1 Einführung WPF <Window xmlns="http://schemas.microsoft.com/winfx/2006/</pre> xaml/presentation" WPF: Windows Presentation Foundation xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" 1.1 Layout/Grössen xmlns:d="http://schemas.microsoft.com/expression/blend Layout in C# oder XAML geschrieben. XAML ist leichter und kürzer. Als Grösseneinheit wird DIP (Device Independent Pixels) verwendet. xmlns:mc="http://schemas.openxmlformats.org/markupcompatibility/2006" 1.2 Hello WPF xmlns:local="clr-namespace:Vorlesung_09" Dateien: mc:Ignorable="d" • App.xaml: Markup der Startup-Klasse ... /> • App.xaml.cs: Coed-Behind der Startup-Klasse • MainWindow.xaml: Markup des Hauptfensters 2.1.3 Named Elements Elemente können benannt werden. → Ermöglicht Zugriff auf Code-• MainWindow.xaml.cs: Code-Behind des Hauptfensters Behind. Attribut führt zu Property in generierter Klasse • Assembly Info.cs: Projekt spezifische Meta-Daten // XAML: Deployment: <TextBlock Name="WpfAttribute" Text="WPF" /> • Framework-Dependent Executable (FDE): .NET Core muss manuell <TextBlock x:Name="XamlAttribute" Text="XAML" /> installiert werden. Erzeugt sehr kleines Binary. // Code Behind: • Self-Contained Deployment (SCD): .NET Core in Binary integriert. this.WpfAttribute.Text = "..."; → Sehr grosses Binary (150MB für hello world) this. XamlAttribute. Text = "..."; 2 GUI-Programmierung 2.1.4 Syntaxen 2.1 XAML Allgemein // Attribute Syntax: Beschreibungssprache von Microsoft zur Gestaltung graphischer Oberflä-<Button Background="Blue" Foreground="Red" XAML XAML BAML + Content="Mein Button" /> Code Prozessor C# Code // Property Element Syntax: <Button> Compiler Executable <Button.Background> C# <SolidColorBrush Color="Blue"/> Code </Button.Background> <Button.Foreground> Für Design kann auch C# verwendet werden, XAML ist jedoch leichter, <SolidColorBrush Color="Red"/> kürzer, lesbarer und hat einen Designer. Microsoft Blend für das Designen. </Button.Foreground> 2.1.1 Visual Tree und Logical Tree <Button.Content> Mein Button </Button.Content> </Button> OK 2.1.5 Type Converters Visual Tree (grün + blau) <local:LocationControl Center="10, 20" /> · Vollständiger, gezeichneter Baum // Control: public class LocationControl : TextBlock { · Enthält Elemente, die wir nicht selber definieren public Location Center { Logical Tree (grün) set => this.Text = \$"{value.Lat} / {value.Long}"; · Vereinfachung des vollen Baums · Umfasst die durch uns definierten XAML-Elemente // Model: 2.1.2 Namespaces [TypeConverter(typeof(LocationConverter))] Mit xmlns werden XML-Namespaces definiert. public class Location { → Ohne Doppelpunkt: Standard-Namespace (Elemente können ohne Präpublic double Lat { get; set; } fix verwendet werden) public double Long { get; set; } → Mit Doppelpunkt: Nenannter Namespace (Elemente können nur mit Präfix verwendet werden) // Type Converter: Übliche Namespaces in WPF: public class LocationConverter : TypeConverter { • Der Standard-Namespace wird auf die WPF Control Library gesetzt public override object ConvertFrom(ITypeDescriptorContext context, • x für XAML-spezifische Elemente CultureInfo culture, • d für Elemente des visuellen Designers

• mc für Elemente der «Markup Kompatibilität»

• local für Elemente aus unserem eigenen Assembly

object value) {

//Zur Kürzung des Beispiels auf Checks verzichtet:

// - Ist value wirklich ein string?

```
</Button>
2.1.7 Markup Extensions
Erlauben die Erweiterung des XAML-Markup mit zusätzlicher Logik. Die
Logik wird in geschweiften Klammern platziert { ... }. Verwendet bei Sty-
ling und Data Binding.
// XAML:
<TextBlock Text="{local:LocationExtension Lat=10,Long</pre>
    =20}" />
// Marup Extension:
public class LocationExtension : MarkupExtension {
   public string Lat { get; set; }
   public string Long { get; set; }
   public override object ProvideValue (IServiceProvider
      return this.Lat + " / " + this.Long;
2.1.8 Attached Properties
Setzt Eigenschaft auf einem Element, die zu einem anderen Element ge-
hört. Die Eigenschaft wird sozusagen einem anderen Element angehängt.
<Grid>
   <Grid.RowDefinitions>
      <RowDefinition Height="30" />
      <RowDefinition Height="20" />
      <RowDefinition Height="10" />
   </Grid.RowDefinitions>
   <TextBlock Grid.Row="0" Name="G" Background="Green"
       />
   <TextBlock Grid.Row="1" Name="R" Background="Red" />
   <TextBlock Grid.Row="2" Name="B" Background="Blue" />
</Grid>
```

