Markus Krugel, Pieter Vorreiter 06.02.2018

Nachschlagewerk zu Xamarin.Forms



Inhaltsverzeichnis

[**Aufbau der Content Pages für die MTMDE App von SPF**](#_vabr4wvxdi1o) **3**

[**Xamarin UI Elements**](#_38hy755gahai) **4**

[Für alle geltend (View Class)](#_h4ogberkfg0n) 4

[ActivityIndicator](#_wma2gp9jhf76) 8

[BoxView](#_23pm3blq5qyh) 10

[Buttons](#_k0j9ep3mtije) 12

[DatePicker](#_4m8p4i986h7t) 15

[Editors (mehrzeilige Texte)](#_pgh0kf9wdhwi) 17

[Entries (einzeilige Texte)](#_afp8qzjxv4lv) 23

[Images](#_4b5qfaggonav) 29

[Labels](#_eh9lmx1p4xcp) 32

[Listview (für lange Listen)](#_aub53o9ee68g) 35

[Picker](#_qs10bgub8b75) 40

[ProgressBar](#_417a7zntaoeq) 42

[SearchBar](#_hiqjn6x999bw) 46

[Stepper](#_t7a7zxr45eg) 49

[Switch](#_i9fnz6cjqwkd) 52

[TableView (für Listen mit Eingabefeldern)](#_ncbzc2vnlds4) 54

[TimePicker](#_yjsrc4nybq41) 58

[**Xamarin Layouts**](#_6e94uxm21slo) **60**

[AbsoluteLayout](#_gwple0tq7dfg) 61

[Grid](#_pqy01j68vm7x) 66

[RelativeLayout](#_yu5c64h3c6os) 69

[Stacklayout](#_wy7ckkcj4l9x) 72

[ScrollView als Ergänzung](#_1o72g0mig5s3) 75

[**Sonstiges zu Xamarin**](#_xozmwleesm7f) **79**

[Data Binding](#_cici3smt7h1n) 79

[Model-View-ViewModel (MVVM)](#_q8cllyqo1u7u) 85

[Styles](#_xc2fyji9y3o0) 93

[Trigger](#_cnl8qipijngp) 100

[**Notizen:**](#_pfp4ttvce92u) **109**

# Aufbau der Content Pages für die MTMDE App von SPF

Wir haben uns für den Aufbau der Contentpages für ein Verschachteltes Layout entschieden.

Alle Views werden dabei zuerst in ein Relativelayout geschoben, was den Vorteil hat, dass wir sie mit den Relativelayout exklusiven Constraints genau an die ParentViews oder Andere Views innerhalb des Layouts anpassen können unabhängig davon ob sie nicht direkt im Relativelayout drinn sind oder vorher noch Children von einem anderen Layout sind.

Davon abgeleitet sind sowohl der Bereich für die uns zur Verfügung gestellten Buttons als auch der TableView in das unsere CustomCells rein kommen.

Die Trennung von den Buttons und dem Content hat den Vorteil, dass die Buttons nicht mit den Restlichen Views im TableView mitscrollen und immer am unteren Teil der App zu sehen sind.

Wie bereits erwähnt ist das TableView dafür zuständig, dass wir unabhängig von der Masse des Contents immer gewährleisten können, dass alles durch das scrollen angezeigt werden kann (erreichbar ist). Alternativ könnte man hier ein beliebiges anderes Layout benutzen und dieses in einen ScrollView packen.

Innerhalb der Zellen ist momentan ein Grid vorgesehen damit man die Views, die sich innerhalb der Zellen befinden ohne Probleme und mit einheitlichen Spalten anzeigen kann. Damit aber die Höhe der Zellen nicht limitiert ist und so ein Teil des in den Zellen verwendeten grids abgeschnitten ist bietet es sich an durch die Property HasUnevenRows = true

dies auszuschalten.

***Der Content soll vom Layout durch das MVVM-Model gekapselt sein, was eine Bearbeitung des Models und des ViewModels unabhängig voneinander ermöglicht.***

# 

# Xamarin UI Elements

## 

## Für alle geltend (View Class)

**Attribute:**

AnchorX: Setzt die X-Koordinate des Ursprungspunktes von dem aus das

Element gezeichnet wird.

AnchorY: Setzt die Y-Koordinate des Ursprungspunktes von dem aus das

Element gezeichnet wird.

BackgroundColor: Verändert die Hintergrundfarbe des Elementes dem es

zugeordnet ist.

BindingContext: Das Objekt, mit dem man ein Binding hat.

Bounds (readonly): Gibt den Rahmen des Elements als Rechteck wieder.

ClearValue(BindableProperty property): Entfernt die Werte der

angegebenen Property

*Resetet die Property auf den standardWert*

FindByName<T> (string name): Sucht in den untergeordneten Elementen

nach dem Element des Typs T mit dem

vorgegebenen Namen

Focus: Versucht das Element zu fokussieren

GetValue(BindableProperty property): Gibt den Wert der angegebenen

Property wieder

Height(readonly) : Die Höhe des Elementes

HeightRequest: Hiermit verändert man die Höhe des Elementes

HorizontalOptions:

“Center”: Zentriert die View innerhalb des Layouts

“End”: Platziert die View am rechten Ende des Layouts

“Fill”: Platziert die View so, dass es kein Padding gibt

“Start”: Platziert die View am linken Anfang des Layouts

“CenterAndExpand”: Zentriert die View innerhalb des Layouts und

nimmt dann soviel Platz ein, wie das Layout

erlaubt

“EndAndExpand”: Platziert die View am rechten Ende des Layouts

und breitet sich von dort aus

“FillAndExpand”: Platziert die View so, dass es kein Padding gibt

und breitet sich dann auf das gesamte Layout

aus

“StartAndExpand”: Platzier die View am linken Anfang des Layouts

und breitet sich von dort aus

IsEnabled: Hiermit schaltet man die Interaktionsmöglichkeit des Elementes

mit den Benutzer an und aus

IsFocused (readonly): Gibt wieder ob der Benutzer das Element ausgewählt hat

IsVisible: Die Sichtbarkeit des Elementes

Layout (Rectangle bounds): Passt den Rahmen des Elementes an die

mitgegebene Rectangle an

Opacity: Die Durchsichtigkeit des Elementes

Parent: Das Eltern - Element

RemoveBinding (BindableProperty property): Entfernt das Binding von Element

RemoveDynamicResource (BindableProperty property): Entfernt den

dynamischen Style

Resources: Das Resource-Dictionary des Elementes

Rotation: Die Rotation um die Z-Achse

RotationX: Die Rotation um die X-Achse

RotationY: Die Rotation um die Y-Achse

Scale: Der Skalierungsfaktor des Elements

SetBinding (BindableProperty targetProperty, BindingBase binding):

Fügt das Binding für die Zielproperty des Elementes hinzu

SetDynamicResource (BindableProperty property, String key):

Setzt für die Property einen dynamischen Style ein

SetValue (BindableProperty property, object value): Setzt den Wert der

Property

Style: Bestimmt das Aussehen des Elementes

Triggers: Eine Liste der Triggers des Elementes

Width (readonly): Breite des Elementes

WidthRequest: Hiermit verändert man die Länge des Elementes

VerticalOptions:

“Center”: Zentriert die View innerhalb des Layouts

“End”: Platziert die View am unteren Ende des Layouts

“Fill”: Platziert die View so, dass es kein Padding gibt

“Start”: Platziert die View am oberen Anfang des Layouts

“CenterAndExpand”: Zentriert die View innerhalb des Layouts und

nimmt dann soviel Platz ein, wie das Layout

erlaubt

“EndAndExpand”: Platziert die View am unteren Ende des Layouts

und breitet sich von dort aus

“FillAndExpand”: Platziert die View so, dass es kein Padding gibt

und breitet sich dann auf das gesamte Layout

aus

“StartAndExpand”: Platzier die View am oberen Anfang des Layouts

und breitet sich von dort aus

**Methoden/Events**

BindingContextChanged: wird aufgerufen, wenn sich der BindingContext

ändert

ChildAdded: wird aufgerufen, wenn dem Element ein weiteres Element

untergeordnet wird

ChildRemoved: wird aufgerufen, wenn ein untergeordnetes Element

entfernt wird

ChildrenReordered wird aufgerufen, wenn die Reihenfolge der

untergeordneten Elemente sich verändert hat

Focus: wird aufgerufen, wenn das Element im Fokus gerät

PropertyChanged: wird aufgerufen, wenn sich ein Property verändert hat

PropertyChanging: wird aufgerufen, wenn eine Property dabei ist sich zu

verändern

SizeChanged: wird aufgerufen, wenn sich die Höhe oder Länge des Elementes

verändert

UnFocused: wird aufgerufen, wenn das Element Fokus verliert

## 

## ActivityIndicator

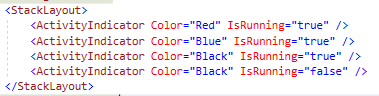
**Attribute**

Color: Die Farbe des ActivityIndicators

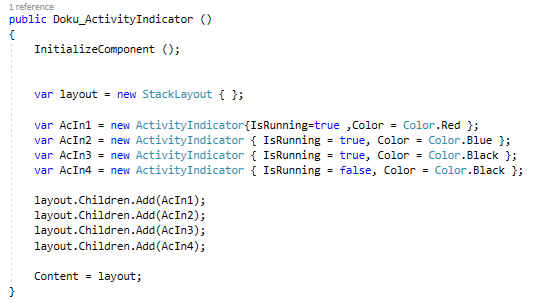
IsRunning: Der Wert, ob der ActivityIndicators aktiv ist oder nicht.

