

Windows Presentation Foundation

- Grafische Basis-APIs von Windows
 - GDI32 (Graphics Device Interface)
 - Mit Hardwareunterstützung, relativ schnell
 - Über 30 Jahre alt
 - Historisch gewachsen, sehr unstrukturiert
- Neuere APIs
 - DirectX
 - Hardwareunterstützung wg. Spielen
 - Komplexe Programmierung
 - GDI+ (mit .NET veröffentlicht)
 - Keine Hardwareunterstützung
 - Weiterentwicklung fraglich

- Oberfläche
 - Schickere Graphiken
 - Transparenz
 - Animationen
 - Hardwareunterstützung
 - 2D- / 3D-Grafiken
 - Medien (Videos, Sound)

- Programmierung und Design
 - Trennung von Darstellung und Logik / Daten
 - Aufgabenteilung Designer und Programmierer
 - Durchgängige und nachvollziehbare Datenbindungsmechanismen
 - Vorlagenbasierte Darstellungen, Austauschbarkeit des Designs

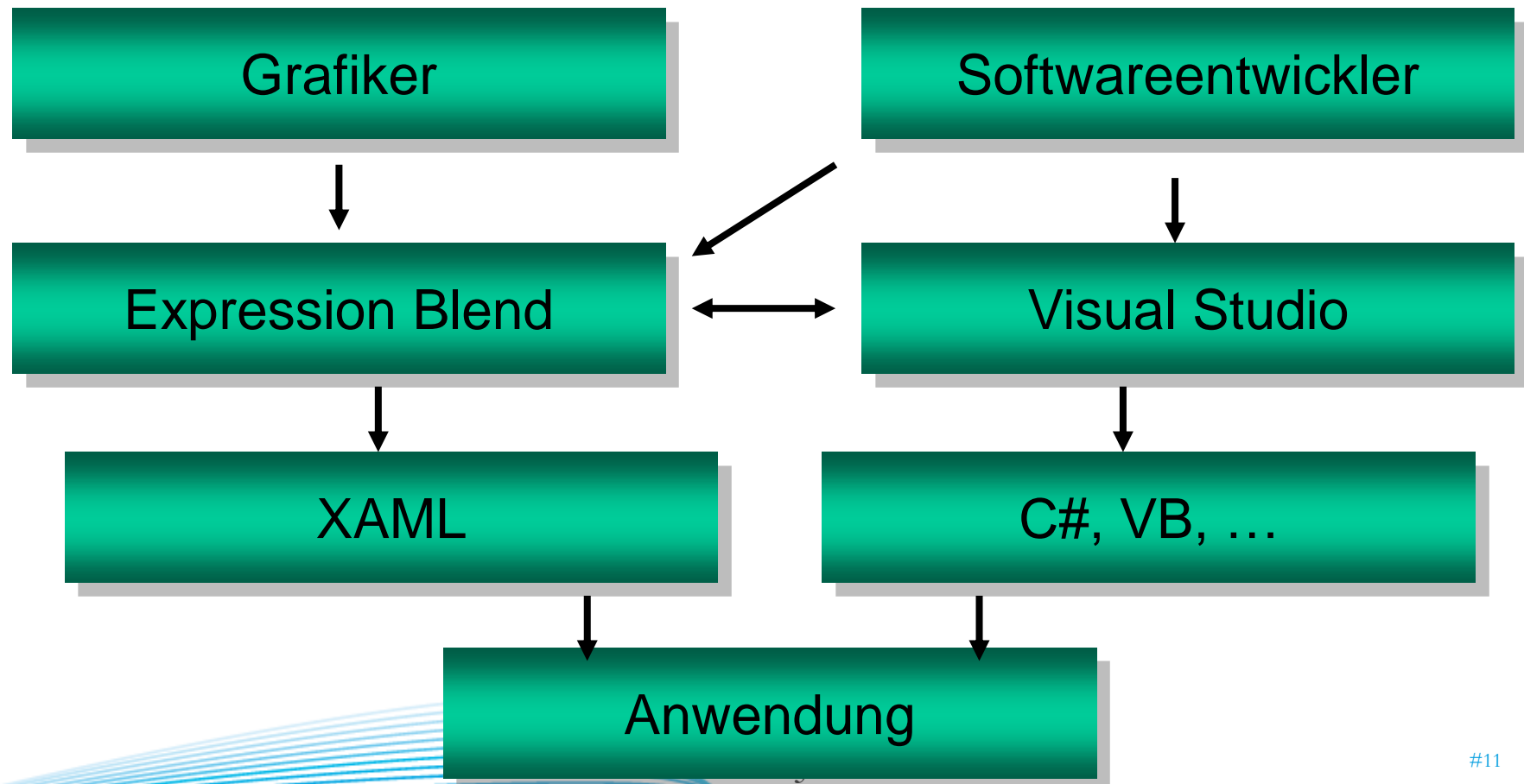
- .NET 3.0 = .NET 2.0 + WPF + WCF + WF
- Setzt auf DirectX auf
 - DirectX 7 oder neuer
 - DirectX 9 oder neuer ab Vista
- Lauffähig ab Windows XP
- Ab .NET 4.5 wird Windows XP nicht mehr unterstützt!

- Vektororientiert (außer Bitmaps)
 - Darstellungsgröße frei skalierbar
 - Hohe Performance durch Hardwareunterstützung
 - Koordinaten mit Gleitkommazahlen
 - 32 Bit ARGB (RGB + Alpha-Transparenz)

Deklarative Programmierung von UI und Animationen

- XAML (eXtensible Application Markup Language)
 - XML-Syntax
 - Aufbau geschachtelter Objekthierarchien
 - Trennung von Design und Code
 - Designerwerkzeuge generieren XAML
 - Entwickler arbeitet mit C# oder VB
 - Werkzeuge
 - Design: Blend
 - Logik: Visual Studio 2017

Zusammenspiel zwischen deklarativer und imperativer Programmierung



- XAML
- Content Controls und Layout Container
- WPF-Ressourcen
- Datenbindungen
- Templates
- Styles
- Commands
- Zusammenspiel, MVVM-Pattern