// - Enthält das Array exakt 2 Elemente?

var valueArray = valueAsString.Split(',');

Lat = Convert.ToDouble(valueArray[0]),

Long = Convert.ToDouble(valueArray[1])

Jedes XAML-Element kann genau eine Eigenschaften als seinen Inhalt defi-

nieren. Einige Elemente können, neben reinem Text, auch andere Elemente

TextAlignment="Center"

Foreground="#888888" />

FontSize="20" />

<TextBlock Text="Und hier klein"

FontSize="12"

var valueAsString = (string) value;

return new Location {

};

enthalten.

2.1.6 Content Properties

<Button Content="Label" />

<Button Width="150" Height="60">

<TextBlock Text="Gross"

<Button>Label</Button>

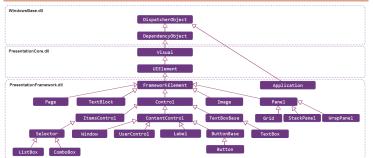
<StackPanel>

</StackPanel>

// - Sind die strings zu double konvertierbar?

2.2 Grundelemente

2.2.1 Klassenhierarchie

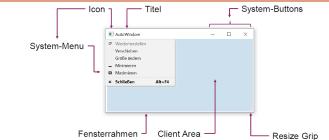


2.2.2 Application

Einstiegspunkt in die Anwendung. Main()-Methode in generiertem Code. Erzeugt Application-Instanz. Definiert via StartupUri die erste View.

```
// XAML:
<Application x:Class="Vorlesung_09.App"</pre>
  StartupUri="MainWindow.xaml">
</Application>
// Code Behind:
public partial class App : Application{ ]
// Generated Code:
// Nur ein Auszug
public partial class App : System. Windows. Application {
  public void InitializeComponent() {
      this.StartupUri = new System.Uri("MainWindow.xaml"
          , System. UriKind. Relative);
   public static void Main() {
      Vorlesung_09.App app = new Vorlesung_09.App();
      app.InitializeComponent();
      app.Run();
```

2.2.3 Window - Sichtbare Elemente



2.2.4 Window - Wichtige Eigenschaften

- Title Name des Fensters
- Icon Icon des Fensters
 - Bild mit Build Action "Resource"hinzufügen
 - Verschiedene Dateiformate unterstützt
- ShowInTaskbar Sichtbarkeit in Taskleiste
- WindowStyle Aussehen des Fensters
- WindowStartupLocation Anzeigeposition
- ResizeMode Modus zur Grössenänderung

2.2.5 UIElement

Wichtigste Basisklasse für visuelle WPF-Elemente.

Definiert grundlegende Elemente, Methoden und Events:

IsEnabled: Reagiert das Element auf Interaktionen?

IsFocused: Ist das Element gerade aktiv?

Visibility: Ist das Element sichtbar? → z.B. Collapsed (Unsichtbar, keinen Platz), Hidden (Unsichtbar, belegt Platz), Visible (Sichtbar), etc.

2.2.6 FrameworkElement

Erweitert UIElement um zusätzliche Funktionalität, unter anderem:

- Name-Property für Zugriff
- Logical Tree
- Layout System
- Visuelles Styling (Woche 10)
- Data Binding (Woche 11)

Grössenangaben:

Width, Height und Margin, Kein Padding. Zusätzlich MinWidth, Max-Width und MinHeight, MaxHeight.

Dimensionen:

Auto: Automatische Grösse (wrap content) px: Device Independent Pixels, 1in == 96px

Ausrichtungen:

· Alignment beeinflusst die Ausrichtung innerhalb des Eltern-Elements



2.2.7 Control

Basis-Klasse für Controls mit Benutzerinteraktion.

