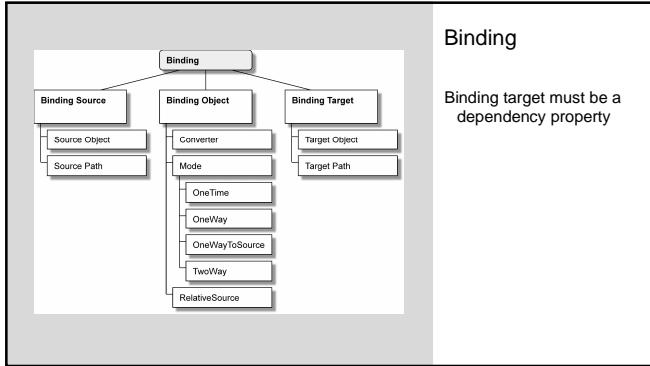
Binding Basics

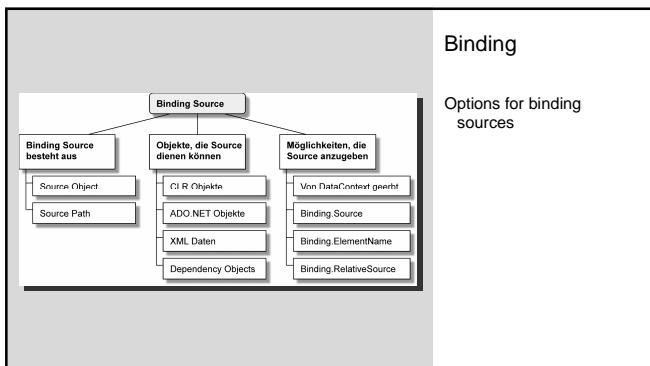
- ▶ *{Binding ...}* Markup Extension
- ▶ *Binding* Class
- ▶ Not a concept of XAML, concept of WPF

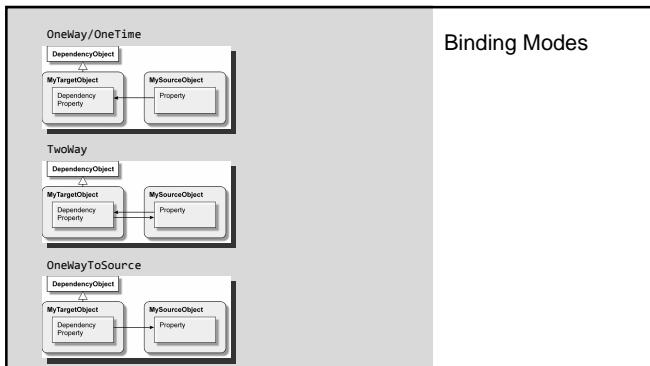
```
<Window.Resources>
    <local:Competitor x:Key="MyCompetitor"
        FirstName="Benjamin"
        LastName="Raich" />
</Window.Resources>
<StackPanel>
    <TextBox Text="{Binding Source={StaticResource MyCompetitor}, Path=FirstName}" />
    <TextBox Text="{Binding Source={StaticResource MyCompetitor}, Path=LastName}" />
    <TextBlock Text="Competitor:</TextBlock>
    <TextBlock Text="{Binding ElementName=FirstNameTextBox, Path=Text}" />
    <TextBlock Text="{Binding ElementName=LastNameTextBox, Path=Text}" />
</StackPanel>

<StackPanel Orientation="Horizontal">
    <Button Click="Button_Click">Show Competitor</Button>
    <Button Click="Button_Click_1">Change Competitor</Button>
</StackPanel>
```

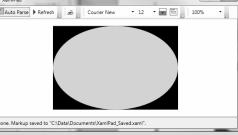
Binding Basics







```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
<Canvas>
    <Canvas> Height="100" Background="Black">
        <Ellipse Canvas.Top="0" Canvas.Left="0" Fill="LightGray" Height="Binding Path=Height"/>
        <RelativeSource Mode="FindAncestor AncestorType=Canvas></RelativeSource>
        <Ellipse>
            <Binding Path=Width>
                <RelativeSource Mode="FindAncestor" AncestorType="Canvas"/>
            </Binding>
        </Ellipse>
    </Canvas>
</Page>
```