- Details zu Dependency Properties, Markup Extensions, TypeConverter
- Trigger
- Behaviors
- Transformationen
- Visual States
- Animationen
- Validierungen
- Verschiedene Sprachen unterstützen
- Asynchrone Programmierung
- Eigene Controls mit Control-Templates

XAML-Syntax: Objektdeklarationen

```
<Window x:Class="XamlGrundlagen.XamlBeispiel1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="XamlBeispiel1" Height="300" Width="300">
  <CanvasButton Content="Bitte drücken" Height="34" Width="96"
      Canvas.Left="12" Canvas.Top="12" Click="Button_Click" />
    <Label Canvas.Left="16" Canvas.Top="82" Name="label1">
      <Label.Width>150</Label.Width>
      <Label.Height>50</Label.Height>
      *** Anzeigetext ***
    </Label>
  </Canvas>
</Window>
```

Objektdeklarationen durch Angabe der **Typnamen**

```
<Window x:Class="XamlGrundlagen.XamlBeispiel1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="XamlBeispiel1" Height="300" Width="300">
  <Canvas>
    <Button Content="Bitte drücken" Height="34" Width="96"
      Canvas.Left="12" Canvas.Top="12" Click="Button_Click" />
    <Label Canvas.Left="16" Canvas.Top="82" Name="label1">
      <Label.Width>150</Label.Width>
      <Label.Height>50</Label.Height>
      *** Anzeigetext ***
    </Label>
  </Canvas>
</Window>
```

Angabe der **Namensräume**, ähnlich wie **using** in C# oder **Imports** in VB

```
<Window x:Class="XamlGrundlagen.XamlBeispiel1"  
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
  Title="XamlBeispiel1" Height="300" Width="300">  
  <Canvas>  
    <Button Content="Bitte drücken" Height="34" Width="96"  
      Canvas.Left="12" Canvas.Top="12" Click="Button_Click" />  
    <Label Canvas.Left="16" Canvas.Top="82" Name="label1">  
      <Label.Width>150</Label.Width>  
      <Label.Height>50</Label.Height>  
      *** Anzeigetext ***  
    </Label>  
  </Canvas>  
</Window>
```

Einstellen von **Eigenschaftswerten**, alternativ über Attribute, geschachtelte Elemente oder Textknoten

XAML-Syntax: Attached Dependency Properties

```
<Window x:Class="XamlGrundlagen.XamlBeispiel1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="XamlBeispiel1" Height="300" Width="300">
  <Canvas>
    <Button Content="Bitte drücken" Height="34" Width="96"
      Canvas.Left="12" Canvas.Top="12" Click="Button_Click" />
    <Label Canvas.Left="16" Canvas.Top="82" Name="label1">
      <Label.Width>150</Label.Width>
      <Label.Height>50</Label.Height>
      *** Anzeigetext ***
    </Label>
  </Canvas>
</Window>
```

Attached Dependency Properties sind in der angegebenen Klasse (hier Canvas) definiert, die Werte werden jedoch vom jeweiligen Objekt gespeichert (hier Button bzw. Label)

XAML-Syntax: Events

```
<Window x:Class="XamlGrundlagen.XamlBeispiel1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="XamlBeispiel1" Height="300" Width="300">
  <Canvas>
    <Button Content="Bitte drücken" Height="34" Width="96"
      Canvas.Left="12" Canvas.Top="12" Click="Button_Click" />
    <Label Canvas.Left="16" Canvas.Top="82" Name="label1">
      <Label.Width>150</Label.Width>
      <Label.Height>50</Label.Height>
      *** Anzeigetext ***
    </Label>
  </Canvas>
</Window>
```

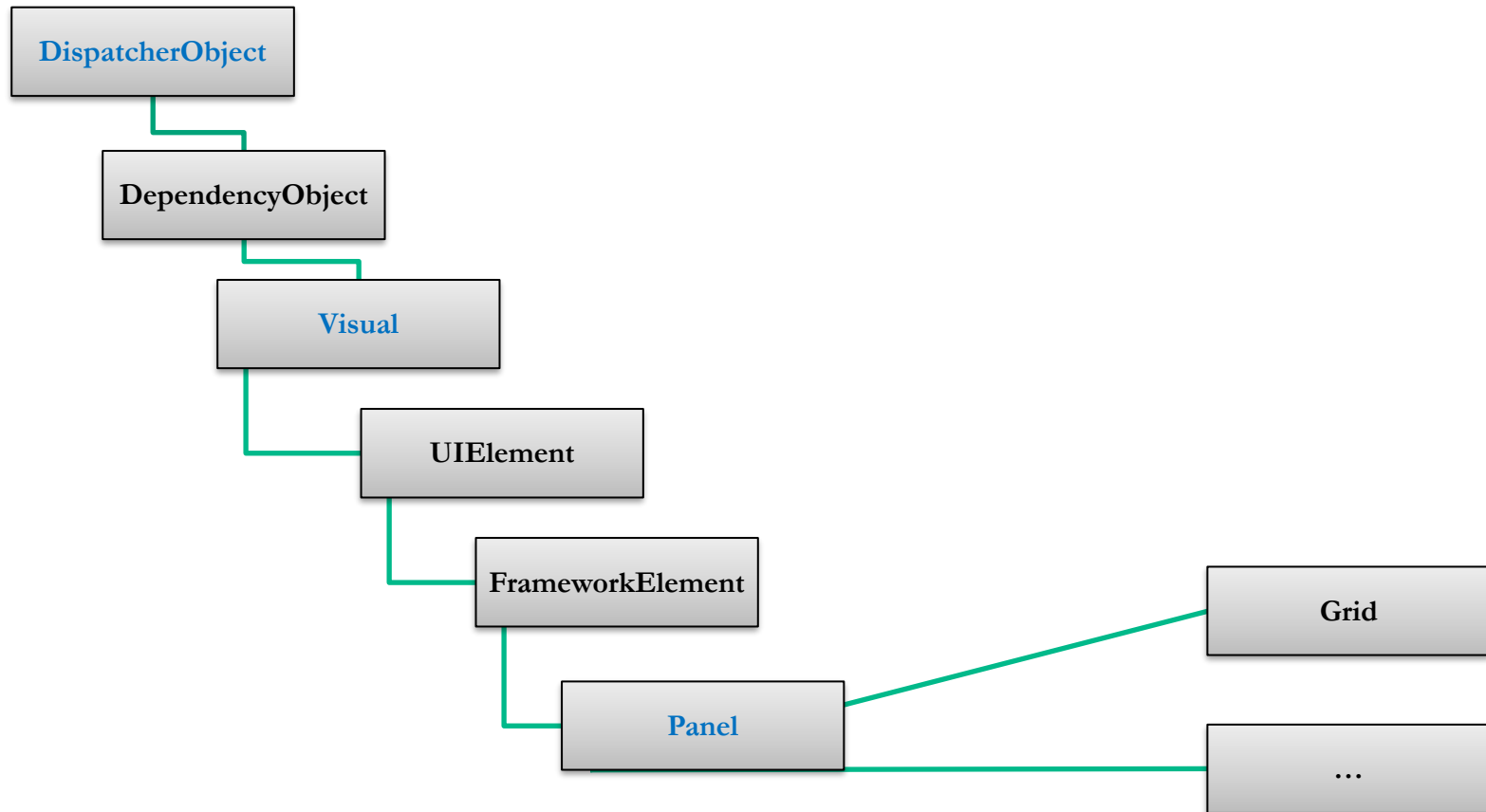
EventHandler werden als Attribut unter Angabe des Methodennamens gebunden