Erweitert Framework Element um zusätzliche Funktionalität: Gestaltungsmöglichkeiten (Farben, Schriften, Ränder), Ausrichtungen der Kind-Elemente, Control Templates (Woche 10)

Rahmen/Ränder:

Neue Eigenschaften: padding (Innenabstand), BorderThickness (Rahmenstärker), CornerRadius (Radius für abgerundete Ecken)

Grössenangaben für Margin und Padding:

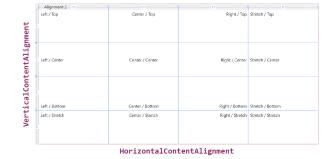
n - Selber Wert für alle Seiten

x,y - X für Horizontal, Y für Vertikal

l,t,r,b - Links, Oben, Rechts, Unten

Ausrichtung:

• ContentAlignment beeinflusst die Ausrichtung der Kind-Elemente



Farben/Schriften:

Farbgebung mit Brushes ("Pinsel"): Foreground, Background, Border-

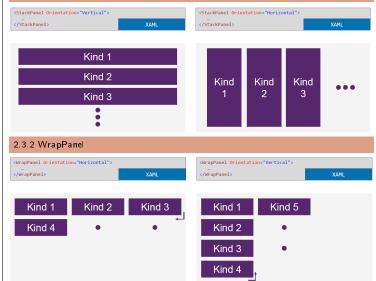
Schriftbild: FontFamily, FontSize, FontStretch, FontStyle, FontWeight

2.3 Layouts

Layouts sind Container für Kind-Elemente. Haben eine Parent-Child Beziehung. Verschachtelung ist möglich. Verfügbare Layouts in WPF:

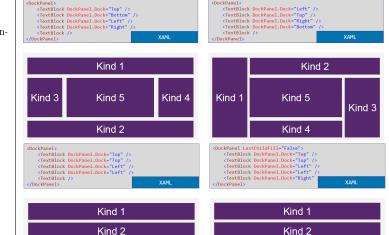
- StackPanel Horizontale oder vertikale Auflistung
- WrapPanel Wie Stack, aber mit Zeilen-/Spaltenumbruch
- DockPanel Kinder werden an Seiten/im Zentrum "angedockt"
- Grid Kinder werden den Zellen einer Tabelle zugeordnet





2.3.3 DockPanel

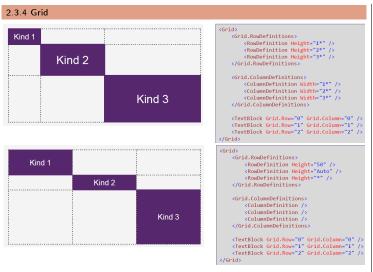
Kind 3 Kind 4



Kind 3 Kind 4

Kind 5

Kind 5



3 GUI-Design

3.1 Controls

Das Aussehen von Controls wird über Attribute beeinflusst.

3.1.1 Image

- Bilddatei zum Projekt hinzufügen
 - Build Action: Resource
- Integration in Binärdatei des Projekts
- Source für Dateipfad
 - Relativer Pfad beginnend bei XAML-Datei
 - Verwendung von Ordnern möglich
- Stretch für Kontrolle der Skalierung

<Border BorderThickness="2"

- Uniform: Bildverhältnis beibehalten (Standard)
- Fill: Fläche füllen, Bildverhältnis ignorieren
- UniformToFill: Fläche füllen, Bildverhältnis beibehalten
- None: Bild gemäss Originalgrösse darstellen

<Image Source="../Bilder/Logo.jpg" Stretch="Uniform" />

3.1.2 Border

Container für genau ein Element (Controls, oder Layouts). Verwendung zur Gruppierung oder Hervorhebung von Inhalten via Rahmen, Hintergrundfarbe, Runde Ecken, Sichtbarkeit

```
BorderBrush="Black"
   Background="#f0f0f0">
   <StackPanel>
      <Button Content="Button 1" Margin="5" />
      <Button Content="Button 2" Margin="5" />
      <Button Content="Button 3" Margin="5" />
   </StackPanel>
</Border>
```

3.1.3 Canvas

2D-Zeichenfläche für einfache geometrische Objekte (Shapes). Absolute Positionierung in X/Y-Raster (Keine Layout-Logik. Kind-Elemente erhalten Attached Porperties)



```
Height="200"
      Background="#6E1C50">
   <Window.Clip>
      <RectangleGeometry RadiusX="30"</pre>
                     RadiusY="30"
                     Rect="0,0,400,200" />
   </Window.Clip>
   <Grid>
      <Label Content="Clipped Window"</pre>
           HorizontalAlignment="Center"
           VerticalAlignment="Center"
           Foreground="White" />
  </Grid>
</Window>
```

Beliebige Objekte, die in XAML definiert werden können: Brush, Color,

Resourcen besitzten eine eindeutige Identifikation. Zuweisung des XAML-Attributs mit x:key. Dies erlaubt später eine Referenzierung.

