1 Einführung WPF

WPF: Windows Presentation Foundation

1.1 Layout/Grössen

Layout in C# oder XAML geschrieben. XAML ist leichter und kürzer. Als Grösseneinheit wird DIP (Device Independent Pixels) verwendet.

1.2 Hello WPF

Dateien:

- App.xaml: Markup der Startup-Klasse
- App.xaml.cs: Coed-Behind der Startup-Klasse
- MainWindow.xaml: Markup des Hauptfensters
- MainWindow.xaml.cs: Code-Behind des Hauptfensters
- AssemblyInfo.cs: Projektspezifische Meta-Daten

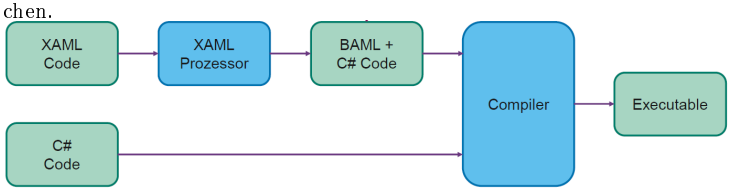
Deployment:

- Framework-Dependent Executable (FDE): .NET Core muss manuell installiert werden. Erzeugt sehr kleines Binary.
- Self-Contained Deployment (SCD): .NET Core in Binary integriert. → Sehr grosses Binary (150MB für hello world)

2 GUI-Programmierung

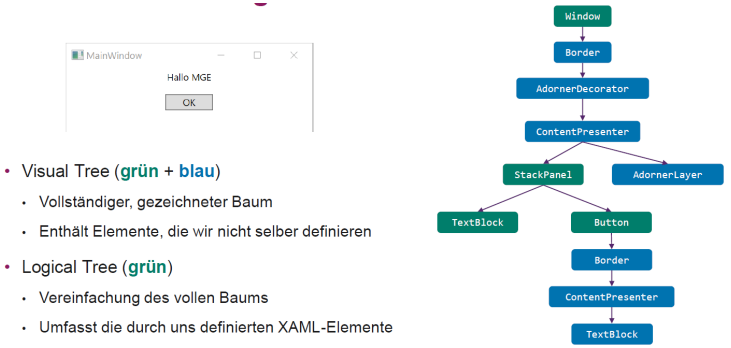
2.1 XAML Allgemein

Beschreibungssprache von Microsoft zur Gestaltung graphischer Oberflächen.



Für Design kann auch C# verwendet werden, XAML ist jedoch leichter, kürzer, lesbarer und hat einen Designer. Microsoft Blend für das Designen.

2.1.1 Visual Tree und Logical Tree



- Visual Tree (grün + blau)
  - Vollständiger, gezeichneter Baum
  - Enthält Elemente, die wir nicht selber definieren
- Logical Tree (grün)
  - Vereinfachung des vollen Baums
  - Umfasst die durch uns definierten XAML-Elemente

2.1.2 Namespaces

Mit xmlns werden XML-Namespaces definiert.
→ Ohne Doppelpunkt: Standard-Namespace (Elemente können ohne Präfix verwendet werden)
→ Mit Doppelpunkt: Nenannter Namespace (Elemente können nur mit Präfix verwendet werden)

Übliche Namespaces in WPF:

- Der Standard-Namespace wird auf die WPF Control Library gesetzt
- x für XAML-spezifische Elemente
- d für Elemente des visuellen Designers
- mc für Elemente der «Markup Kompatibilität»
- local für Elemente aus unserem eigenen Assembly

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/
    xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend
      /2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-
      compatibility/2006"
  xmlns:local="clr-namespace:Vorlesung_09"
  mc:Ignorable="d"
  ... />
```

2.1.3 Named Elements

Elemente können benannt werden. → Ermöglicht Zugriff auf Code-Behind. Attribut führt zu Property in generierter Klasse

```
// XAML:
<TextBlock Name="WpfAttribute" Text="WPF" />
<TextBlock x:Name="XamlAttribute" Text="XAML" />
// Code Behind:
this.WpfAttribute.Text = "...";
this.XamlAttribute.Text = "...";
```

2.1.4 Syntaxen

```
// Attribute Syntax:
<Button Background="Blue"
      Foreground="Red"
      Content="Mein Button" />
// Property Element Syntax:
<Button>
   <Button.Background>
      <SolidColorBrush Color="Blue"/>
   </Button.Background>
   <Button.Foreground>
      <SolidColorBrush Color="Red"/>
   </Button.Foreground>
   <Button.Content>
      Mein Button
   </Button.Content>
</Button>
```

2.1.5 Type Converters