XAML-Syntax

- Elemente = Instanzen von Klassen
- <Element.Eigenschaft> = Eigenschaften
- Attribute = Eigenschaften oder Events
- Attribute oft als Kurzform für komplexe Inhalte
- Attached Dependency Properties
- {} - Erweiterungen

- APP.xaml
 - StartupUri
 - ShutdownMode
 - OnLastWindowClose
 - OnMainWindowClose
 - OnExplicitShutdown
 - Events
 - Startup, Exit, SessionEnding
 - Activated, Deactivated
 - DispatcherUnhandledException

- Größenangaben werden umgerechnet
 - Basis: 1 / 96 Zoll
- Absolute Positionierung mit Bildschirmkoordinaten sollte unbedingt vermieden werden
- Absolute Größen sollten vermieden werden. Die Größe soll sich aus dem Inhalt ergeben
- Container teilen sich den verfügbaren Bereich mit den Kindelementen
- Container können verschachtelt werden



Layout-Container

- Abgeleitet von `System.Windows.Controls.Panel`
- Eigenschaften
 - Children
 - Background
 - `IsItemsHost`

Core Layout Panels

- StackPanel
- WrapPanel
- DockPanel
- Grid
- UniformGrid
- Canvas

- Eigenschaften des Containers
 - Orientation
- Eigenschaften der Kindelemente
 - HorizontalAlignment
 - VerticalAlignment
 - Margin
 - MinWidth, MinHeight
 - MaxWidth, MaxHeight
 - Width, Height

Beispiel

DockPanel und WrapPanel

- DockPanel
 - Andocken der Kindelemente oben, links, rechts oder unten
 - Attached Dependency Properties
 - DockPanel.Dock
 - Eigenschaften des Containers
 - LastChildFill
- WrapPanel
 - Umbruch, wenn der Platz nicht ausreicht
 - Eigenschaften des Containers
 - Orientation

Beispiel

Beispiel

- ColumnDefinitions
 - Breite (absolut oder relativ)
- RowDefinitions
 - Höhe (absolut oder relativ)
- Typischerweise ein Kindelement pro Zelle
- Zuordnung der Elemente über Grid.Row und Grid.Column
- Grid.RowSpan, Grid.ColumnSpan

Beispiel

Grid – einheitliche Spaltenbreite

- `SharedSizeGroup`
 - definiert Spalten gleicher Breite
 - Identifizierung über Namensangabe
- `Grid.IsSharedSizeScope`
 - gibt an, dass ein Container `SharedSizeGroup`-Definitionen berücksichtigen soll
- Einheitliche Spaltenbreite bei variierenden Inhalten (z. B. bei Lokalisierung)
- Kann zum Anpassen mehrerer Grids verwendet werden

GridSplitter

- Ändern der Spaltenbreiten und Zeilenhöhen durch den Anwender
- Ein GridSplitter muss einer Zelle zugeordnet werden
- Best Practice:
 - leere Zeilen oder Spalten für GridSplitter freihalten
 - RowSpan / ColumnSpan setzen
 - VerticalAlignment und HorizontalAlignment einstellen

Beispiel

UniformGrid

- Gleichgroße Zellen
- Einstellung über Rows- und Columns-Eigenschaften
- Elemente werden automatisch den Zellen zugeordnet
- Sonderform des Grids

