Handout Einführung in WPF

von Luna, Leon und Damian

Erklärung und Einführung in WPF

WPF (Windows Presentation Foundation) ist wie WinForms eine Möglichkeit, eine Benutzeroberfläche zu designen. Im Gegensatz zu WinForms werden hier aber die Elemente nicht statisch per Drag and Drop platziert, sondern via Code und relativen Angaben. Dies sorgt dafür, dass sich bei Größenänderung des Fensters die Elemente auch formen und anpassen.

Typischerweise beginnt man den Code wie folgt: Es werden zwei Dateien benötigt - eine .cs-Datei, in der alle Funktionen implementiert werden, und die .xaml-Datei, in der das Layout der Oberfläche bestimmt wird. In Visual Studio werden beim Schreiben der .xaml-Datei oben im Designer auch die Änderungen live veranschaulicht.

Bei Projektstart wird automatisch die erste Codebasis für eine Oberfläche generiert. Dennoch möchten wir darauf eingehen.

Die .xaml-Datei beginnt mit dem Tag “Window”, um das Fenster zu definieren.

Dann folgen die nötigen Definitionen der XAML-Schemata.

Darauf folgt der Titel (Eigenschaft “Title”). Die Größe des Fensters wird mittels Height=““ und Width=““ festgelegt, was Höhe und Breite bestimmt.

Hiernach wird der Layout Manager angegeben, in dem das Layout festgelegt wird. Dieser wird von der folgenden Gruppe behandelt. Wir nehmen hier in allen Beispielen Grid.

<u>Der Code sollte nun wie folgt aussehen:</u>

```
<Window x:Class="WPF_Beispiel.MainWindow"
        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:WPF_Beispiel"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">
        <Grid>
        </Grid>
</Window>
```

Für die .cs-Datei gilt wie immer: oben alle benötigten Using-Anweisungen, danach der Namespace (in diesem Beispiel der Projektname). Die Menge der Using-Anweisungen wurde hier gekürzt. Bei vor-generiertem Code sind oft mehr Anweisungen vorhanden.

Hiernach folgt schon die Verknüpfung der Klasse MainWindow, die von Window erbt, in der wir alle Funktionen implementieren, die zum zugehörigen Fenster gehören.

Zur Syntax gibt es nicht besonders viel zu sagen. Ähnlich wie in HTML ist alles in in sich geschachtelte Blöcke integrierbar. Auch Ausrichtungen und Attribute werden in die Köpfe der Blöcke gesetzt oder sind im Block über den Blocknamen immer wieder konfigurierbar. In den folgenden Beispielen wird zu Blockbeginn die Liste der Attribute gesetzt und angesprochen, wäre aber auch im Block immer ansprechbar und setzbar. Ein Beispiel für fast jedes Kontrollelement wäre das Attribut Background. Dieses kann man im Block wie eben auch im Beispiel aufrufen und setzen. Sogar im Code der .cs-Datei ist dieses Attribut jederzeit über den Blocknamen aufrufbar.

Das Ausrichten ist auch immer in jedem Block zur darüberliegenden Ebene möglich.

Wenn ein Grid verwendet wird und im Grid der Margin angesprochen wird, bezieht sich der Margin auf das Grid. Wenn darin eine weitere Unterteilung erstellt wird und innerhalb davon der Margin angesprochen wird, bezieht sich der Margin auf den neuen inneren Kontext, nicht auf das außenliegende Grid.

Wie aus HTML bekannt, werden Blöcke immer mit einem Anfang und Ende begrenzt. Das Begrenzen selbst kann in derselben Zeile durch den Schlusstag /> am Ende passieren oder in Zeilen darunter durch den entsprechenden Tag.

Beispiel:

Öffnen und Schließen eines Grid-Blocks

Mehrzeiler:	Einzeiler:
<Grid>	<Grid />
</Grid>	

Man beachte, dass dies zwar Zeilen spart, aber ohne korrektes Einrücken auch schnell die Leserlichkeit beeinträchtigen kann. Bei einfachen Kontrollelementen kann es nützlicher sein, sie in eine Zeile zu schreiben, doch in anderen Fällen kann ohne Kommentare und Einrücken schnell Verwirrung herrschen. Hier ist Vorsicht geboten. In den folgenden Beispielen zeigen wir oft beide Möglichkeiten, um zu demonstrieren, dass beide Versionen funktionieren, aber am Ende muss man selbst entscheiden, welche Variante für sich und die eigene Anwendung die beste ist.