3.2.1 Resource Directory

- Container zur Speicherung von Resources
- Zugriff über Schlüssel der Resource (x:Key)
- Teil aller FrameworkElement-Ableitungen (Zugriff über Property Element Syntax)
- Beispiele:
- Application.Resources
- Window.Resources
- Button.Resources
- Label.Resources

3.2.2 Verwendung von Resources

```
Ziel: Objekte zentral definieren und n-fach wiederverwenden
// XAML
<Window>
 <Window.Resources>
  <SolidColorBrush x:Key="OSTBrush" Color="#6E1C50" />
 </Window.Resources>
 <StackPanel>
   <Label Content="Variante 1" Foreground="White">
    // Property Element Syntax
    <Label.Background>
      <StaticResource ResourceKey="OSTBrush" />
```

```
</StackPanel>
</Window>
```

// via FrameworkElement.FindResource(...)

var brush = FindResource("OSTBrush") as Brush;

// Attribute Syntax mit Markup Extension

// Attribute Syntax mit Markup Extension

Background="{StaticResource OSTBrush}" />

Background="{StaticResource ResourceKey=OSTBrush

3.2.3 Auflösung von Resources

</Label.Background>

<Label Content="Variante 2"</pre>

} " />

<Label Content="Variante 3"

Foreground="White"

Foreground="White"

</Label>

Suchreihenfolge (bricht beim ersten Treffer ab): 1. Aktuelles Element und alle Parent-Elemente

2. In Application. Resources 3. In System-Ressourcen

3.2.4 Statische und dynamische Ressourcen

Statische Resources:

- Einmalige Auswertung der Resource
- Auswertung bei Kompilierung
- Unveränderlich zur Laufzeit
- Extension: {StaticResource Key}
- Dynamische Resources:

• Mehrfache Auswertung der Resource

- Auswertung bei Ausführung
- Veränderlich zur Laufzeit
- Extension: {DynamicResource Key}

```
<Window>
 <Window.Resources>
  <SolidColorBrush x:Key="OSTBrush" Color="#6E1C50" />
 </Window.Resources>
```

// XAML

</Window>

<StackPanel> <Label Content="OK"

Foreground="White" Background="{DynamicResource OSTBrush}" /> <Button Content="Update"

Click="UpdateResource" /> </StackPanel>

// Code Behind

private void UpdateResource(object sender, RoutedEventArgs e) { Resources["OSTBrush"] = new SolidColorBrush(Colors.

Blue);