**Beispiele**

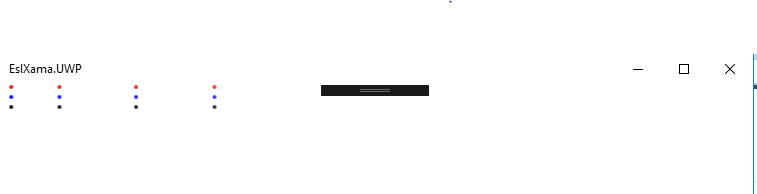
**XAML:**

****

**C#:**

****

**Ausgabe:**



## 

## BoxView

Bei der Boxview handelt es sich um ein farbig gezeichnetes Rechteck.

Es wird oft für einfache Visualisierungen verwendet.

**Attribute**

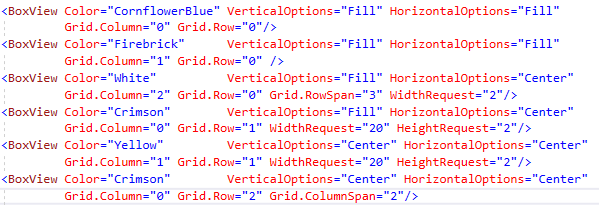
Color: Die Farbe der Viewbox

**Events/Methoden**

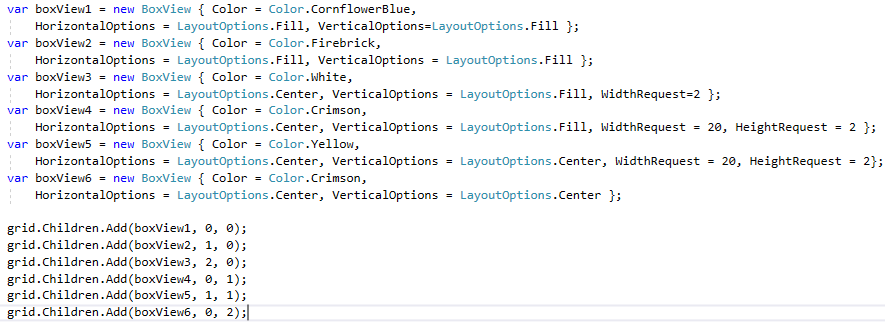
**Keine exklusiven Methoden oder Events**

**Beispiele**

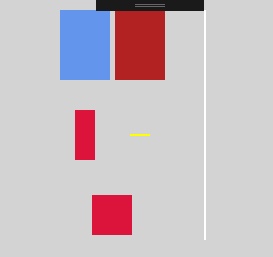
**XAML:**

****

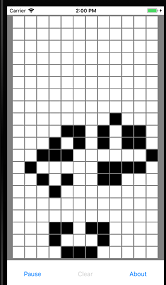
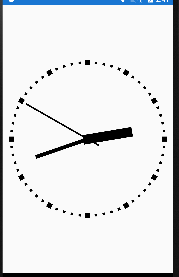
**C#:**

****

**Ausgabe:**

****

**stärkere Beispiele für Verwendung von BoxViews:**

****

## 

## Buttons

**Attribute**

BorderColor Die Farbe des Randes kann gesetzt und abgerufen

werden.

BorderRadius/ Der Radius der Randecken

CornerRadius

BorderWidth Die Breite des Randes

Font Die Schriftart des sich im Button befindlichen Textes.

FontAttributes Der Schrifttyp des Buttontextes (fett, italic,nichts)

FontSize Die Größe der Schrift innerhalb des Textes.

FontFamily Die Schriftfamilie des sich im Button befindlichen

Textes.

Image Das Bild das optional noch vom Button angezeigt

werden soll.(Dabei können keine Bilder, die als

ressourcen verwendet werden ,angezeigt werden.)

Text Der Text der in dem Button steht

TextColor Die Farbe in der der Text des Buttons angezeigt wird.

**Methoden/Events**

Clicked Event wird durch einen Button klick ausgelöst.

Pressed Das Event wird durch das runterdrücken des Buttons

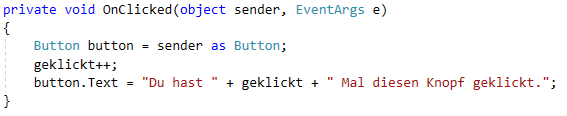
ausgelöst.

Released Das Event wird durch das loslassen des herunter gedrückten

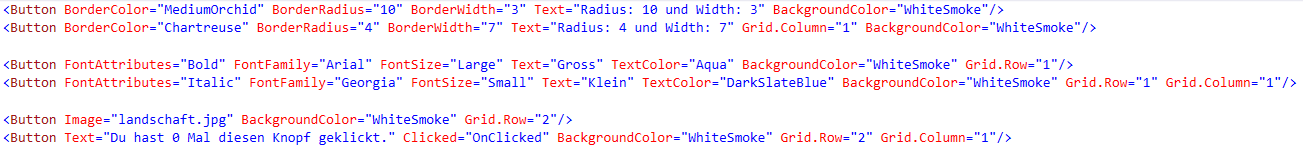
Buttons ausgelöst.

**Beispiel:**

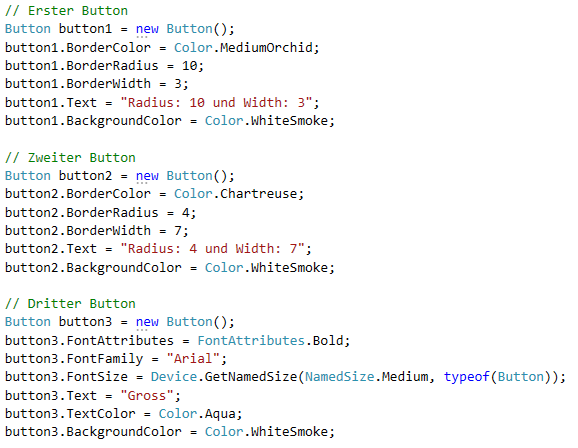
**OnClicked - Methode:**

****

**XAML:**

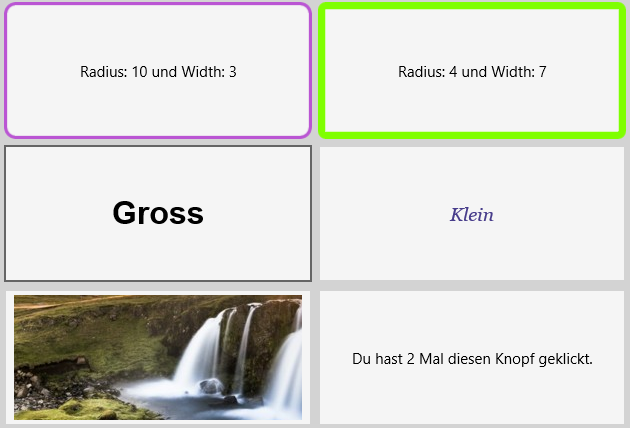
****

**C#:**

****

****

**Ausgabe:**

****

## DatePicker

**Attribute**

Format: Das Anzeigeformat des Datums, z.B. “dd-MM-yyyy”

Date: Das ausgewählte Datum

MinimumDate: Das erste Datum, welches man auswählen darf

MaximumDate: Das letzte Datum, welches man auswählen darf

**Methoden/Events**

DateSelected: Wrd aufgerufen, wenn ein Datum gewählt wurde

**Beispiel:**

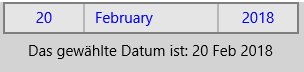
**XAML**

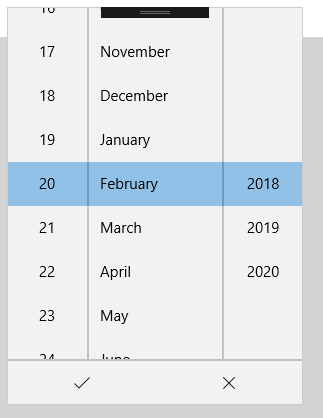
****

**C#**

****

**Ausgabe**

****

****

## 

## Editors (mehrzeilige Texte)

***Im Gegensatz zu Entries* kann für Editoren weder die Eigenschaft *HorizontalTextAlignment* noch *Placeholder* oder *isPassword* setzen.**