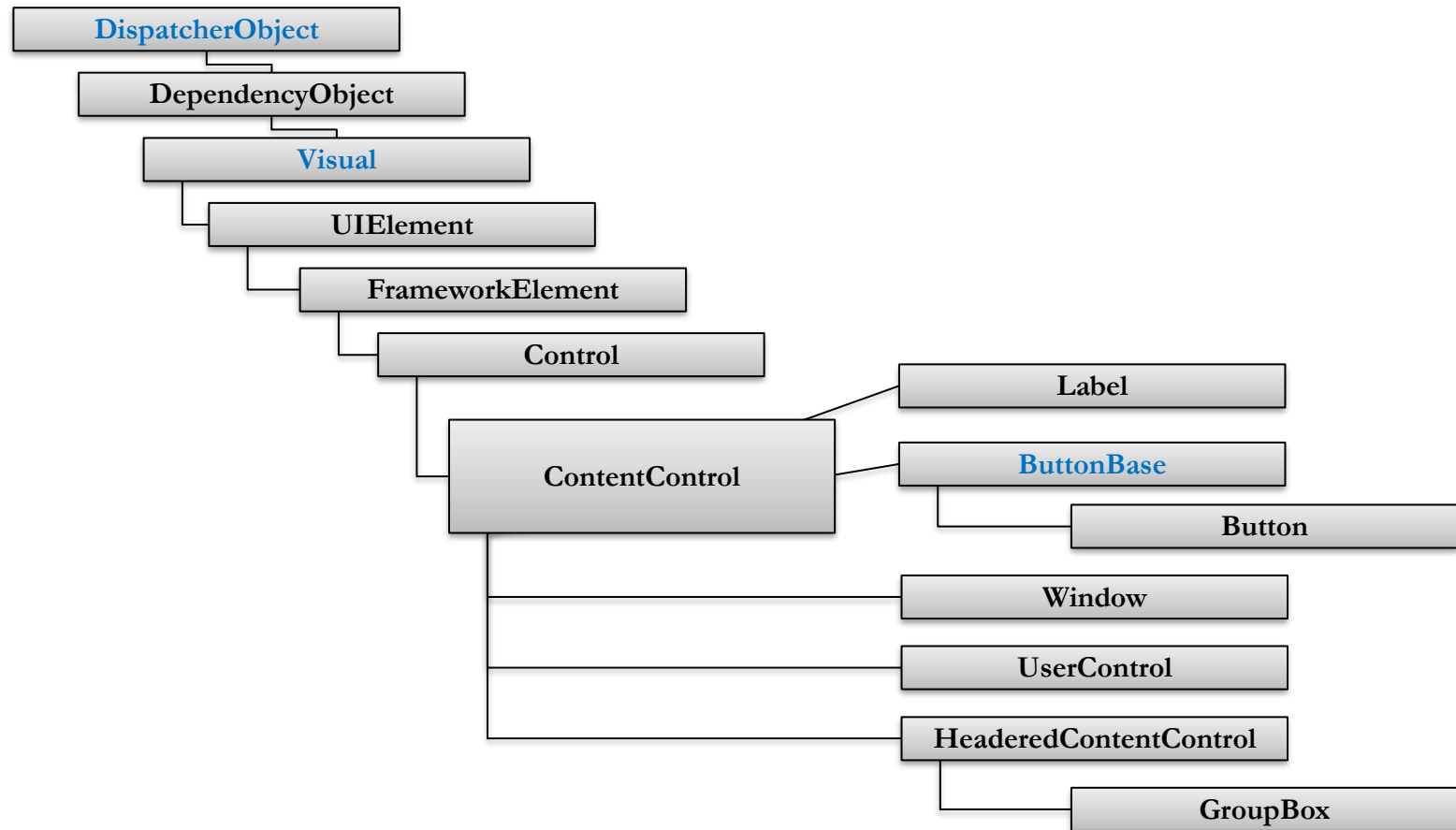
Beispiel

- Absolute Koordinaten
- Einstellung über
 - Canvas.Top
 - Canvas.Bottom
 - Canvas.Left
 - Canvas.Right
- Überlappung steuerbar durch Canvas.ZIndex

Beispiel

- Enthalten ein einziges Kindelement als Inhalt
- Abgeleitet von ContentControl
- Beispielklassen:
 - Label
 - ButtonBase
 - ToolTip
 - ScrollView
 - UserControl
 - Window
 - HeaderedContentControl

Klassenhierarchie Content Controls



Content-Eigenschaft

- Kindelemente, die nicht von UIElement abgeleitet sind
 - Aufruf von ToString() um den Text zu lesen
- Kindelemente, die von UIElement abgeleitet sind
 - Darstellung durch UIElement.OnRender()

- ContentControl mit zusätzlichem Titel
 - GroupBox
 - TabItem (für TabControl)
 - Expander
 - ExpandDirection

Beispiel

Beispiel

Spezialisierte Container

- **ScrollView**
 - Zeigt automatisch Scrollbalken an, wenn der Inhalt nicht vollständig gezeigt werden kann
- **Border**
 - Umrahmt Controls wie StackPanel
- **ViewBox**
 - Maximiert den Inhalt

Beispiel

Beispiel

- Anzeige von Texten
- Eigenschaft Text
 - Typ: String
- Oder als Unterelement mit Formatierungen ähnlich HTML

```
<TextBlock>
  <Span Foreground="Green">
    <Run Text="Hallo "/>
    <Bold>
      <Run Text="Welt"/>
    </Bold>
  </Span>
  <Run Text=" Hier gehts zu: "/>
  <Hyperlink NavigateUri="www.aldi.de">
    <Run Text="Aldi"/>
  </Hyperlink>
</TextBlock>
```

- Umbruch über Eigenschaft TextWrapping einstellbar

- Key-/Value-Liste eines FrameworkElements
- Oberste Ebene: App.xaml
- Bereitstellen beliebiger Objekte
- Einbindung über die Markup-Extensions{StaticResource} oder {DynamicResource}