```
// XAML:
<local:LocationControl Center="10, 20" />
// Control:
public class LocationControl : TextBlock {
   public Location Center {
      set => this.Text = $"{value.Lat} / {value.Long}";
   }
}
// Model:
[TypeConverter(typeof(LocationConverter))]
public class Location {
   public double Lat { get; set; }
   public double Long { get; set; }
}
// Type Converter:
public class LocationConverter : TypeConverter {
   public override object ConvertFrom(
     ITypeDescriptorContext context,
     CultureInfo culture,
     object value) {
       //Zur Kürzung des Beispiels auf Checks verzichtet:
       // - Ist value wirklich ein string?
       // - Enthält das Array exakt 2 Elemente?
       // - Sind die strings zu double konvertierbar?
       var valueAsString = (string) value;
       var valueArray = valueAsString.Split(',');
       return new Location {
          Lat = Convert.ToDouble(valueArray[0]),
          Long = Convert.ToDouble(valueArray[1])
       };
   }
}
```

2.1.6 Content Properties

Jedes XAML-Element kann genau eine Eigenschaften als seinen Inhalt definieren. Einige Elemente können, neben reinem Text, auch andere Elemente enthalten.

```
<Button Content="Label" />
<Button>Label</Button>
<Button Width="150" Height="60">
   <StackPanel>
      <TextBlock Text="Gross"
              TextAlignment="Center"
              FontSize="20" />
      <TextBlock Text="Und hier klein"
              FontSize="12"
              Foreground="#888888" />
   </StackPanel>
</Button>
```

2.1.7 Markup Extensions

Erlauben die Erweiterung des XAML-Markup mit zusätzlicher Logik. Die Logik wird in geschweiften Klammern platziert { ... }. Verwendet bei Styling und Data Binding.

```
// XAML:
<TextBlock Text="{local:LocationExtension Lat=10,Long
    =20}" />
// Marup Extension:
public class LocationExtension : MarkupExtension {
   public string Lat { get; set; }
   public string Long { get; set; }
   public override object ProvideValue(IServiceProvider
       s) {
      return this.Lat + " / " + this.Long;
   }
}
```

2.1.8 Attached Properties

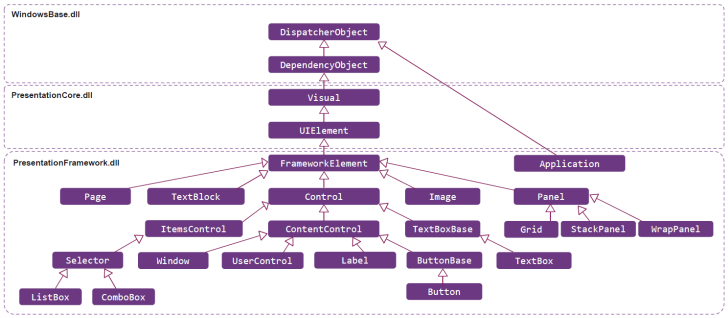
Setzt Eigenschaft auf einem Element, die zu einem anderen Element gehört. Die Eigenschaft wird sozusagen einem anderen Element angehängt.