3.2.5 Beliebige Typen

Ein Resource Dictionary nimmt alle Elemente auf, die in XAML definier-

bar sind. Aufig werden Basistypen benötigt wie strings, Zahlen, etc. // Stystem.Runtime enthält Basistypen

```
<Window xmlns:s="clr-namespace:System;assembly=System.</pre>
                                                                    // Mit oder ohne TargetType
                                                                                                                                   <Window.Resources>
    Runtime">
                                                                    <Style x:Key="MyButtonStyle" TargetType="Button">
                                                                                                                                    <Style x:Key="BrushButton" TargetType="Button">
                                                                      <Setter Property="Background" Value="Blue" />
 <Window.Resources>
                                                                                                                                      <Setter Property="Background">
   <s:Double x:Key="MarginVertical">2</s:Double>
                                                                      <Setter Property="Foreground" Value="Black" />
                                                                                                                                        <Setter.Value>
                                                                      <Setter Property="BorderBrush" Value="Black" />
   <s:Double x:Key="MarginHorizontal">5</s:Double>
   <Thickness x:Key="Margin"
                                                                      <Setter Property="BorderThickness" Value="1" />
      Top="{StaticResource MarginVertical}"
                                                                    </Style>
      Bottom="{StaticResource MarginVertical}"
                                                                   </Window.Resources>
      Left="{StaticResource MarginHorizontal}"
                                                                   <StackPanel>
      Right="{StaticResource MarginHorizontal}" />
                                                                    <Button Style="{StaticResource MyButtonStyle}"</pre>
 </Window.Resources>
                                                                       Content="OK" />
</Window>
                                                                    <Button Style="{StaticResource MyButtonStyle}"</pre>
                                                                       Content="Cancel" />
                                                                                                                                    </Style>
3.2.6 Zugriff auf CLR-Werte
                                                                  </StackPanel>
Gelegentlich ist es nötig, auf statische Werte der CLR zuzugreifen. Zugriff
                                                                 </Window>
via Markup Extension: x:Static. Keine WPF-Resources. Werte definiert in
normalen Klassen.
                                                                 3.3.2 Implizite Styles
                                                                                                                                   3.3.5 Trigger
                                                                 Ohne Key wirkt der Style für alle Controls des angegeben Typs.
public static class MyRes {
                                                                 <Window>
   public static SolidColorBrush OSTBrush = new
                                                                   <Window.Resources>
       SolidColorBrush (Color.FromRqb (110, 28, 80));
                                                                    <Style TargetType="Button"> // Style erhält
                                                                         automatisch den key x:key="{x:Type Button}"
                                                                                                                                   <Window.Resources>
                                                                      <Setter Property="Background" Value="Blue" />
                                                                                                                                    <Style x:Key="TriggerButton" TargetType="Button">
// XAML
                                                                      <Setter Property="Foreground" Value="Black" />
<Label Content="x:Static"
                                                                      <Setter Property="BorderBrush" Value="Black" />
  Background="{x:Static local:MyRes.OSTBrush}"
                                                                      <Setter Property="BorderThickness" Value="1" />
  Foreground="{x:Static SystemColors.ControlLightBrush}"
                                                                    </Style>
  FontFamily="{x:Static SystemFonts.CaptionFontFamily}"
                                                                   </Window.Resources>
  FontSize="{x:Static SystemFonts.CaptionFontSize}" />
                                                                   <StackPanel>
                                                                    // Keine Style Attribute mehr nötig
3.2.7 Eigenständige Resource Dictionaries
                                                                    <Button Content="OK" />
In separater .xaml-Datei mit XML-Root <ResourceDictionary>. In andere
                                                                    <Button Content="Cancel" />
Dictionaries als Merged Dictionary integrierbar.
                                                                   </StackPanel>
// MyDictionary.xaml
                                                                 </Window>
<ResourceDictionary>
                                                                 3 3 3 Styles erweitern
 <SolidColorBrush x:Key="OSTBrush2" Color="#6E1C50" />
</ResourceDictionary>
                                                                 Styles sind mit Inline-Attributen kombinierbar (eignet sich für einmalige
// MainWindow.xmal
                                                                 Anpassungen). Styles können auch vererbt werden. → Kann Umfang der
<Window>
                                                                                                                                   3.3.6 Themes
                                                                 Ressourcen reduzieren
 <Window.Resources>
                                                                                                                                   WPF hat kein Themes-Konzept. Kann aber nachgebaut werden:
                                                                 // Mit Inline Attribute
  // Für Merges zwingend, sonst optional
                                                                                                                                   Dazu gleiche x:Key styles in mehreren Resource Dictionaries definieren.
                                                                 <Button Style="{StaticResource NormalButton}"</pre>
   <ResourceDictionary>
                                                                    Background="Red"
                                                                                                                                   Laden des gwünschten Dictionary zur Laufzeit und Zuweisung der Styles
    <SolidColorBrush x:Key="OSTBrush" Color="#6E1C50" />
                                                                                                                                   über DynamicResource.
                                                                    Content="Cancel" />
    <ResourceDictionary.MergedDictionaries>
      <ResourceDictionary Source="MyDictionary.xaml"/>
                                                                 // Vererbung
    </ResourceDictionary.MergedDictionaries>
                                                                                                                                   • Control Templates beschreiben die visuelle Repräsentation von XAML-
                                                                 <Window.Resources>
   </ResourceDictionary>
                                                                                                                                     Controls (Elemente, die im Visual tree eingefügt werden)
                                                                  <Style x:Key="NormalButton" TargetType="Button">
 </Window.Resources>
                                                                                                                                   • Der Zugriff auf das aktuelle Template erfolgt über das Attribut Con-
 <Label Content="Externe Resource"</pre>
                                                                   </Style>
                                                                                                                                     trol.Template
   Foreground="White"
                                                                   <Style x:Key="DangerButton"
   Background="{StaticResource OSTBrush2}" />
                                                                                                                                   • Eigene Control Templates: Option1: Als ControlTemplate-Resource.
                                                                      BasedOn="{StaticResource NormalButton}"
</Window>
                                                                                                                                     Option2: Innerhalb eines Styles
                                                                      TargetType="Button">
3.2.8 Externe Resources
                                                                    <Setter Property="Background" Value="Red" />
                                                                                                                                   ContentPresenter:
Dictionaries können via Pack URI aus anderen Assemblies eingebunden
                                                                  </Style>
werden.
                                                                 </Window.Resources>
                                                                                                                                   TemplateBinding:
                                                                 <Button Style="{StaticResource DangerButton}"</pre>
Resources können mehrfach verwendet werden, müssen aber bei jedem Ele-
                                                                                                                                   verwendbar.
                                                                    Content="Cancel" />
ment referenzieren. Das gibt Duplizierten Code.
                                                                                                                                   3.5 Guidelines
3.3.1 Explizite Styles
                                                                 3.3.4 Komplexe Werte
                                                                                                                                   Keine expliziten, bzw. ernstzunehmenden Guidelines für WPF verfügbar.
                                                                 Können über setzten des Value-Attributs auf dem Setter Element verwen-
                                                                                                                                   Empfehlung: Guidelines und Libraries nutzen, die sich andernorts bewährt
<Window>
 <Window.Resources>
                                                                 det werden. Beispiel eines Gradient Hintergrund:
                                                                                                                                   haben. z.B. Material Design in XAML, MahApps.Metro
```