Jetzt wollen wir noch auf relative und absolute Bezüge und Größen eingehen. Wenn mittels Margin eine Größe eingestellt wird, ist diese relativ zum Parent-Kontext, also Parent-Größe - X in jede Richtung. Ebenso geht dies mit Align, was so viel wie einreihen oder anpassen bedeutet. Also align right = rechts anliegend. Diese Angaben sind bei allen dynamischen Fenstergrößen-Änderungen immer gleich, aber relativ zum betrachteten Referenzpunkt/-partner. Absolut wäre, wenn eine feste Größe angegeben wird, was ebenso möglich ist. Ändert sich nun die Größe des Fensters, so bleibt die Größe des Elements, dem eine absolute Größe zugewiesen ist, fest, auch wenn mehr Platz verfügbar wäre. Dasselbe gilt natürlich immer auch fürs Verkleinern des Fensters.

Der Code sollte nun wie folgt aussehen:

```
using System;

using System.Windows;



namespace WPF_Beispiel

{

        /// <summary>

        /// Interaktionslogik für MainWindow.xaml

        /// </summary>

        public partial class MainWindow : Window

        {

                public MainWindow()

                {

                        InitializeComponent();

                }

        }

}
```

Liste an Kontrollelementen

Button

Einfacher Knopf, wie man ihn kennt, nur dass dieser auch noch in Größe und Design durch seine Attribute angepasst werden kann

Code-Beispiel:

Definition in der .xaml-Datei

```
<Button Name="btn_Eins" Content="Klick mich" Foreground="Beige"

Width="100" Height ="100" Background="Blue" Click="click_Me_Function">

</Button>
```

Defintion in der .cs-Datei

```
void click_Me_Function(object sender, RoutedEventArgs e)

{

        MessageBox.Show("Hallo das ist eine Nachricht");

}
```

CheckBox

Wie bereits aus WinForms bekannt, ein einfaches CheckBox-Kästchen zum Abhaken

Code-Beispiel:

Defintion in der .xaml-Datei

```
<CheckBox Width="150" Height="20" Content="Ankreuzen" Background="Aqua"
Foreground="BlueViolet">

</CheckBox>
```

Oder:

```
<CheckBox Width="150" Height="20" Background="Aqua" Foreground="BlueViolet">

        <!--Hier steht der Text der Checkbox-->

        Ankreuzen

</CheckBox>
```

ComboBox

Ein Dropdown-Menü mit darin enthaltenen Einträgen, von denen einer ausgewählt werden kann

Code-Beispiel:

Defintion in der .xaml-Datei

```
<ComboBox Width="150" Height="25">

        <ComboBoxItem> Hallo </ComboBoxItem>

        <ComboBoxItem> Welt </ComboBoxItem>

        <ComboBoxItem> das </ComboBoxItem>

        <ComboBoxItem> sind </ComboBoxItem>

        <ComboBoxItem> Einträge </ComboBoxItem>

</ComboBox>
```

Label

Einfaches bekanntes Label, um Text darzustellen, wie aus WinForms bekannt

Code-Beispiel:

Defintion in der .xaml-Datei

```
<Label Width="50" Height="30" Content="Hallo">

</Label>
```

Oder:

```
<Label Width="50" Height="30">

        Hallo

</Label>
```

ListBox

Klassische ListBox ähnlich einer Textbox, die eine einfache Auflistung enthält

Code-Beispiel:

Defintion in der .xaml-Datei

```
<ListBox Width="150" Height="150" Background="Blue" Foreground="White">

        <ListBoxItem>Hallo</ListBoxItem>

        <ListBoxItem>Welt</ListBoxItem>

        <ListBoxItem>das</ListBoxItem>

        <ListBoxItem>ist</ListBoxItem>

        <ListBoxItem>ein</ListBoxItem>

        <ListBoxItem>ListBox Item</ListBoxItem>

</ListBox>
```

Menu

Stellt, wie aus WinForms bekannt, eine Menüleiste dar, die ihr zugeteilte Menüpunkte enthält und bei Auswahl dieser das Event des Klicks auch an die zugeteilten Funktionen weitergibt

Code-Beispiel:

Defintion in der .xaml-Datei

```
<Menu DockPanel.Dock="Top">
        <!--Menüpunkt-->
        <MenuItem Header="_Datei">
                <!--Untermenüpunkte im Menüpunkt Datei-->
                <MenuItem Header="_Neu" Click="click_Menu_New"></MenuItem>
                <MenuItem Header="_Öffen" Click="click_Menu_Open"></MenuItem>
                <MenuItem Header="_Speichern" Click="click_Menu_Save"></MenuItem>
                <!--Trennlinie-->
                <Separator></Separator>
                <MenuItem Header="Beenden" Click="click_Close"></MenuItem>
                <!--Ende Menüpunkt Datei-->
                </MenuItem>
        <!--Nächte Menüpunkte-->
        <MenuItem Header="_Bearbeiten"></MenuItem>
        <!--Nächte Menüpunkte-->
        <MenuItem Header="_Ansicht"></MenuItem>
        <!--Nächte Menüpunkte-->
        <MenuItem Header="_Einfügen"></MenuItem>
        <!--Nächte Menüpunkte-->
        <MenuItem Header="_Hilfe"></MenuItem>
</Menu>
```

Image

Image ist, genau wonach es klingt, ein einfaches eingebundenes Bild. Hier kann mittels Höhe, Breite, Quelle(Source) und anderen Attributen die Darstellung eines Bildes in einem WPF-Fenster realisiert werden.

Code-Beispiel:

Defintion in der .xaml-Datei

```
<Grid Background="Beige">

        <Image Name="img_One" Width="300" Height="150" Source="F:\BBSI LF Aufgaben\Lehrjahr 3\LF 10 Jenet\Workshop WPF\Beispiele\WPF-Beispiel\bin\Debug\BspBild.png"/>

</Grid>
```

RadioButton

Wie in vielen Anwendungen ist auch hier der RadioButton der CheckBox ähnlich, es kann aber nur ein einziger RadioButton gleichzeitig im selben Kontext aktiviert sein. Daher wird ein RadioButton deaktiviert, wenn ein anderer aktiviert wird.

Code-Beispiel:

Defintion in der .xaml-Datei

```
<RadioButton Width="100" Height="30" Content="Radiobutton 1" Name="rb_One">

</RadioButton>
```

Oder:

```
<RadioButton Width="100" Height="30" Content="Radiobutton 1" Name="rb_One" />
```

Slider

Ein einfacher Schieberegler, der einen ihm vorgebenen Wertebereich von links nach rechts in einer ihm übergebenen Schrittweite durchläuft

Code-Beispiel:

Defintion in der .xaml-Datei

```
<Slider Width="150" Height="30" Name="slider_One" Minimum="0" Maximum="100" Value="0" Interval="1"
ValueChanged="update_Value" TickPlacement="TopLeft" TickFrequency="5"  IsSnapToTickEnabled="True">

</Slider>
```

Defintion in der .cs-Datei

```
void update_Value(object sender, RoutedEventArgs e)

{

        double Value_Slider = slider_One.Value;

}
```

TextBox

Ein ganz einfaches Textfeld, in dem Text eingetragen werden kann. Die Größe und Darstellung ist durch die Attribute anpassbar und selbst der darin eingestellte Text kann angepasst und ausgelesen werden.

Das Attribut TextWrapping ermöglicht, dass der Text am rechten Rand der Box umgebrochen wird. Das Attribut AcceptsReturn macht das Benutzen der Entertaste zum Zeilenumbruch möglich.

Code-Beispiel:

Defintion in der .xaml-Datei

```
<TextBox Width="400" Height="300" Name="tb_Textbox_one" Background="Beige"          TextWrapping="Wrap"
AcceptsReturn="True">

</TextBox>
```

Oder:

```
<TextBox Width="400" Height="300" Name="tb_Textbox_one" Background="Beige"          TextWrapping="Wrap"
AcceptsReturn="True" />
```

Liste an typischen Attributen

Attributname	Beschreibung
Name=“ “	Gibt Kontrollelementen einen Namen, um im Code das Element ansteuern zu können
Width=“ “	Legt die Breite des Element fest
Height=“ “	Legt die Höhe des Element fest
Background=“ “	Bestimmt die Hintergrundfarbe des Elements
Foreground=“ “	Legt die Schriftfarbe Fest
Content=“ “	Legt bei Elementen den Text fest
Value=“ “	Setzt den Wert bei Elementen, die Werte besitzen
Click=“ “	Legt die OnClick-Methode fest
ValueChanged=“ “	Legt die Methode fest, die bei Wertänderungen ausgeführt wird
Minimum=“ “	Setzt bei Wertebereichen von Elementen den Minimum-Wert
Maximum=“ “	Setzt bei Wertebereichen von Elementen den Maximum-Wert