```
<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>
```

2.2 Grundelemente

2.2.1 Klassenhierarchie

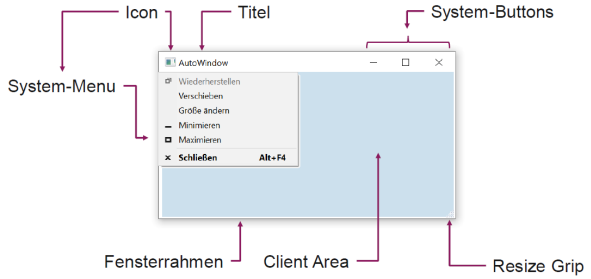


2.2.2 Application

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"
    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:

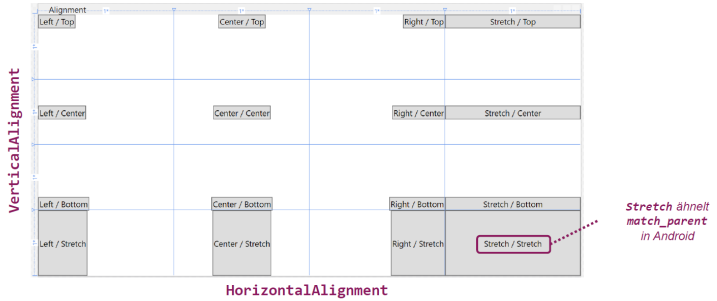
- Name-Property für Zugriff
- Logical Tree
- Layout System
- Visuelles Styling (Woche 10)
- Data Binding (Woche 11)

Grössenangaben:
Width, Height und Margin, Kein Padding. Zusätzlich MinWidth, MaxWidth und MinHeight, MaxHeight.

Dimensionen:
Auto: Automatische Grösse (wrap_content)
px: Device Independent Pixels, 1in == 96px

Ausrichtungen:

- `Alignment` beeinflusst die Ausrichtung **innerhalb des Eltern-Elements**



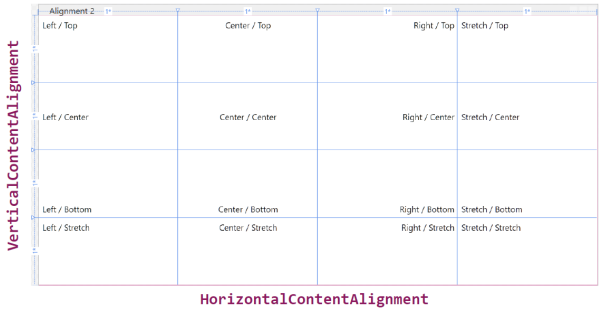
2.2.7 Control

Basis-Klasse für Controls mit Benutzerinteraktion.
Erweitert FrameworkElement 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, BorderBrush
Schriftbild: FontFamily, FontSize, FontStretch, FontStyle, FontWeight

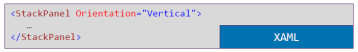
2.3 Layouts

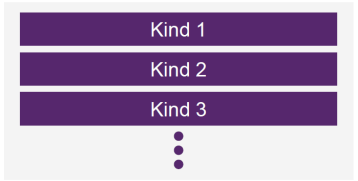
Layouts sind Container für Kind-Elemente. Haben eine Parent-Child Beziehung. Verschachtelung ist möglich.

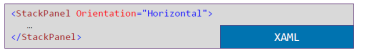
Verfügbare Layouts in WPF:

- 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

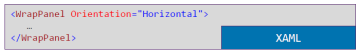
2.3.1 StackPanel

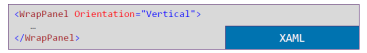


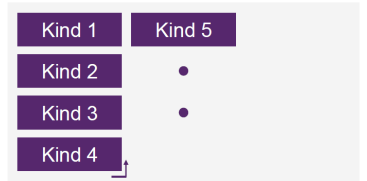




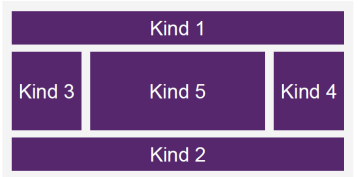
2.3.2 WrapPanel

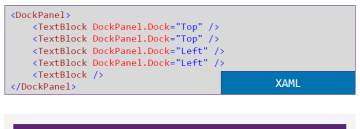


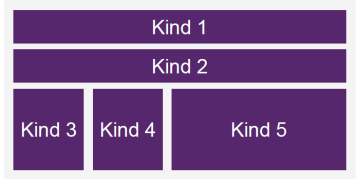


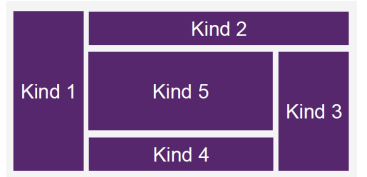


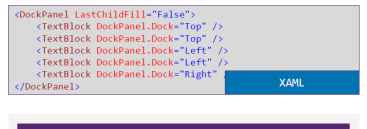
2.3.3 DockPanel

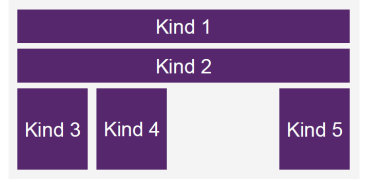




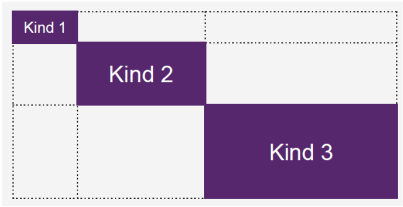








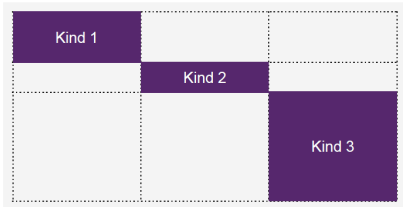
2.3.4 Grid




```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="1*" />
        <RowDefinition Height="2*" />
        <RowDefinition Height="3*" />
    </Grid.RowDefinitions>

    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="1*" />
        <ColumnDefinition Width="2*" />
        <ColumnDefinition Width="3*" />
    </Grid.ColumnDefinitions>

    <TextBlock Grid.Row="0" Grid.Column="0" />
    <TextBlock Grid.Row="1" Grid.Column="1" />
    <TextBlock Grid.Row="2" Grid.Column="2" />
</Grid>
```




```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="50" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>

    <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>

    <TextBlock Grid.Row="0" Grid.Column="0" />
    <TextBlock Grid.Row="1" Grid.Column="1" />
    <TextBlock Grid.Row="2" Grid.Column="2" />
</Grid>
```