<LinearGradientBrush StartPoint="0,0" EndPoint="</pre> <GradientStop Offset="0" Color="Red" /> <GradientStop Offset="0.5" Color="Yellow" /> <GradientStop Offset="1" Color="Red" /> </LinearGradientBrush> </Setter.Value> </Setter> </Window.Resources> <Button Style="{StaticResource BrushButton}" Content="</pre> Brush" /> Trigger erlauben Stylings basierend auf dem Zustand eines Elementes. Beliebige Attribute des Elements sind auswertbar. Beispiel: Veränderung des Cursors abhängig vom Button-Label:

<Style.Triggers> <Trigger Property="Content" Value="Link"> <Setter Property="Cursor" Value="Hand" /> </Trigger> <Trigger Property="Content" Value="Edit"> <Setter Property="Cursor" Value="Pen" /> </Trigger> </Style.Triggers> </Style> </Window.Resources> <Button Style="{StaticResource TriggerButton}" Content="</pre> Link" /> <Button Style="{StaticResource TriggerButton}" Content="</pre> Edit" />

3.4 Control Templates

Platzhalter für den Content des Elements

Markup Extension für das Binding an Attribute. Nur in Control Templates

```
4 Data Binding
                                                                   4.2.1 DataBinding - Mode
                                                                   • OneTime - Einmalige Aktualisierung des Ziels beim Setzen der Quelle
       XAML
                                                  C# Klassen
                           Data Binding
                                                                   • OneWay - Ziel wird bei Änderungen der Quelle aktualisiert
                                                                   • OneWayToSource - Quelle wird bei Änderungen des Zieles aktualisiert
                                                     Domäne
    Name Content
                                                                   • TwoWay - Änderungen werden in beide Richtungen propagiert
                           Code Behind
                                                                   • Default - Wert abhängig von Ziel-Eigenschaft
                                                                   4.2.2 DataBinding - Value Converter
                            Event Handler
                                                   Business Logik
    Content Click
                                                                   Datenumwandlung zwischen Quelle und Ziel. Bsp: Bool zu Visibi-
                                                                   lity, Strings konvertieren. Mithilfe von IValueConverter:
                                                                   Convert(...) - Quelle zu Ziel. ConvertBack(...) - Ziel zu Quelle
                            Controller
                                                     Model
4.1 Data Binding in WPF
// User.cs
public class User {
   public string FirstName { get; set; } = "Joel";
   public string LastName { get; set; } = "Schaltegger";
// XAML
<Window>
 <StackPanel>
   // DataBinding mit Murkup Extension
   <Label Content="{Binding FirstName}" />
   <Label Content="{Binding LastName}" />
 </StackPanel>
</Window>
// Code Behind
public partial class MainWindow : Window {
   private readonly User user:
public MainWindow()
   InitializeComponent();
   user = new User();
   this.DataContext = user; // Datenquelle für
        DataBinding
4.1.1 Binding Typen
Bindings verknüpfen Ziel und Quelle miteinander.
                                                                   Converter.
Binding: für 1:1 Verknüpfungen
MultiBinding: für 1:n Verknüpfungen
                                                                   <TextBlock>
PriorityBinding: für 1:n / 1:1 Verknüpfungen
4.2 Binding Typen
Path: Name der Quell-Eigenschaft. Objektpfad-Syntax möglich (z.B.
Mode: Richtung des Datenflusses.
Converter: Datenumwandlung zwischen Quelle und Ziel.
<!-- Attribute Syntax + Markup Extension -->
<TextBox Text="{Binding Path=FirstName,
   Mode=TwoWay,
                                                                   </TextBlock>