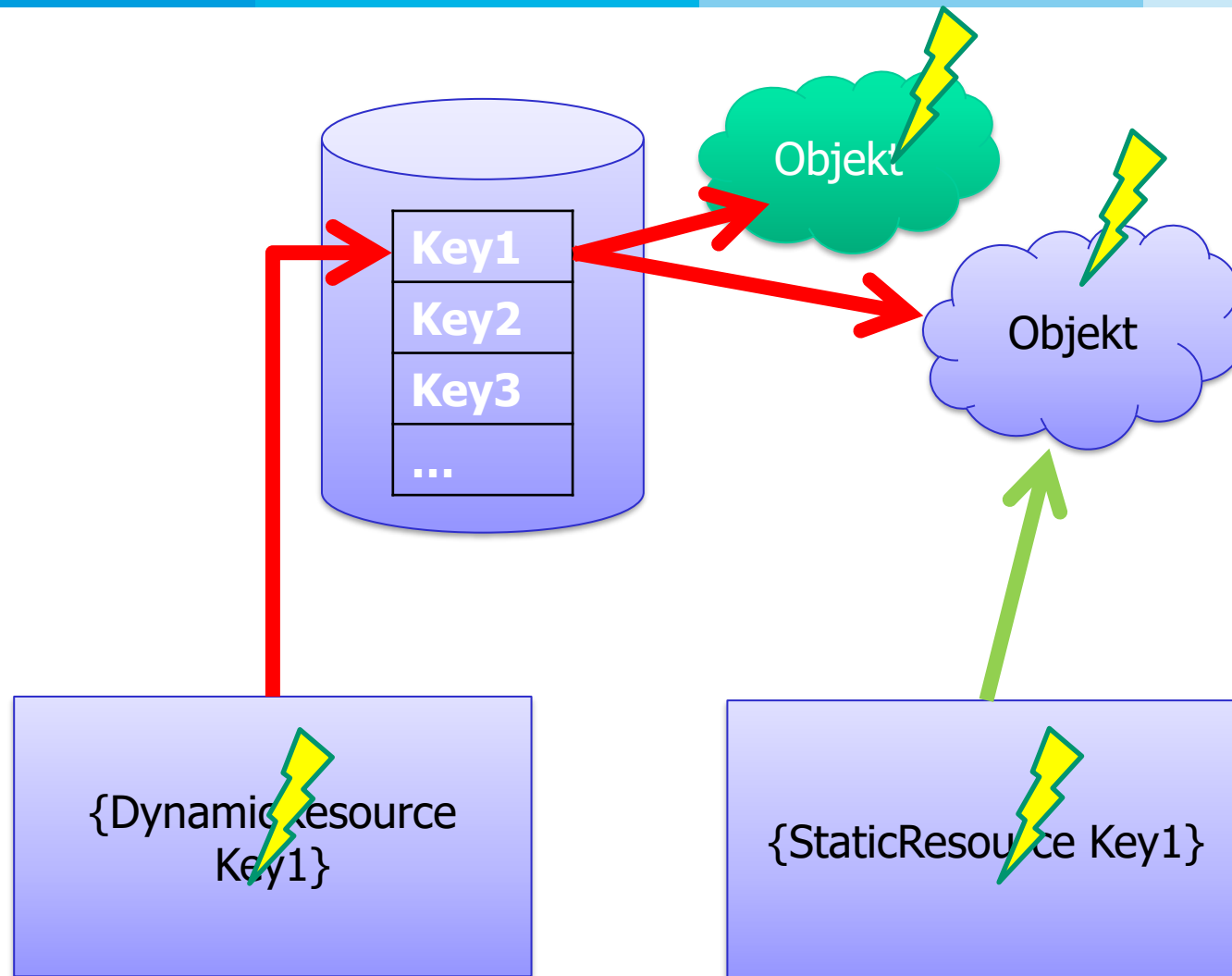
```
<Window.Resources>
  <LinearGradientBrush x:Key="LGB1"
    StartPoint="0,0" EndPoint="1,0" >
    <LinearGradientBrush.GradientStops>
      <GradientStop Offset="0" Color="Red"/>
      <GradientStop Offset="0.7" Color="Yellow"/>
      <GradientStop Offset="1" Color="Green"/>
    </LinearGradientBrush.GradientStops>
  </LinearGradientBrush>
</Window.Resources>
```

```
<Rectangle Fill="{StaticResource LGB1}"
  Grid.Column="1" />
<Rectangle Fill="{DynamicResource LGB1}"
  Grid.Column="1" Grid.Row="1"/>
```

Ablage von Ressourcen

- Resources-Knoten beliebiger FrameworkElement-Object-Elemente im Visual Tree
- Resources-Knoten in App.xaml
- ResourceDictionary (eingebunden z. B. in App.xaml)

StaticResource vs. DynamicResource



- Quellen
 - können beliebige Objekttypen sein
 - UI-Elemente ({Binding ElementName=...})
 - DataContext (hierarchisch im Visual Tree)
 - Explizite Objektreferenzen ({Binding Source=...})
 - Relative Angaben (Visual Tree, Templated Parent)
- Ziel
 - MUSS eine Dependency Property sein!

- Nähere Spezifikation der Quelle über Pfadangabe ({Binding Path=...})
- Ein Endpunkt in einem Quellobjekt MUSS eine öffentliche Eigenschaft sein (**public**)
 - Die Bindung an mit **internal** gekennzeichnete Eigenschaften ist NICHT möglich
 - Die Bindung an öffentliche Felder ist ebenfalls NICHT möglich

- Angabe der Richtung (Mode=...)
 - OneWay, TwoWay, OneWayToSource, OneTime
- Angabe des Benachrichtigungsverhaltens (UpdateSourceTrigger=...)
 - LostFocus, PropertyChanged
- Konvertierungshilfen
 - Converter, ConverterParameter
- Textformatierung
 - StringFormat
- Delay
 - Zeitverzögerung für Bindung Ziel -> Quelle

Vorsicht!

Es gilt die für XML eingestellte Kultur

Im Normalfall ist das en-US!!! ☹

Änderung entweder über xml:lang oder

```
FrameworkElement.LanguageProperty.OverrideMetadata(  
    typeof(FrameworkElement),  
    new FrameworkPropertyMetadata(  
        XmlLanguage.GetLanguage(  
            CultureInfo.CurrentCulture.Name)  
        )  
    );
```

Datenbindungen

Benachrichtigungen bei Änderungen

- Automatisch durch Dependency-Property-Infrastruktur bei vom Typ `DependencyObject` abgeleiteten Klassen
- Implementierung von **`INotifyPropertyChanged`** in anderen Datentypen
- Implementierung von `INotifyCollectionChanged` bei Auflistungen (z. B. `ObservableCollection`)

Templates

- Data-Templates
- Control-Templates
- HierarchicalDataTemplate

DataTemplate

- Beschreibt die visuelle Repräsentation für ein Objekt
- Besitzt genau ein Unterelement
- Kann bei Definition als Ressource als Standard für einen bestimmten Datentyp vorgesehen werden, wenn statt des Keys die Eigenschaft DataType gesetzt wird
- Typische Eigenschaften: ContentTemplate, ItemTemplate
- Kann Trigger enthalten (DataTrigger...)

- Definieren Eigenschaftswerte von Objekten
- Können als Standard für bestimmte Objekttypen festgelegt werden (TargetType=...)
- Es können nur Dependency Properties gesetzt werden
- Können Trigger enthalten
- Können voneinander erben (BasedOn=...)

ControlTemplate






- Beschreibt die visuelle Repräsentation eines Controls
- Teilweise ist die Angabe des Controltyps erforderlich
- Kann nicht direkt allgemein als Standard für Controls eines bestimmten Typs festgelegt werden (indirekt jedoch möglich über Styles)
- Kann Trigger enthalten

Themes als Ressource hinzufügen

The screenshot shows the NuGet Package Manager interface. At the top, there are tabs for 'Browse', 'Installed', and 'Updates' (with a blue badge '1'). The 'Browse' tab is selected. Below the tabs is a search bar containing the text 'wpf theme material'. To the right of the search bar are icons for refreshing and a checkbox for 'Include prerelease'. The search results are displayed in a list. The first result is 'MahApps.Metro.Resources' by Jan Karger, Dennis Daume, and Brendi, version v0.6.1. The second result is 'MaterialDesignThemes' by James Willock, version v2.2.1.750, with 81,9K downloads. The third result is 'MaterialDesignThemes.MahApps' by James Willock, version v0.0.9, with 15,7K downloads. The fourth result is 'Infragistics.Themes.MetroDark.Wpf' by Brian Lagunas, version v1.0.0, with 8,3K downloads. The fifth result is 'Infragistics.Themes.MetroLight.Wpf' by Brian Lagunas, version v1.0.0, with 7,78K downloads.