Binding
Advanced Scenarios

Relative Source Binding

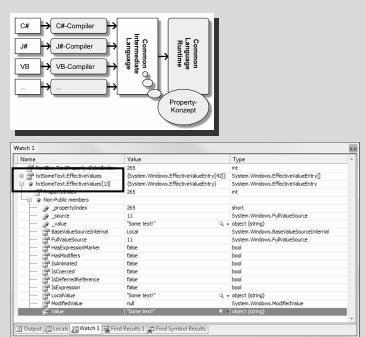
```
[Label Grid.Column="1" Grid.Row="2"
Content="{Binding Source={StaticResource spaceInfo},
Path=FreeSpaceRatio,
Converter={StaticResource ratioToStringConverter},
ConverterParameter="#.##%"}]

[ValueConversion(typeof(Double), typeof(String))]
public class RatioToStringConverter : ValueConverter
{
    public object Convert(object value, Type targetType,
        object parameter, System.Globalization.CultureInfo culture)
    {
        return ((Double)value).ToString(parameter as String,
            culture.NumberFormat);
    }

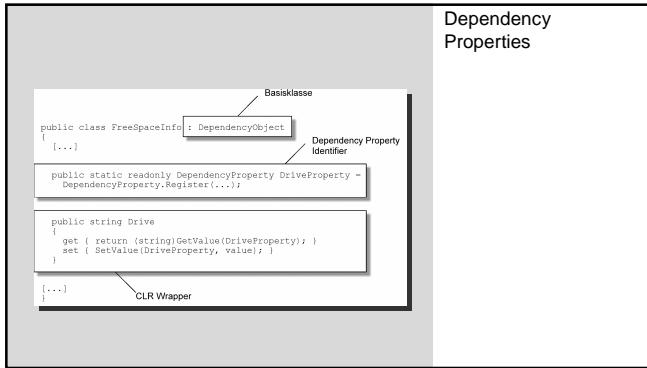
    public object ConvertBack(object value, Type targetType,
        object parameter, System.Globalization.CultureInfo culture)
    {
        return System.Convert.ToDouble(value as string,
            culture.NumberFormat);
    }
}
```