**Attribute**

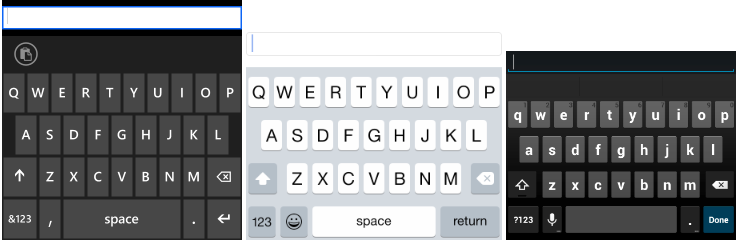
Keyboard:

Der Editor unterstützt über die Eigenschaft ‘Keyboard’ eine Reihe von

Tastaturlayouts, die erst bei den **mobilen Geräten** zum Einsatz

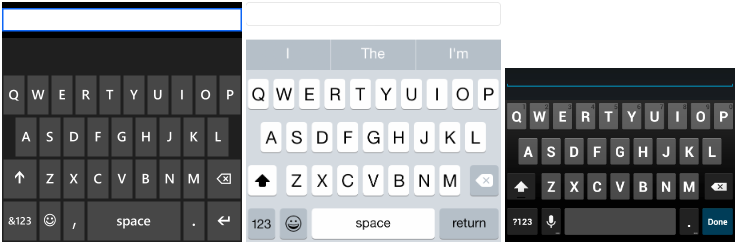
kommen.

“Default” : Die Standardtastatur.



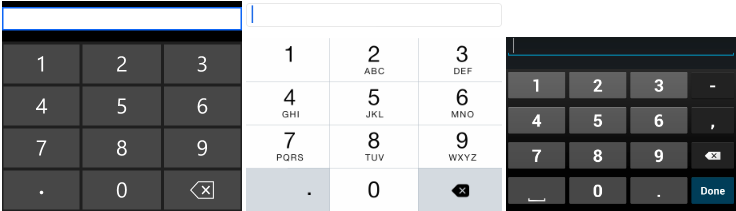
“Chat”: Eine Tastatur, die eingesetzt wird für wenn auch Emojis

benutzt werden sollen.



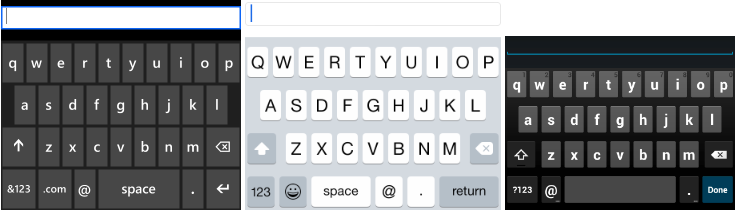
“Numeric” : Die Tastatur, die für reine Zahlen eingaben verwendet

werden soll.



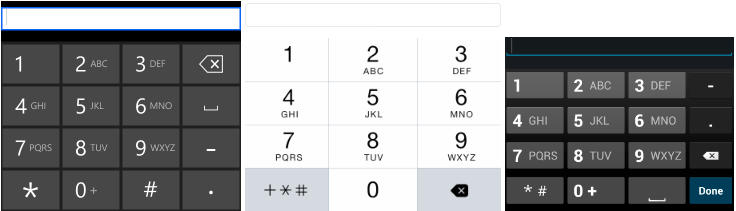
“E-Mail” : Die Tastatur die für die Eingabe von E-Mails benutzt

werden sollen.



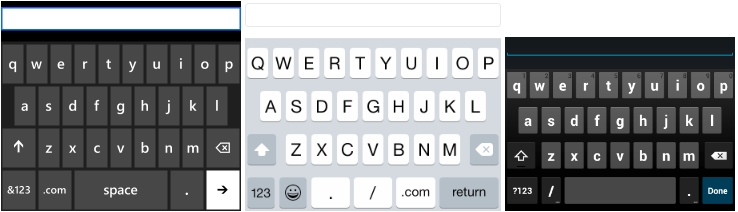
“Telefone”: Die Tastatur die für das Eingeben von Telefonnummern

verwendet wird.



“Url”: Die Tastatur die für Pfade von Webseiten und

Dokumenten verwendet wird.



Methoden/Events

TextChanged: Das Event wird immer dann ausgelöst, wenn der Text innerhalb

des Editors sich ändert.

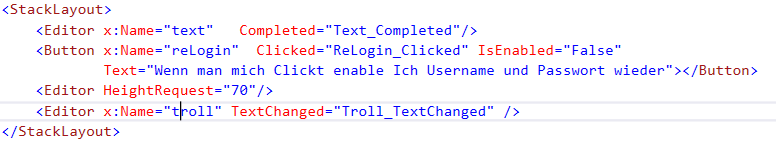
Innerhalb der Methode kann man mithilfe der mitgegebenen ‘**TextChangedEventArgs**’ über die Eigenschaft ‘**OldTextValue**’ auf den alten Wert und über die Eigenschaft ‘**NewTextValue**’ auf den neun Wert zugreifen.

Completed: Das Completed Event wird erst dann ausgelöst, wenn der

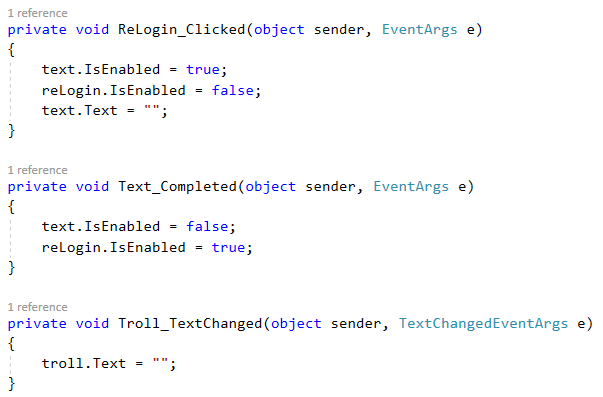
Benutzer seine Eingabe mit der ‘**Enter**’ Taste bestätigt.

**Beispiel**

**XAML**



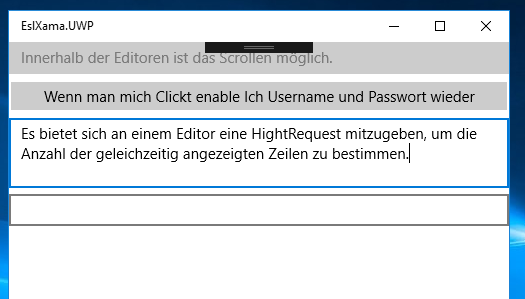
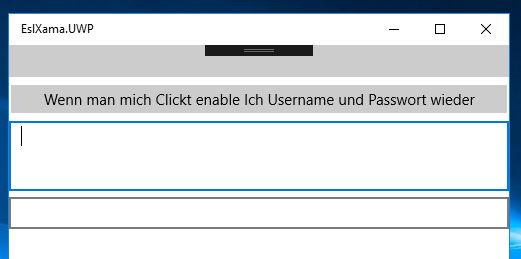
**Eventmethoden**

****

C#:

****

**Ausgabe:**



## 

## Entries (einzeilige Texte)

**Attribute**

***Im Gegensatz zu Editoren* kann man für Entries die Eigenschaft *HorizontalTextAlignment* , *Placeholder* und *isPassword* setzen.**

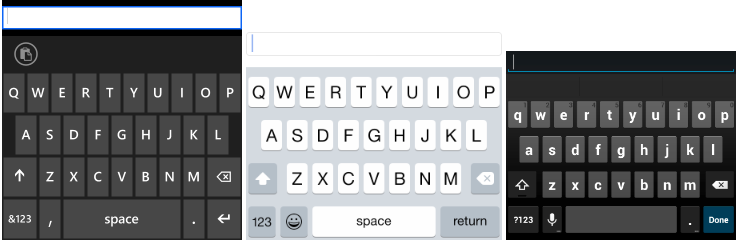
Keyboard:

Das Entry unterstützt über die Eigenschaft ‘Keyboard’ eine Reihe von

Tastaturlayouts, die erst bei den **mobilen Geräten** zum Einsatz

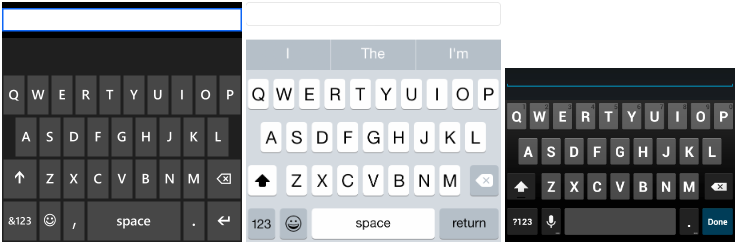
kommen.