[Browse](#) Installed Updates **1** NuGet Packag

wpf theme material x ↻ ☐ Include prerelease P

-  **MahApps.Metro.Resources** by Jan Karger, Dennis Daume, Brendi v0.6.1
"loose" file XAML Resource for Metro UI's.
All icons taken from "Modern UI Icons" created by Austin Andrews
-  **MaterialDesignThemes** by James Willock, **81,9K** download: ✓ v2.2.1.750
ResourceDictionary instances containing Material Design templates and styles for WPF controls.
-  **MaterialDesignThemes.MahApps** by James Willock, **15,7K** dov v0.0.9 ↓
ResourceDictionary instances containing Material Design templates and styles for WPF controls in the MahApps library.
-  **Infragistics.Themes.MetroDark.Wpf** by Brian Lagunas, **8,3K** ✓ v1.0.0
The Infragistics WPF MetroDark Theme provides control theming for 19 of the most common Microsoft built-in WPF controls.
-  **Infragistics.Themes.MetroLight.Wpf** by Brian Lagunas, **7,78K** v1.0.0
The Infragistics WPF MetroLight Theme provides control theming for 19

Commands

- Ansatzweise Implementierung des Command-Patterns
- Vorgefertigte Command-Klassen (über 100) ersetzen das Vergeben von Namen
- Eigene Command-Klassen erstellbar
- Ein Command kann an mehrere Controls gebunden werden
- Über das Command können die Controls enabled / disabled werden

- **ApplicationCommands**
 - Clipboard, Undo, Redo, Open, Close, Print, ...
- **NavigationCommands**
 - Back, Forward, NextPage...
- **ComponentCommands**
 - Verarbeitung von Informationen aus Controls (ScrollUp, ScrollDown, MoveToEnd usw.)
- **EditingCommands**
 - Textformatierungsbefehle wie Bold, Italic, Alignment ...
- **MediaCommands**
 - Abspielen von Medien (Play, Pause, Lautstärke, Track selection usw.)

Command-Bindungen

```
<MenuItem Header="Textbox" Click="MenuItem_Click">
    <MenuItem Header="Speichern" Name="MnuSpeichern"/>
    <MenuItem Header="Neu" Command="ApplicationCommands.New"/>
</MenuItem>
```

```
<Window.CommandBindings>
    <CommandBinding x:Name="StopCommand"
        Command="ApplicationCommands.Stop"
        Executed="CommandBinding_Executed"
        CanExecute="CommandBinding_CanExecute"/>
    <CommandBinding Command="ApplicationCommands.New"
        Executed="NewCommandBinding_Executed"
        CanExecute="NewCommandBinding_CanExecute" />
</Window.CommandBindings>
```

```
private void NewCommandBinding_Executed(object sender, Ex
{
    lastContent = "";
    EditText.Text = "";
}

private void NewCommandBinding_CanExecute(object sender,
{
    e.CanExecute = EditText.Text != "";
}
```

Der Haken daran

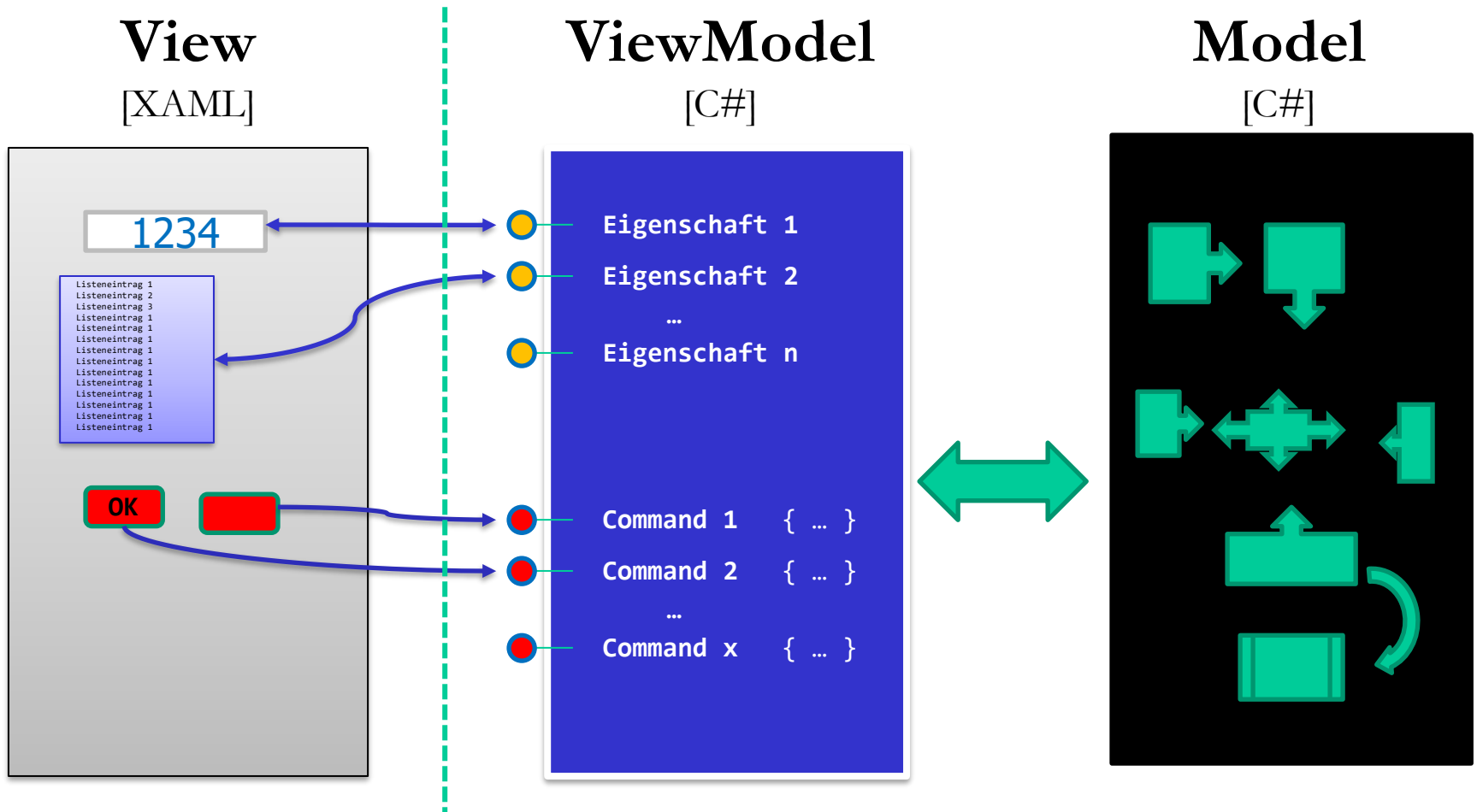
- Die Commands müssen an Eventhandler gebunden werden, die im CodeBehind der Fensterklasse anzulegen sind
- Daraus ergibt sich eine ungünstige Verknüpfung zwischen Oberfläche und Logik