   Converter={StaticResource MyCnv}}" />
<!-- Property Element Syntax -->
<TextBox>
   <TextBox.Text>
      <Binding Path="FirstName"
         Mode="TwoWay"
          Converter="{StaticResource MyCnv}" />
   </TextBox.Text>
</TextBox>
```

```
Erzeugung von Converter:
Option1: In resources (StaticResource). Option2: In Code (x:static)
// Parameter der Methoden gekürzt zwecks Lesbarkeit
public class ReverseConverter : IValueConverter {
   public object Convert(object value, ...) {
      var stringValue = (string) value;
      var reversedChars = stringValue.Reverse().ToArray
      var reversedString = new string(reversedChars);
      return reversedString;
   public object ConvertBack(object value, ...) {
      return Convert(value, ...);
4.2.3 DataBinding - Weitere Eigenschaften
• Delay - Verzögerung in Millisekunden bei Updates vom Ziel zu Quelle
• StringFormat - Formatangabe für Bindings mit dem Zieltyp string
• FallbackValue - Ergebnis, wenn Binding fehlschlägt (z.B. falscher Pfad)
• Target Null Value - Ergebnis, wenn Quell-Eigenschaft null liefert
• UpdateSourceTrigger - Zeitpunkt, zu welchem das Quell-Element ak-
  tualisiert wird. z.B. LostFocus oder PropertyChanged; Standard ab-
  hängig vom Ziel
4.2.4 Multi Binding
Verwendung analog zu Binding. Unterschiede: Beliebig viele Quell-
Eigenschaften. Nur Property Element Syntax. Converter mit IMultiValue-
 <TextBlock.Text>
   // { } startet das "Escaping": nachfolgende Zeichen
       als String interpretieren
   <MultiBinding StringFormat="{}{0} {1} ({2} Jahre)">
     <Binding Path="FirstName" />
     <Binding Path="LastName" />
     <Binding Path="Age" />
   </MultiBinding>
 </TextBlock.Text>
4.2.5 Data Context
• Property der Klasse FrameworkElement
• Setzt die Standardquelle für Bindings
• Falls undefiniert: Traversierung des Logical Trees nach oben bis zum
  ersten Treffer
• Jeder Path ist relativ zum DataContext
• Beliebige Objekte möglich: C#-Klassen, WPF-Elemente, etc. Typi-
  scherweise: View Models (Woche 12)
```

```
4.2.7 Weitere Quellen
Mit RelativeSource werden Elemente im Visual Tree referenziert:
<Label Content="{Binding RelativeSource={RelativeSource}</pre>
    FindAncestor, AncestorType=Window}, Path=Title}" />
Mit ElementName werden Elemente über Namen referenziert:
<TextBox Name="MyText" Text="Hallo MGE" />
<TextBox Text="{Binding ElementName=MyText, Path=Text}"
4.2.8 Design Time Support
Der XAML Designer kennt den Typ des Objekts im Data Context stan-
dardmässig nicht. Das heisst: keine Autovervollständigung verfügbar (In-
telliSense). Als Abhilfe kann das Attribut d:DataContext beim Window
gesetzt werden:
// Variante 1: Objekterzeugung in XAML und Markup
    Extension {d:DesignInstance ... }
d:DataContext="{d:DesignInstance Type=local:User,
    IsDesignTimeCreatable=True}"
// Variante 2: Objekterzeugung in C# und Markup
    Extension {x:Static ... }
d:DataContext="{x:Static local:DesignerData.User}"
4.3 Aktualisierung von Daten
4.3.1 POCOs als Data Context
Als Datenquelle können beliebige Objekte verwendet werden, also auch
POCOs (Plain Old CLR Objects). Unsere Bindings funktionieren - aller-
dings nur mit Einschränkungen. Besser: INotifyPropertyChanged.
4.3.2 INotifyPropertyChanged
Bestandteil des .NET Framework. Ein Interface mit nur einem Event. Na-
me des geänderten Property in EventArgs.
public interface INotifyPropertyChanged {
   event PropertyChangedEventHandler PropertyChanged;
public delegate void PropertyChangedEventHandler(object
    sender, PropertyChangedEventArgs eventArgs);
public class PropertyChangedEventArgs : EventArgs {
   public PropertyChangedEventArgs(string propertyName)
      this.PropertyName = propertyName;
   public virtual string PropertyName { get; }
4.3.3 Beispiel INotify Property Changed
Ohne Hilfsmittel:
// User.cs
public class User : INotifyPropertyChanged {
 // Property mit Zugehörigem Backing Field
 private string _firstName = "Joel";
 public string FirstName {
  get => _firstName;
    if ( firstName != value) {
```