“Default” : Die Standardtastatur.



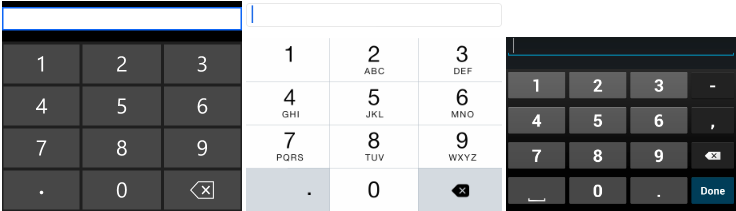
“Chat”: Eine Tastatur, die eingesetzt wird für wenn auch Emojis

benutzt werden sollen.



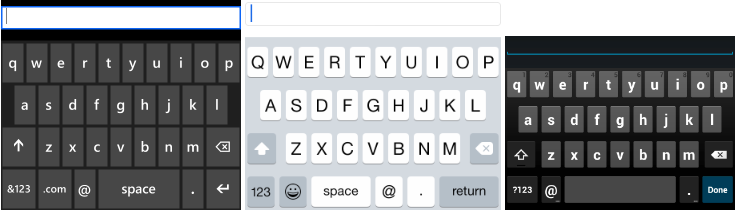
“Numeric” : Die Tastatur, die für reine Zahlen eingaben verwendet

werden soll.



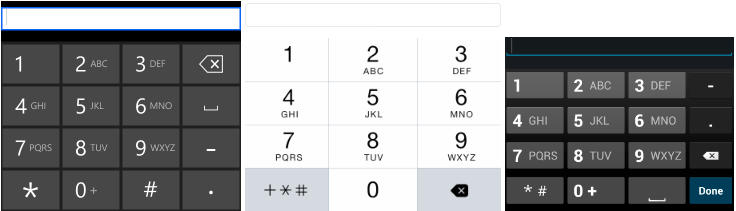
“E-Mail” : Die Tastatur die für die Eingabe von E-Mails benutzt

werden sollen.



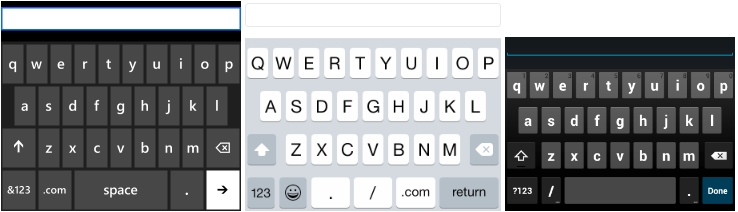
“Telefone”: Die Tastatur die für das Eingeben von Telefonnummern

verwendet wird.



“Url”: Die Tastatur die für Pfade von Webseiten und

Dokumenten verwendet wird.



Placeholder: Ein String der in dem Entry gezeigt wird, solange der User

keine Eingabe gemacht hat.

isPassword: Sollte dieses Attribut “true” sein, werden die Eingaben in

diesem Entry als Punkte dargestellt, sodass das eingegebene

Zeichen nicht zu erkennen ist.

**Methoden/Events**

TextChanged: Das Event wird immer dann ausgelöst, wenn der Text innerhalb

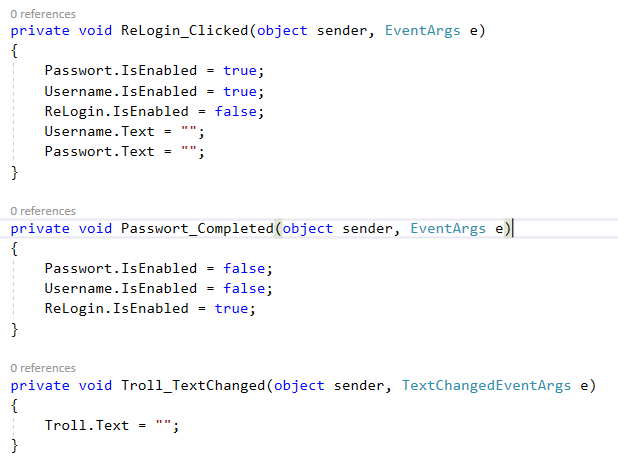
des Entries sich ändert. Innerhalb der Methode kann man mithilfe der mitgegebenen ‘**TextChangedEventArgs**’ über die Eigenschaft ‘**OldTextValue**’ auf den alten Wert und über die Eigenschaft ‘**NewTextValue**’ auf den neun Wert zugreifen.

Completed: Das Completed Event wird erst dann ausgelöst, wenn der

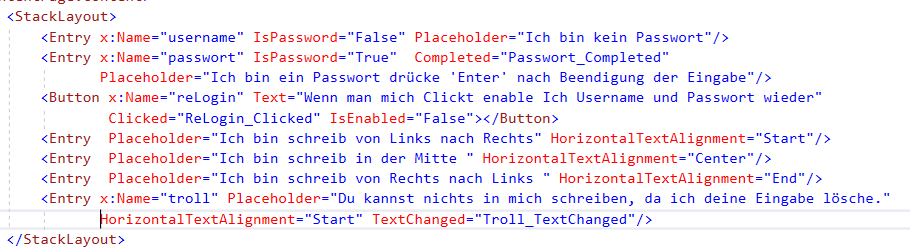
Benutzer seine Eingabe mit der ‘**Enter**’ Taste bestätigt.

**Beispiel**

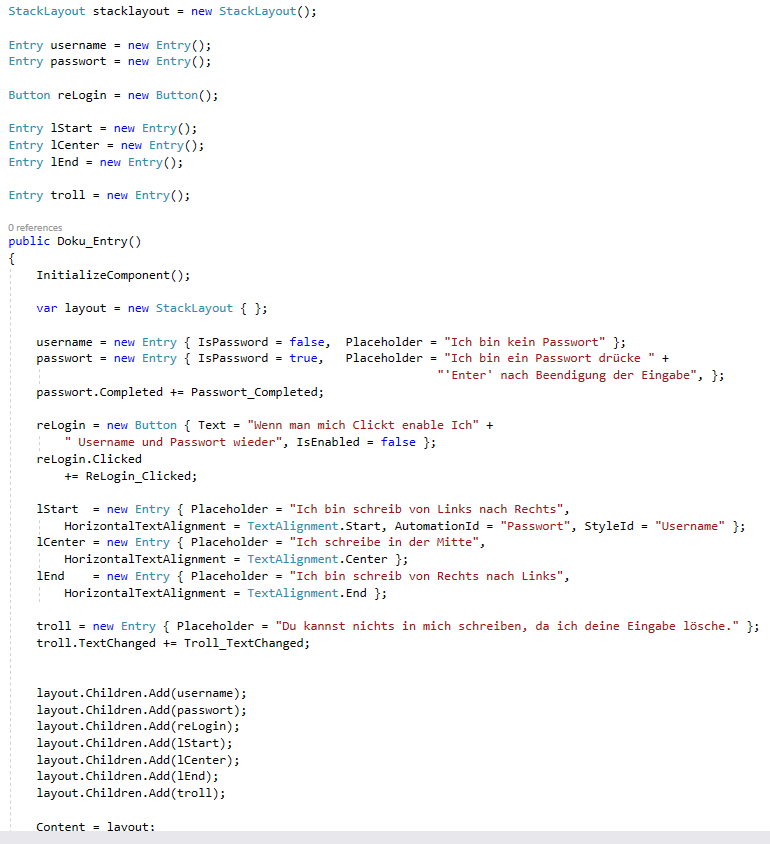
**Eventmethoden**

****

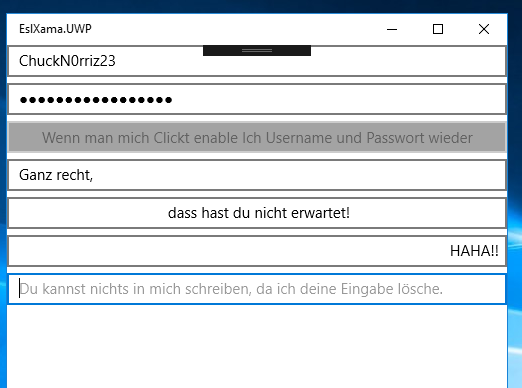
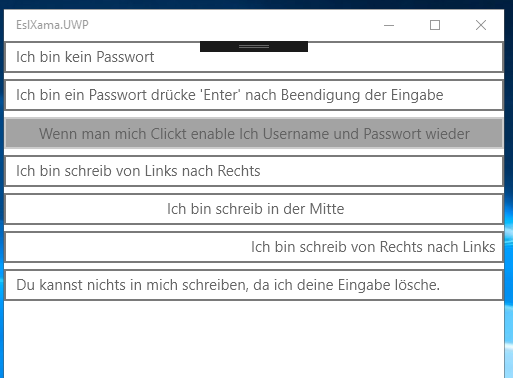
**XAML**

****

**C#**

****

**Ausgaben**

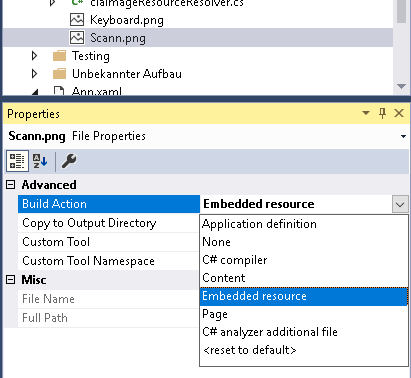


## 

## Images

Zu Anfang des Projektes sollte **.Net Standard** als **Code Sharing Strategy** eingestellt werden da wir von der Plattformunabhängig programmieren wollen.