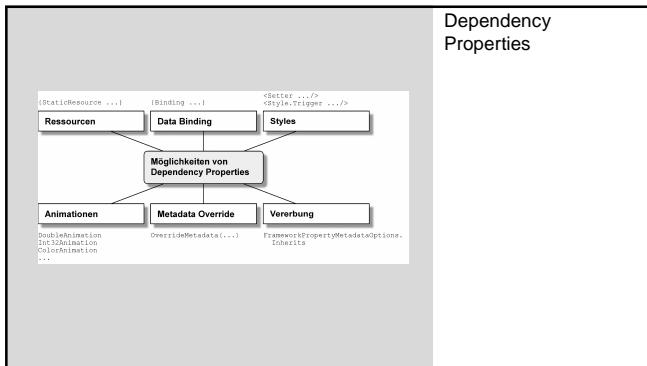
Binding
Advanced Scenarios

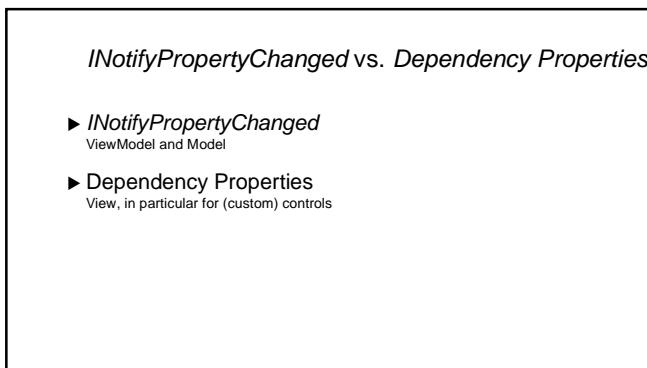
Converter

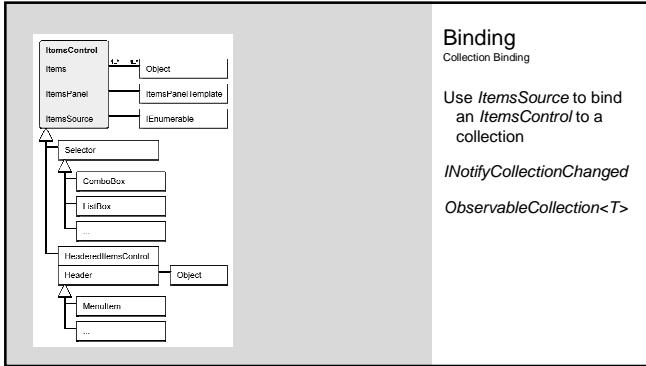


Dependency Properties



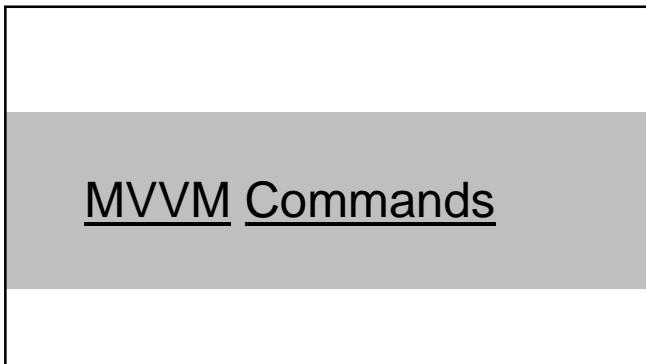




**Binding**

Collection Binding

Use *ItemsSource* to bind an *ItemsControl* to a collection*INotifyCollectionChanged**ObservableCollection<T>*



```

public class DelegateCommand<T> : DelegateCommandBase
{
    public DelegateCommand(
        Action<T> executeMethod,
        Func<T, bool> canExecuteMethod )
    {
        base((o) => executeMethod((T)o), (o) => canExecuteMethod((T)o))
        ...
    }
}

```

Command Objects*ICommand*Implement *ICommand* yourselfUse Prism's *DelegateCommand<T>*

Interaction Triggers

```
<UserControl [...]>
<x:Namespace Uri="http://schemas.microsoft.com/expression/2010/interactivity" />
<x:Interaction.Triggers>
    <i:InteractionTrigger EventName="MouseLeftButtonDown">
        <i:InvokeCommandAction Command="Binding Path=MyCommand"/>
    </i:InteractionTrigger>
</x:Interaction.Triggers>
</UserControl>
```

Use commands with controls that does not support commands out of the box

Defined in Expression Blend
Tip: Reference assemblies come with Prism

Use *CallMethodAction* to call methods without *ICommand*
No support for parameters

Composite Commands



ShellViewModel *CompositeCommand "SaveAll"*

```
graph TD
    SaveAll[ShellViewModel CompositeCommand "SaveAll"] --> Button[Button]
    SaveAll --> ViewModelA[ViewModel A DelegateCommand "Save A"]
    SaveAll --> ViewModelB[ViewModel B DelegateCommand "Save B"]
    SaveAll --> ViewModelC[ViewModel C DelegateCommand "Save C"]
```

Implemented in *CompositeCommand*
RegisterCommand
UnregisterCommand

Composite Commands



ShellViewModel *CompositeCommand "Zoom"*

```
graph TD
    Zoom[ShellViewModel CompositeCommand "Zoom"] -- Active --> Button[Button]
    Zoom --> ViewModelA[ViewModel A DelegateCommand "Zoom A"]
    Zoom --> ViewModelB[ViewModel B DelegateCommand "Zoom B"]
    Zoom --> ViewModelC[ViewModel C DelegateCommand "Zoom C"]
```

Implement *IActiveAware* on View or ViewModel class

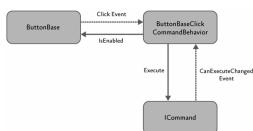
Implement *IActiveAware* on your commands
DelegateCommand and *CompositeCommand* already implement *IActiveAware*

When View/ViewModel becomes inactive, deactivate child commands

Specify true in *monitorCommandActivity* of constructor of *CompositeCommand*

Prism Command Behaviors

- Rarely needed in WPF because *ButtonBase* supports *ICommand* out of the box
- Connect events of a control with a command object (*ICommand*)
Behavior object monitors the control's events
Connection between control and behavior object is created with attached properties



```

<Button Content="Submit All"
prism:Click.Command="={Binding Path=SubmitAllCommand}"
prism:Click.CommandParameter="={Binding Path=TicketSymbol}" />

public static class Click
{
    public static readonly DependencyProperty CommandProperty =
        DependencyProperty.RegisterAttached("Command", typeof(ICommand),
                                         GetCommand,
                                         GetCommand);
    public static ICommand GetCommand(ButtonBase buttonBase)
    {
        return (ICommand)buttonBase.GetValue(CommandProperty);
    }

    private static void SetCommand(ButtonBase buttonBase, ICommand value)
    {
        buttonBase.SetValue(CommandProperty, value);
    }
}

public class ButtonBaseClickCommandBehavior : CommandBehaviorBase<ButtonBase>
{
    public ButtonBaseClickCommandBehavior(ButtonBase clickableObject)
        : base(clickableObject)
    {
        clickableObject.Click += OnClick;
    }

    private void OnClick(object sender, System.Windows.RoutedEventArgs e)
    {
        ExecuteCommand();
    }
}
  