firstName = value:

Der Data Context lässt sich für einzelne Elemente anpassen.

Option1: Im Code Behind das Property DataContext für das Element er-

→ Eher unüblich: Meist wird ein Data Context pro Window verwendet.

4.2.6 Data Context überschreiben

setzen. Option2: Source im Binding setzen.

```
// User.cs
                                                                public class User {
                                                                                                                                 // XAML
                                                                   public string FirstName { get; set; } = "Joel";
                                                                   public string LastName { get; set; } = "Schaltegger";
 // Implementation von INotifyPropertyChanged
                                                                // XAML
 // Event Invoker: Erzeugt Argumente und löst Event aus
                                                                <Window>
 protected virtual void OnPropertyChanged(string name) {
                                                                  <ListBox ItemsSource="{Binding}">
  var eventArgs = new PropertyChangedEventArgs(name);
                                                                   <ListBox.ItemTemplate>
                                                                     <DataTemplate>
                                                                       <StackPanel>
                                                                        <TextBlock Text="{Binding LastName}"/>
                                                                        <TextBlock Text="{Binding FirstName}"/>
                                                                       </StackPanel>
                                                                     </DataTemplate>
                                                                   </ListBox.ItemTemplate>
                                                                  </ListBox>
                                                                </Window>
      set => SetProperty(ref _firstName, value);
                                                                // Code Behind
                                                                public partial class MainWindow : Window {
                                                                   private ObservableCollection<User> users;
                                                                   public MainWindow() {
                                                                      InitializeComponent();
                                                                       // Die Collection müsste natürlich befüllt werden
 // Implementation von INotifyPropertyChanged
                                                                      users = new ObservableCollection<User>();
                                                                                                                                 7 Xamarin und Ausblick
                                                                      this.DataContext = users;
 // Event Invoker: Erzeugt Argumente und löst Event aus
 protected virtual void OnPropertyChanged(string name) {
   var eventArgs = new PropertyChangedEventArgs(name);
                                                                4.4.3 Item Template als Resource
                                                                Das Item Template kann als Resource definiert werden. Das Template ist
 // CallMemberName wird in Namen des Property
                                                                so wiederverwendbar und der XAML Code schlanker. Durch das Attribut
                                                                DataType ist IntelliSense gewährleistet.
 protected bool SetProperty<T>(ref T field, T value, [
                                                                <Window>
      CallerMemberName] string name = null) {
                                                                  <Window.Resources>
                                                                   <DataTemplate x:Key="UserTemplate"</pre>
                                                                      DataType="local:User">
                                                                     <StackPanel>
                                                                      <TextBlock Text="{Binding LastName}"/>
                                                                      <TextBlock Text="{Binding FirstName}"/>
                                                                     </StackPanel>
                                                                   </DataTemplate>
                                                                  </Window.Resources>
                                                                  <ListBox ItemsSource="{Binding}"
Um eine Collection zu binden muss die Quelle INotifyCollectionChanged
                                                                     ItemTemplate="{StaticResource UserTemplate}" />
implementieren und die Ziel-Eigenschaft eine Collection erwarten.
                                                                </Window>
                                                                4.4.4 Selector
Ist Bestandteil des .NET Framework. Enthält (wie INPC)
                                                                Erweitert ItemsControl um Logik zur Selektion von Elementen. Definiert
nur ein Event. Collection-Änderung wird in Event Args beschrieben.
                                                                folgende wichtigen Eigenschaften:
ObserverableCollection<T> implementiert INPC und INCC.
                                                                • SelectedIndex - Index des ausgewählten Elements
                                                                 • SelectedItem - Ausgewähltes Element als Objekt
Wichtigste Basisklasse für WPF-Elemente zur Anzeige von Collections.
                                                                 • Selected Value – Wert des ausgewählten Elements
                                                                 • SelectedValuePath - Objektpfad-Syntax zum Wert, der in SelectedVa-
                                                                  lue zurückgeliefert wird
                                                                // User.cs
• ItemTemplate - Definiert das Template für die Darstellung eines Items
                                                                public class User
→ Bei Verwendung von ItemsSource wird Items zu einer read-only Eigen-
                                                                   public string FirstName { get; set; } = "Joel";
```

public string LastName { get; set; } = "Schaltegger";

OnPropertyChanged(nameof(FirstName));

public event PropertyChangedEventHandler

PropertyChanged?. Invoke(this, eventArgs);

PropertyChanged;

public class User : BindableBase {

public abstract class BindableBase :

public event PropertyChangedEventHandler

PropertyChanged?. Invoke (this, eventArgs);

INotifyPropertyChanged {

if (Equals(field, value)) {

OnPropertyChanged (name);

Definiert folgende Ziel-Eigenschaften:

• Items - Enthält angezeigte Elemente

schaft umgewandelt. Keine Kombination!

• ItemsSource - Füllt Inhalt über Data Binding ab

PropertyChanged;

umgewandelt

return false;

4.4.1 INotifyCollectionChanged

4.4.2 ItemsControl

field = value;

return true:

public string FirstName {

get => _firstName;

private string _firstName = "Joel";

Mit Basisklasse User.cs:

// BindableBase.cs

// User.cs

```
<Window>
 <ListBox ItemsSource="{Binding Users}"
  ItemTemplate="{StaticResource UserTemplate}"
  SelectedIndex="{Binding SelectedUserIndex}"
  SelectedItem="{Binding SelectedUser}"
  SelectedValue="{Binding SelectedUserFirstName}"
  SelectedValuePath="FirstName" />
</Window>
// Code Behind
public partial class MainWindow : Window {
  public ObservableCollection<IUser> Users { get; }
  public IUser SelectedUser { get; set; }
  public int SelectedUserIndex { get; set; }
  public string SelectedUserFirstName { get; set; }
  public MainWindow()
     InitializeComponent();
     // Wie gehabt - hier irgendwie "Users" befüllen
     this.DataContext = this;
5 MVVM
6 Architektur und fortgeschrittene Themen
```