**Wichtig!** Damit die Images auch übergreifend für alle Plattformen verwendet werden können muss man sie noch von ihrem Standartwert “Content” einzeln zu “**Embedded resource**” über ihre Properties festlegen.

****.

Nachdem man das Bild von Content nach Embedded Resource umgestellt hat, muss der Pfad mit einem Punkt zwischen jedem Wert geschrieben werden und einer Methode ImageSource.FromResource() mitgegeben werden damit der Source der Richtige Pfad zum Laden der Ressource übergeben wird.

**Source = ImageSource.FromResource(“EslXama.Images.Scann.png”)**

**Attribute**

ImageSource: Die Property der der Pfad des Bildes mitgegeben wird

und mit deren hilfe das Bild richtig geladen werden

sollte.

Aspect

Die Aspect Eigenschaft ist für die Skalierung des Bildes innerhalb der zur verfügung gestellten Fläche zuständig.

Fill: Streckt das Bild um die Fläche auszufüllen, wodurch es

zu verzerrungen kommen kann

AspectFill: Das Bild wird so in der Fläche skaliert, dass es ohne

Verzerrungen den Gesamten zur verfügung stehenden

Platz füllt.

AspectFit: Skaliert das Bild, so dass es in der Fläche passt, wobei

es nicht den Gesamten zur verfügung Platz verwenden muss und leere Flächen an den Rändern vorkommen können.

IsLoading (readonly): Die Eigenschaft IsLoading gibt als Boolean Wert wieder

ob ein Bild geladen wurde oder nicht.

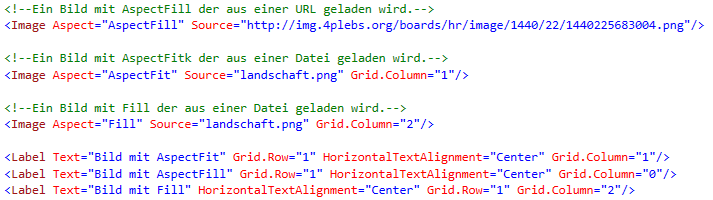
**Methoden/Events**

SetIsLoading(Boolean): Setzt den Wert von IsLoading auf einen

mitgegebenen Wert.

**Beispiel**

**XAML:**

****

**C#:**

****

**Ausgabe:**

****

## 

## Labels

**Attribute**

LineBreakMode:

Diese Eigenschaft ist für das Verhalten des Textes zuständig falls in einer Zeile zu wenig Platz ist für die gesamte Länge des Textes.

“HeadTruncation”: Anfang des Textes wird gekürzt, das Ende wird

vollständig gezeigt

“CharacterWrap”: Bei einem Zeichenlimit findet ein

Zeilenumbruch statt

“MiddleTruncation”: Zeigt den Anfang und das Ende des Textes

ungekürzt, jedoch wird der mittlere Teil gekürzt.

“NoWrap”: Zeigt nur das an, was in einer Zeile passt

“TailTruncation”: Zeigt den Anfang des Textes , das Ende wird

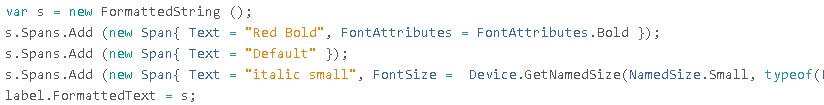
gekürzt.

“WordWrap”: Zeilenumbrüche passen sich an die Wörter an

FormattedText: Die Eigenschaft FormattedText ist vom Typ

FormatedString.

Ein FormatedString kann aus mehreren Spans bestehen, um somit innerhalb eines Textes unterschiedliche Fontattribute verwenden zu können.



**Methoden/Events**

Das Label hat keine exklusiven Methoden oder Events.

**Beispiel**

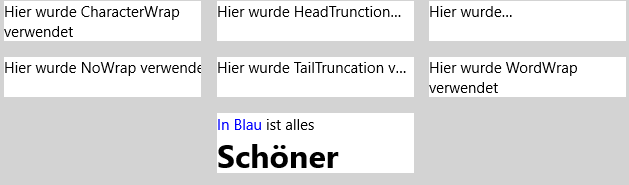
**XAML:**

****

**C#:**

****

**Ausgabe:**

****

## 

## Listview (für lange Listen)

Im Gegensatz zu den TabelVies sollen die Levies mir einheitlichen zelle arbeiten.

ItemsSource: befüllt die Liste mit den angegebenen Werten

Header: hier bestimmt man das Aussehen der Kopfzeile der Listview

(muss in einem Layout verpackt werden)

Footer: hier bestimmt man das Aussehen der Fußzeile der Listview

Groups: man kann die Items in Gruppen aufteilen für eine bessere

Übersicht

Cells: Eine Reihe in der Listview. Gibt es in folgende Varianten:

“TextCell” - Eine Zeile Text mit einer optionalen kleineren zweiten

Zeile.

“ImageCell” - Wie TextCell, jedoch kann man auf der linken Seite ein

Bild darstellen.

“Custom Cell”- In einer CustomCell kann man alle Werte selbst

bestimmen.

isPullToRefreshEnabled bei True kann der Benutzer mit runterziehen die

Listview aktualisieren

SelectedItem Das Item, dass momentan in der Liste ausgewählt ist.

RowHeight Die Höhe der einzelnen Zellen lassen sich darüber

zuweisen.

HasUnevenRows Bei True berechnet Xamarin.Forms für jede Reihe die

optimale Höhe selbst.

SeparatorVisibility: Mit der Eigenschaft lassen sich die Linien die die Zellen

trennen ausblenden.

“Default”- Es werden SeperatorLinien angezeigt

“None”- Es Werden keine SeperatorLinien angezeigt.

**Methoden/Events**

ItemAppearing wird aufgerufen, wenn ein neues Item in der Listview

erscheint

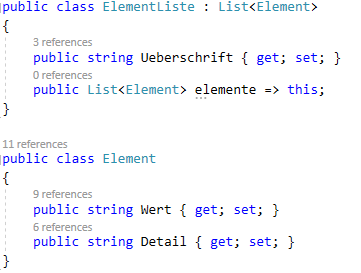
ItemDisappearing wird aufgerufen, wenn ein Item aus der Listview

verschwindet

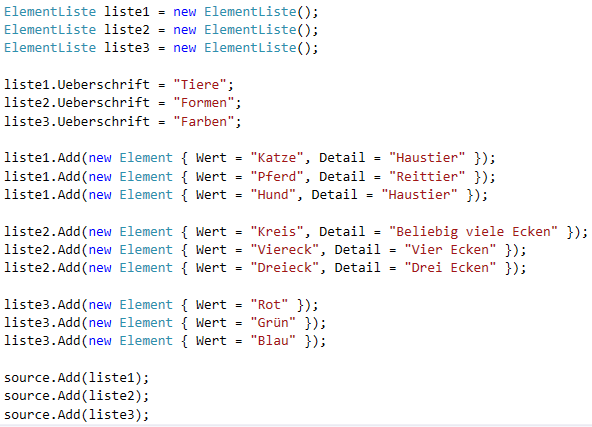
ItemSelected wird aufgerufen, wenn ein Item ausgewählt wird