- Eigene Command-Klasse entwerfen, die über eine Rückrufmethode die Aktion verknüpft
- Im ViewModel solche Command-Objekte über Eigenschaften für die Datenbindung bereitstellen

Beispiel

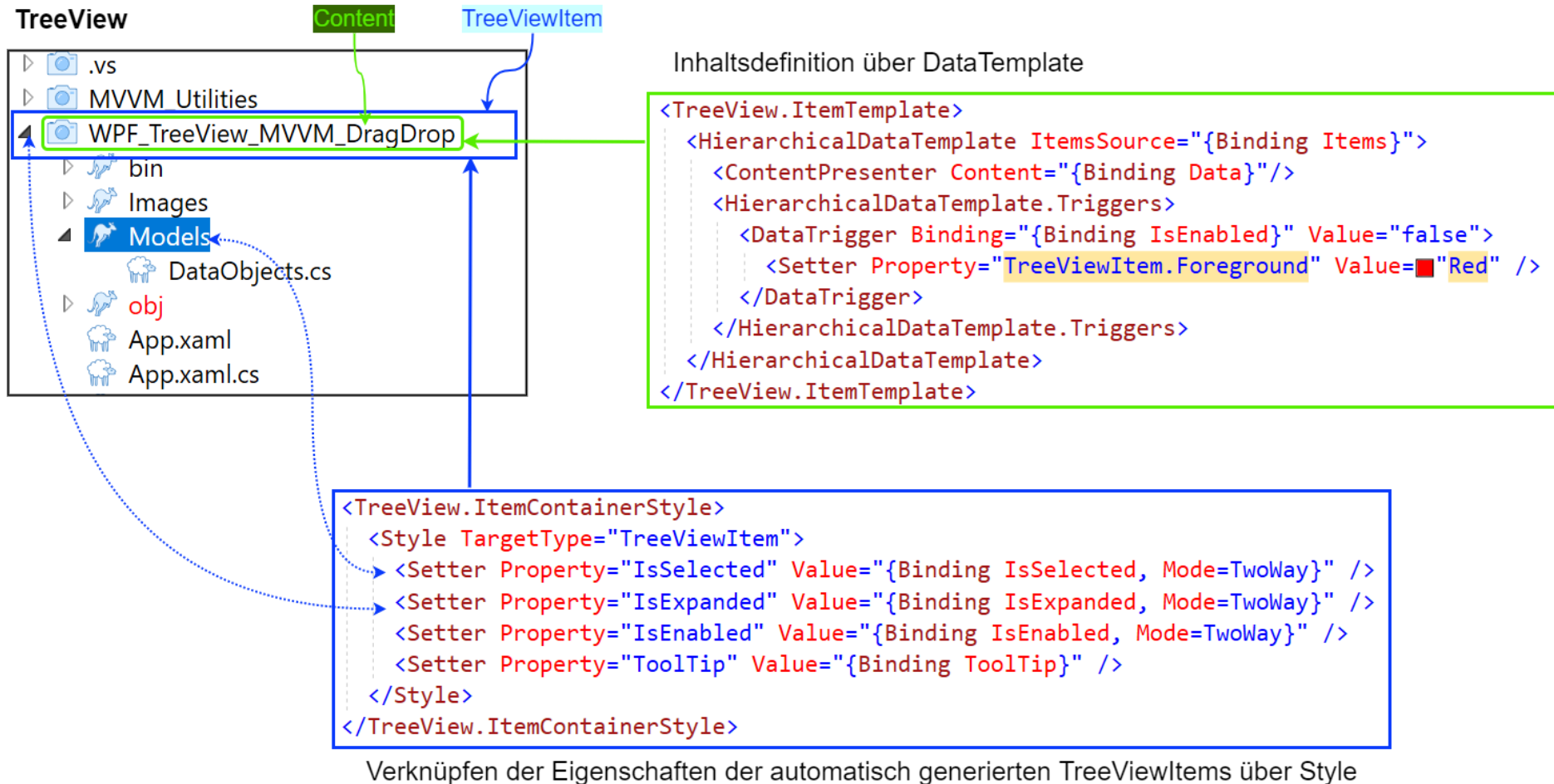
- Trennung von C#- und XAML-Code
- ViewModel bereitet die Daten des Models auf
- Kein Bezug im ViewModel auf Elemente der View
- View wird über Databinding mit ViewModel verknüpft
- Model und ViewModel können ohne View mit Unittests getestet werden
- View ist ohne Änderung des ViewModels änder- und austauschbar