```

Click Behavior

Code Walkthrough

Note that this is not necessary in WPF
You could create custom behaviors based on the same idea if necessary

Interaction Patterns

Triggering UI interaction from the ViewModel

```
var result =
    interactionService.ShowMessageBox(
        "Are you sure you want to cancel this operation?",
        "Confirm",
        MessageBoxButton.OK );
if (result == MessageBoxResult.Yes)
{
    CancelRequest();
}
```

The diagram illustrates the Interaction Services pattern. A Shell View Model contains a DelegateCommand named 'DelegateCommand<Submit>'. This command is triggered by a button in the View. The execution flow goes through the DelegateCommand to the Shell View Model, then to the Interaction Service, and finally back to the View.

Interaction Services
WPF, Silverlight not covered here

ViewModel communicates with View using an [interaction service](#)

The diagram illustrates the Interaction Request Objects pattern. A Shell View Model contains a DelegateCommand named 'DelegateCommand<Submit>'. This command is triggered by a button in the View. The execution flow goes through the DelegateCommand to the Shell View Model, then to an 'Interaction Request' object, and finally back to the View.

Interaction Request Objects

Uses `InteractionRequest<T>` class
Implements `IInteractionRequest`

ViewModel initiates interaction uses the `Raise` method

View uses behaviors to implement UI

```
public class MainWindowViewModel
{
    private InteractionRequest<Confirmation> confirmCancelInteractionRequest;
    public MainWindowViewModel()
    {
        this.confirmCancelInteractionRequest = new InteractionRequest<Confirmation>();
    }
    public IInteractionRequest<Confirmation> ConfirmCancelInteractionRequest
    {
        get
        {
            return this.confirmCancelInteractionRequest;
        }
    }
    public void Cancel()
    {
        this.confirmCancelInteractionRequest.Raise(
            new Confirmation() { Content = "Are you sure you wish to cancel?" },
            confirmation =>
        {
            if (confirmation.Confirmed)
            {
                // Do whatever has to be done
            }
        });
    }
}
```

InteractionRequest ViewModel

InteractionRequest
ViewModel

```
<window x:Class="InteractionRequestSample.Window1" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:sdk="http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk"
    xmlns:i="http://schemas.microsoft.com/expression/2010/internals"
    xmlns:prism="http://www.codeplex.com/prism"
    xmlns:sample="clr-namespace:InteractionRequestSample"
    Title="MainWindow" Height="350" Width="725">
    <i:Interaction.Triggers>
        <prism:InteractionRequestTrigger>
            <eventTrigger EventName="ConfirmCancelInteractionRequest">
                <vw:MessageBoxAction />
            </eventTrigger>
        </prism:InteractionRequestTrigger>
    </i:Interaction.Triggers>
    <Button Content="Click Me!">
        <i:Interaction.Triggers>
            <eventTrigger EventName="Click">
                <ei:CallMethodAction TargetObject="{Binding}" MethodName="Cancel" />
            </eventTrigger>
        </i:Interaction.Triggers>
    </Button>
</window>
```

Note that `CallMethodAction` is for demonstration purposes only
Use `ICommand` in WPF instead

InteractionRequest
Trigger Action

```
using Microsoft.Practices.Prism.Interactivity.InteractionRequest;
using System.Windows;
using System.Windows.Interactivity;
namespace InteractionRequestSample
{
    public class MessageBoxAction : TriggerAction<FrameworkElement>
    {
        protected override void Invoke(object parameter)
        {
            var eventArgs = parameter as InteractionRequestedEventArgs;
            MessageBox.Show(eventArgs.Context.Content.ToString());
        }
    }
}
```

.NET Training Slides

Q&A

Thank your for coming!



Rainer Stropek
software architects gmbh

Mail	rainer@timecockpit.com
Web	http://www.timecockpit.com
Twitter	@rstropek

time cockpit
Saves the day.