**Beispiel**

**Hilfsklassen:**

****

**ListeFuellen - Methode:**

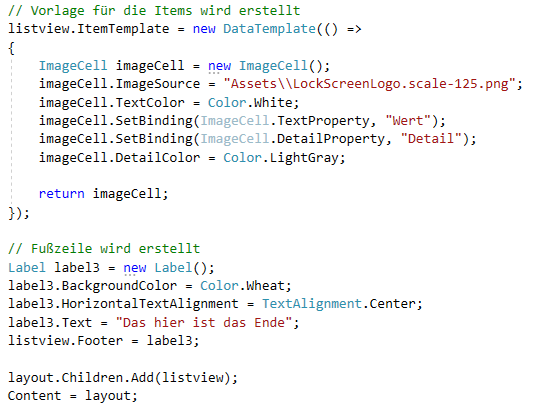
****

**XAML:**

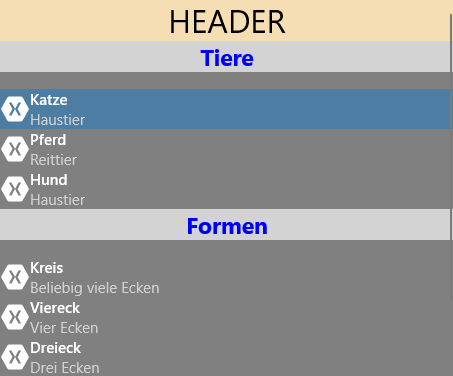
****

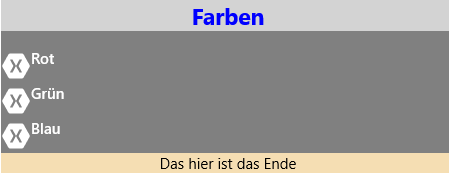
**C#:**

****



**Ausgabe:**

****

****

## 

## Picker

**Attribute**

ItemsSource: befüllt die Liste mit den angegebenen Werten

SelectedIndex: Gibt den Index des ausgewählten Items wieder

SelectedItem: Gibt das ausgewählte Item wieder

Items (readonly): Gibt eine Liste der Items wieder

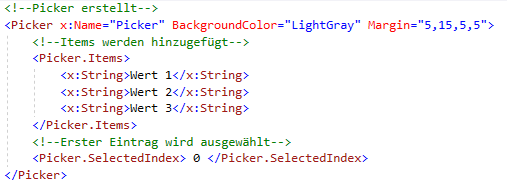
**Methoden/Events**

SelectedIndexChanged: wird aufgerufen, wenn das aktuell ausgewählte

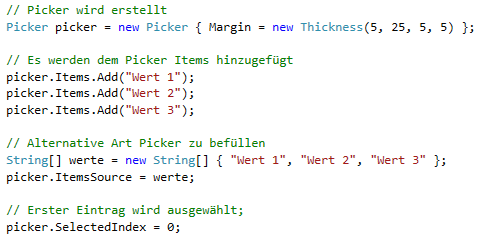
Item sich verändert

**Beispiel**

**XAML:**

****

**C#:**

****

**Ausgabe:**

****

****

## 

## ProgressBar

**Attribute**

Progress: Ein Wert zwischen 0 und 1, der bestimmt zu welchem

Anteil die Progressbar gefärbt ist.

**Methode**

ProgressTo(Double, UInt32, Easing) : System.Threading.Tasks.Task<bool>  
Der Fortschritt bis zu einem gewissen Punkt wird Animiert.

**Beispiel**

ProgressTo(.7, 2500, Easing.Linear)

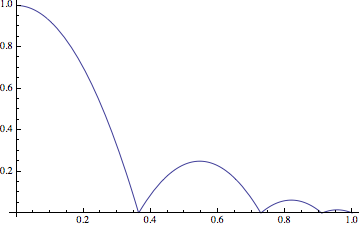
Der Fortschritt bis 70% ausgefüllter Fläche wird über 2,5 Sekunden linear

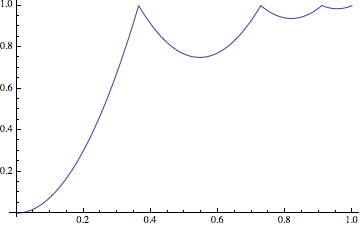
animiert.

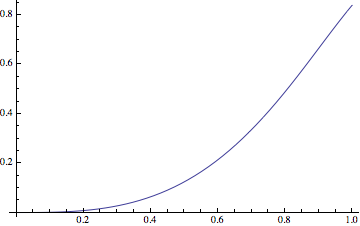
**Easing**

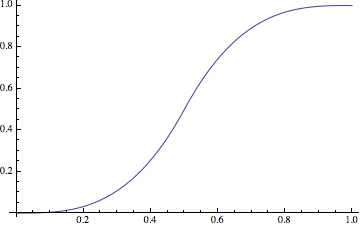
Die Art und Weise mit der die Progressbar animiert werden soll

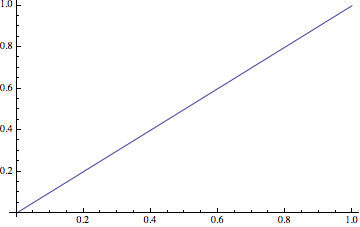
<https://developer.xamarin.com/api/type/Xamarin.Forms.Easing/>

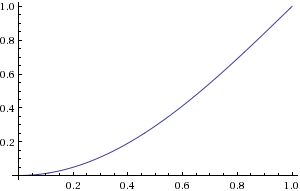
BounceIn 

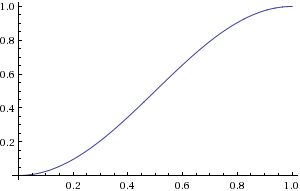
BounceOut: 

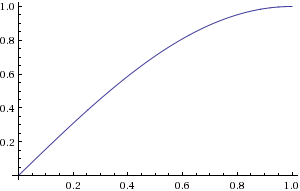
CubicIn 

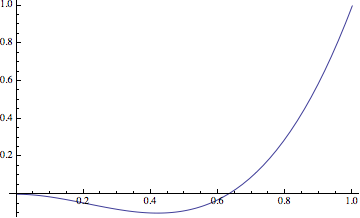
CubicInOut 

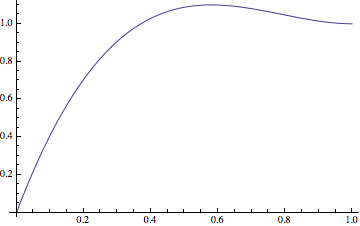
Linear 

SinIn 

SinInOut 

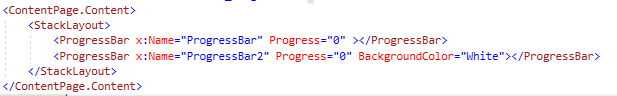
SinOu t 

SpringIn 

SpringOut 

**Beispiel**

**XAML:**

****

**C#**

****

**Ausgabe**

****



## 

## SearchBar

Bei der Searchbar handelt es sich um ein Entry mit einem button.

**Attribute**

CancelButtonColor Die Farbe des CancelButtons

FontAttributes Die FontAtribute des Textes (bold, italic, ...)

FontFamily Die Schriftart des Textes

FontSize Die Größe der Schrift.

HorizontalTextAlignment Anordung der Schrift auf horizontaler Ebene

Placeholder Der Platzhalter, der angezeigt wird wenn noch kein Text

vorhanden ist

PlaceholderColor Die Farbe des Platzhalters

SearchCommand Der Befehl, der ausgeführt wird wenn der SearchButton

gedrückt wird.

SearchCommandParameter Der Parameter, der an das SearchCommand

übergeben wird.

Text Der Text der in der Searchbar angezeigt wird.

TextColor Die Farbe des Textes, der in der Searchbar angezeigt

wird.

**Methoden/Events**

**SearchButtonPressed** Das Event wird ausgelöst, wenn der Search

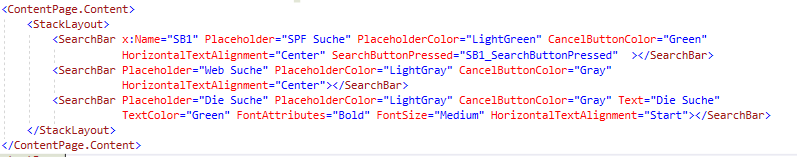
Button betätigt wird

**TextChanged** Das Event wird ausgelöst, wenn sich der Text

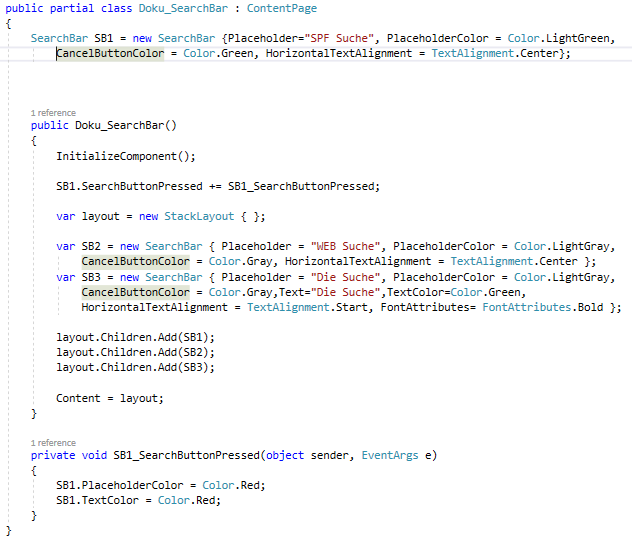
der Search bar ändert.