Model-View-ViewModel

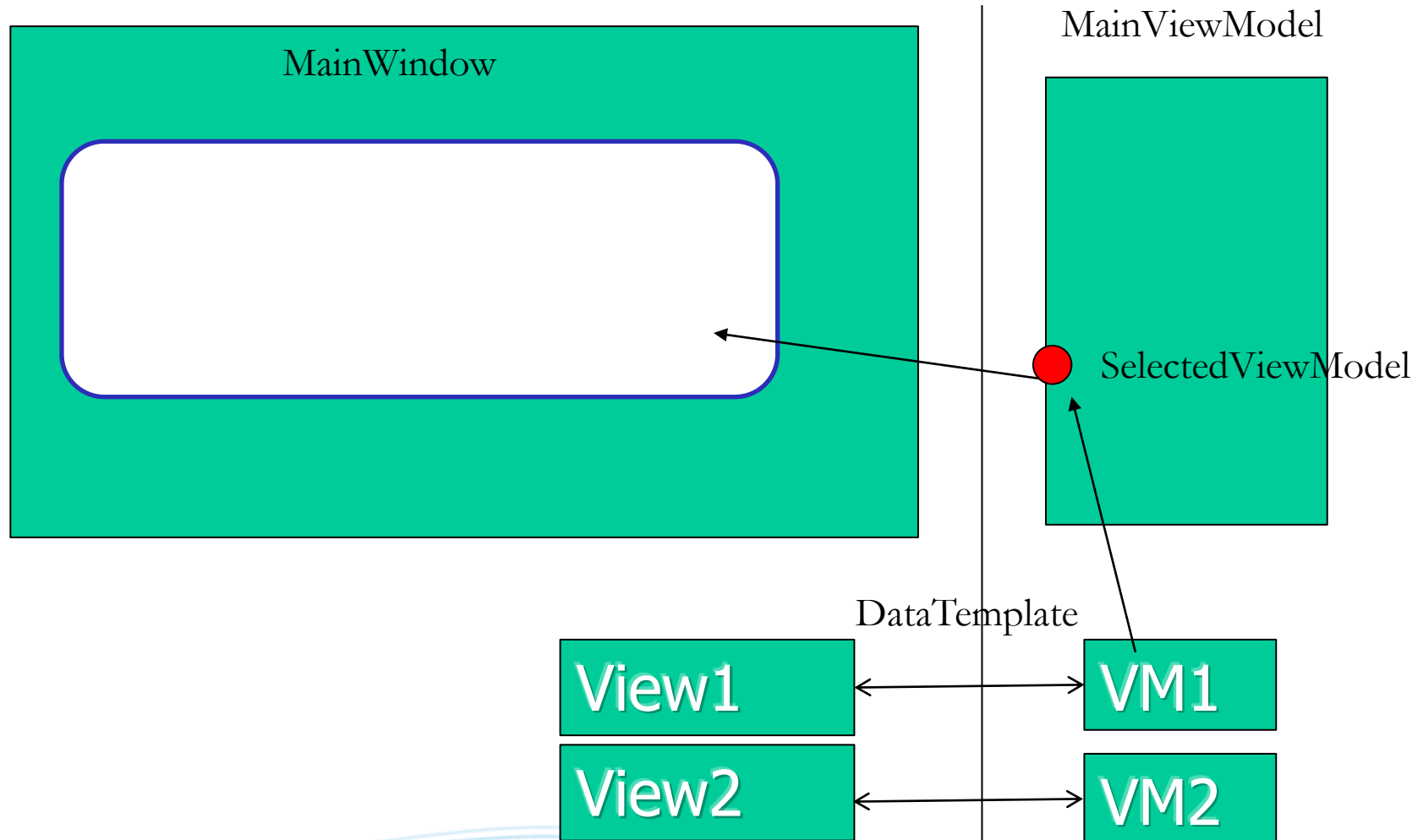


- INotifyPropertyChanged
- Command-Klassen
- Behaviors für das Auslösen von Commands durch beliebige Events
- Hilfsklassen für Auflistungen
- DataTemplates für die ViewModel-Klassen
- Jede View als UserControl implementieren

ItemContainerStyle Beispiel TreeView



MVVM mit mehreren Views (VM first)



- Standard: Einbetten jedes einzelnen Namensraums
 - Nachteile
 - Viele xmlns-Einbindungen
 - Viele Präfixes
 - Die Namensräume jeder Assembly müssen einzeln eingebunden werden

- Attribute in der assembly.info.cs
 - [XmlnsDefinition]
 - bildet einen oder mehrere CLR-Namensraum(-räume) auf einen WPF-Namensraum ab
 - [XmlnsPrefix]
 - Gibt Standardpräfix für Tools und Toolbox vor

```
[assembly: XmlnsDefinition("http://schemas.seminar.wpf", "TestLib")]  
[assembly: XmlnsDefinition("http://schemas.seminar.wpf", "TestLib.SubNS")]  
[assembly: XmlnsPrefix("http://schemas.seminar.wpf", "seminar")]
```

- Vereinfachte XAML-Syntax
 - Einige Varianten
 - {EineErweiterung}
 - {EineErweiterung Wert}
 - {EineErweiterung Eigenschaft=Wert}
 - {EineErweiterung Wert1, Wert2}
 - {EineErweiterung Eigenschaft1=Wert1, Eigenschaft2=Wert2}
 - {EineErweiterung Wert, Eigenschaft1=Wert1, ...}
- Realisiert durch Klasse, abgeleitet von MarkupExtension
 - `class EineErweiterungExtension : MarkupExtension`

- Werte setzen über Eigenschaften
- Werte setzen über Konstruktoren
- Wertübergabe in **ProvideValue**-Überschreibung
 - Abfrage von Ziel-Property und Typ möglich
- Gedacht für einfache Erweiterungen
 - nicht für komplexe DB-Abfragen etc.
- Beispiele aus WPF
 - Binding, StaticResource, x:Null

Community Toolkit

- <https://github.com/CommunityToolkit/dotnet>

WPF-Performance

- [https://msdn.microsoft.com/de-de/library/aa970683\(v=vs.110\).aspx](https://msdn.microsoft.com/de-de/library/aa970683(v=vs.110).aspx)
- <http://www.codeproject.com/Articles/784529/Solutions-for-WPF-Performance-Issue>
- <http://pelebyte.net/blog/2011/07/11/twelve-ways-to-improve-wpf-performance/>
- [https://msdn.microsoft.com/de-de/library/aa969767\(v=vs.110\).aspx](https://msdn.microsoft.com/de-de/library/aa969767(v=vs.110).aspx)