**Beispiel**

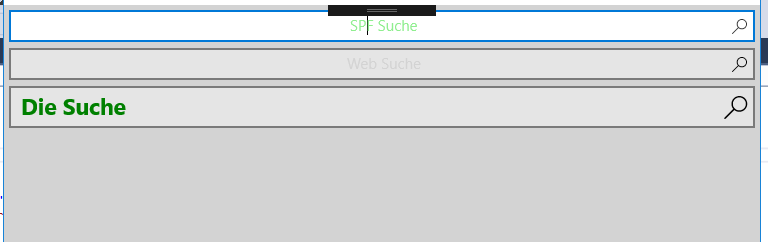
**XAML**

****

**C#**

****

**Ausgabe**

****

## 

## Stepper

bei den Steppern handelt es sich um zwei Buttons mit dem man einen Wert inkrementieren und dekrementieren kann.

**Attribute**

**Increment** Der Doublewert, um den entweder inkrementiert oder

dekrementiert werden soll.

**Maximum** Der Doublewert, der dem Stepper den Maximalen Wert gibt über

den nicht hinaus erhöht werden soll.

werden kann.

**Minimum** Der Doublewert, der dem Stepper den MinimaleWert gibt über

den nicht hinaus verringert werden soll.

**Value** Der momentane Wert des Steppers

**Methoden/Events**

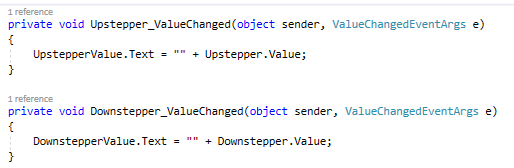
**ValueChanged** Das Event wird ausgelöst, wenn sich der Wert des

Steppers ändert.

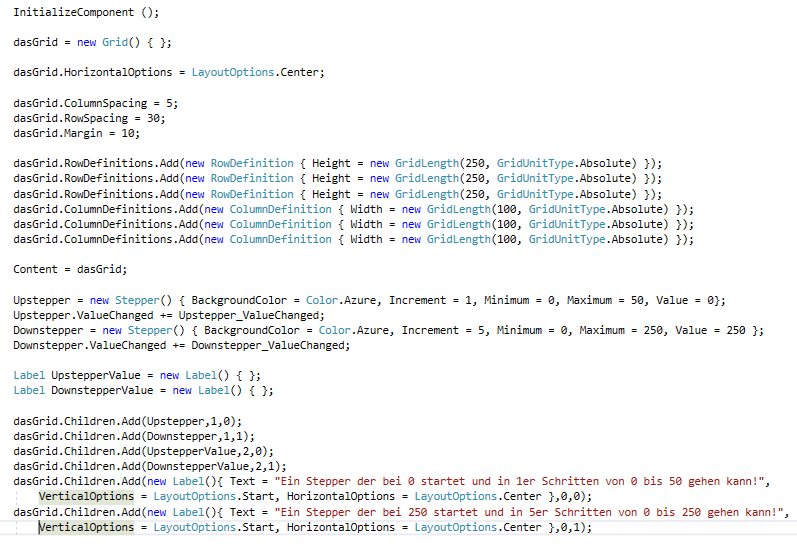
**Beispiel**

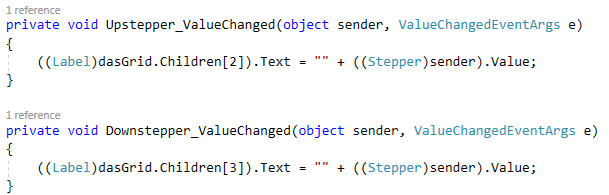
**XAML**

****

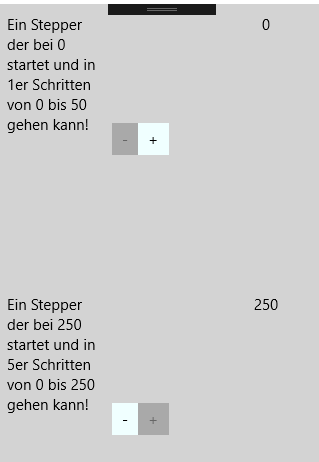
****

**C#**

****

****

**Ausgabe**

****

## 

## Switch

**Attribute**

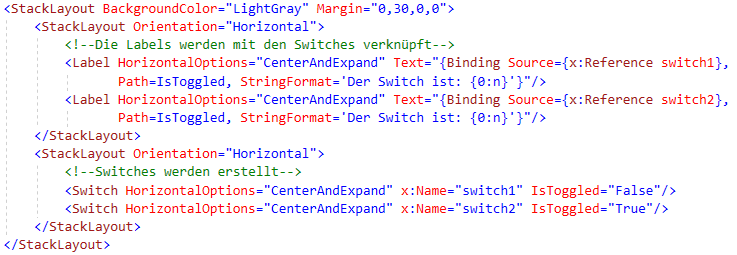
IsToggled: Gibt wieder, ob der Switch an oder aus ist

**Methoden/Events**

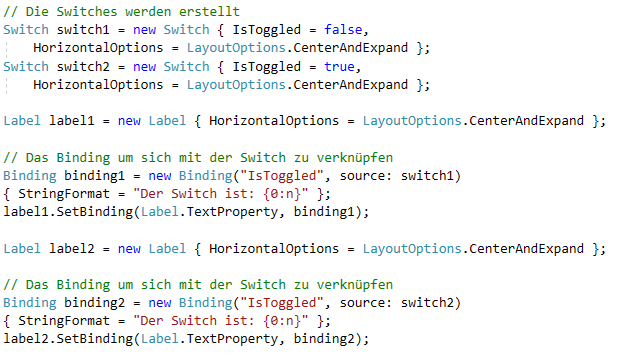
Toggled: wird aufgerufen, wenn” IsToggled” sich verändert

**Beispiel**

**XAML**

****

**C#**

****

**Ausgabe**

****

## 

## TableView (für Listen mit Eingabefeldern)

Der unterschied beim TableView zum List ist, dass es keine Item Source gibt und man davon ausgeht, dass die Zellen unterschiedlich sind.

**Attribute**

Intent: Ist eine Enumeration die angibt auf welche Art das TableView

eingesetzt werden soll.

**Data** Um Dateneinträge anzuzeigen. (Die List View ist dafür

meist geeigneter)

**Form** Die Tableview Einträge sollen wie ein Formular

verwendet werden.

**Menu** Wenn man ein Menu aus Auswahlen haben möchte.

**Settings** Wird für Konfigurationsoptionen verwendet.

RowHeight Die Höhe der einzelnen Zellen lassen sich darüber

zuweisen.

HasUnevenRows Bei True berechnet Xamarin.Forms für jede Reihe die

optimale Höhe selbst.

Root: Attribut zum setzen des “TableRoot”.

**Methoden/Events**

**Keine exklusive Methoden oder Events**

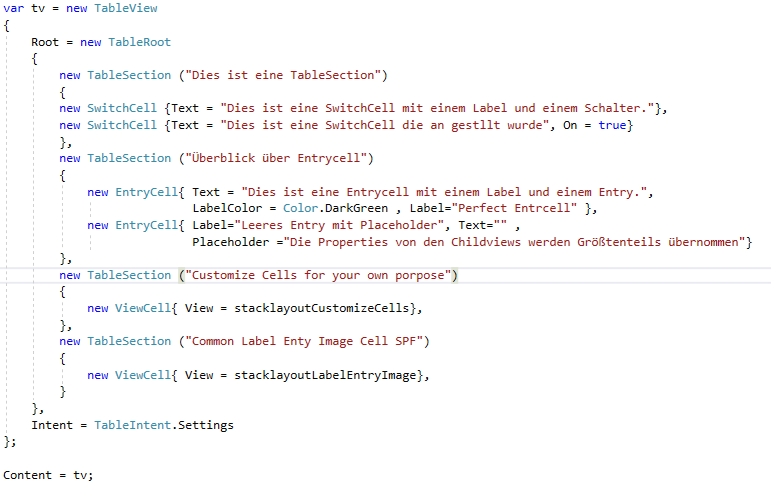
**Beispiel**

**XAML**

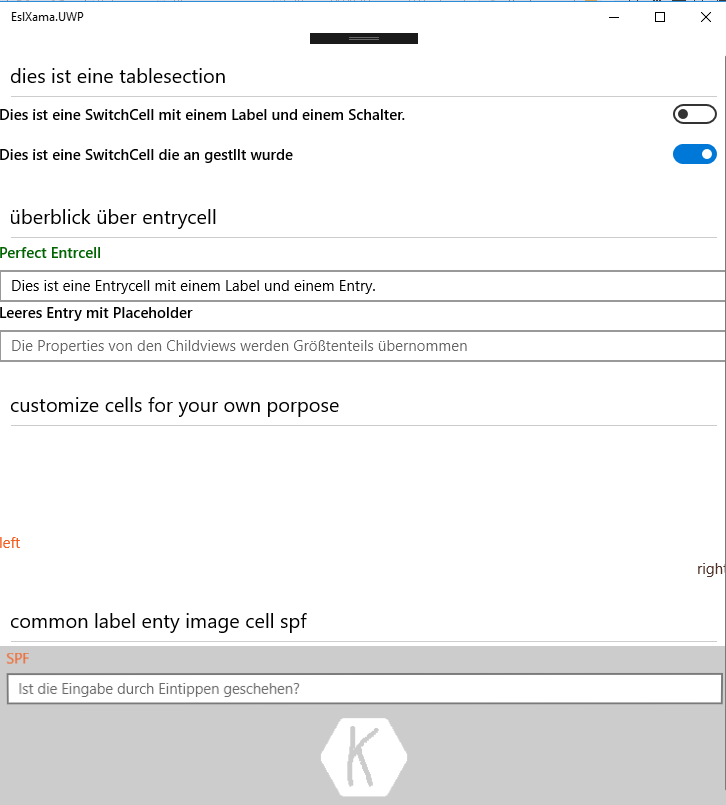
****

**C#**

****

****

**Ausgabe**

****

## 

## TimePicker

**Attribute**

**Format** Das Zeitformat, mit dem die Zeit angezeigt werden

soll.

**TextColor** Die Farbe des Textes innerhalb des TimePickers.

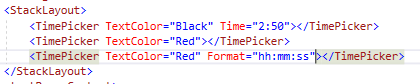
**Time** Die Zeit, die angezeigt wird

**Methoden/Events**

**Keine exklusiven Methoden oder Events**

**Beispiel**

XAML



C#



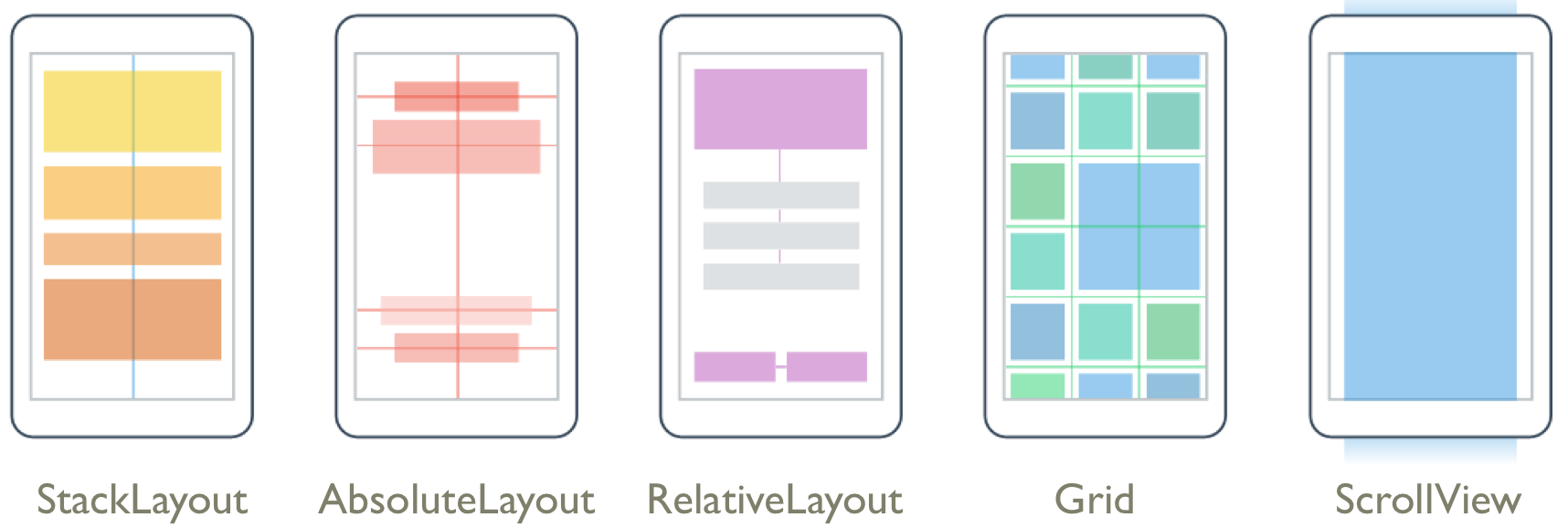
Ausgabe

# 

# 

# 

# Xamarin Layouts



**Erklärung:**

Layouts werden dafür verwendet um die verschiedenen Elemente mithilfe einer Struktur

darzustellen. Hierbei kann man die Layouts auch ineinander verschachteln, um somit

komplexere Layouts zu ermöglichen.

Sowohl ScrollView als auch das RelativeLayout eigenen sich dabei besonders gut für den Start einer Verschachtelung, da sie ihre Layout Properties an beliebig viele Ebenen nach oben zur verfügung stellen können. Sie ermöglichen so allen Views die Innerhalb von ihnene sind mitzuscrollen oder an anderen Views skaliert und positioniert zu werden.

## 

## AbsoluteLayout

Das AbsoluteLayout wird vor allem dazu verwendet um pixelgenaue GUIs zu erstellen.

Die Children einer VIew können dann anhand zum Parent proportional gesetzt oder mit manuellen Werten gesetzt werden.

Mit diesem Layout kann man relativ einfach Elemente zentrieren oder sie an eine Seite anzuheften.

**Attribute**

Children (readonly): gibt eine Liste der Elemente innerhalb des Layouts

wieder

**Attribute für innerhalb des Layouts:**

AbsoluteLayout.Layoutbounds: Spezifiziert die Positionierung und Größe

des Elements innerhalb des Layouts

(X, Y, Breite, Höhe)

AbsoluteLayout.AbsoluteLayoutFlags:

“None”: Alle Werte werden als absolut interpretiert

“All”: Alle Werte werden als proportional interpretiert

“WidthProportional”: Die Breite Werte werden als proportional und

alle anderen Werte als absolut interpretiert

“HeightProportional”: Die Höhe Werte werden als proportional und alle

anderen Werte als absolut interpretiert

“XProportional”: Die X - Werte werden als proportional und alle

anderen Werte als absolut interpretiert

“YProportional”: Die Y - Werte werden als proportional und alle

anderen Werte als absolut interpretiert

“PositionProportional”: Die X - und Y - Werte werden als

proportional und die Größen Werte als

absolut interpretiert

“SizeProportional”: Die Breite und Höhen Werte werden als

proportional und die Positions - Werte als

absolut interpretiert

**Erklärung:** Proportional: Werte zwischen 0 und 1

im Verhältnis zum Layout (z.b. X = 0.5

heißt in der Mitte des Layouts)

Absolut: Konkreter Wert innerhalb des Layouts

**Methoden/Events**

GetLayoutBounds(BindableObject):

Gibt ein Rectangle vom Rahmen des angegebenen Elements

wieder

GetLayoutFlags(BindableObject):

Gibt die spezifizierten LayoutFlags des Ementes zurück

SetLayoutBounds(BindableObject, Rectangle):

Setzt das angegebene Element mit den Rectangle als

Rahmenswerte innerhalb des Layouts

SetLayoutFlags(BindableObject,AbsoluteLayoutFlags):

Übergibt dem angegebenen Element die mitgegebene

LayoutFlags

override LayoutChildren(Double, Double, Double, Double):

Positioniert und ändert die Größe des Kindes?

override OnChildAdded(Element):

wird aufgerufen, wenn ein Element dem Layout hinzugefügt

wird

override OnChildRemoved(Element)

wird aufgerufen, wenn ein Element aus dem Layout entfernt

wird

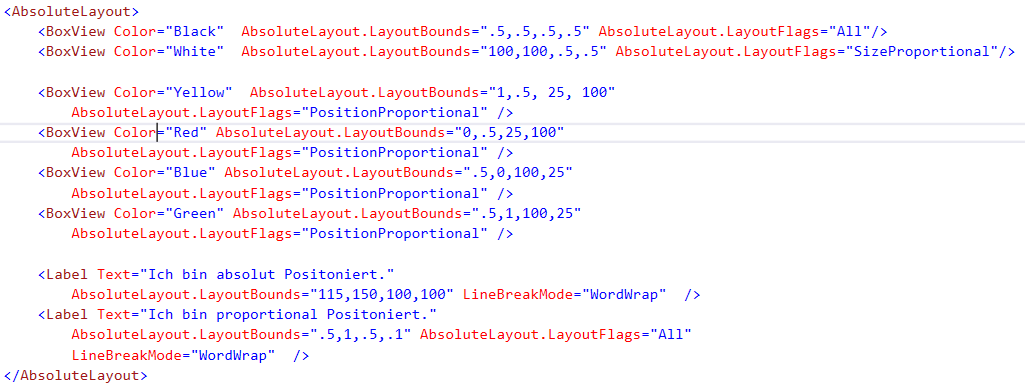
override OnSizeRequest(Double, Double):

wird aufgerufen, wenn die Größe des Layouts verändert wird

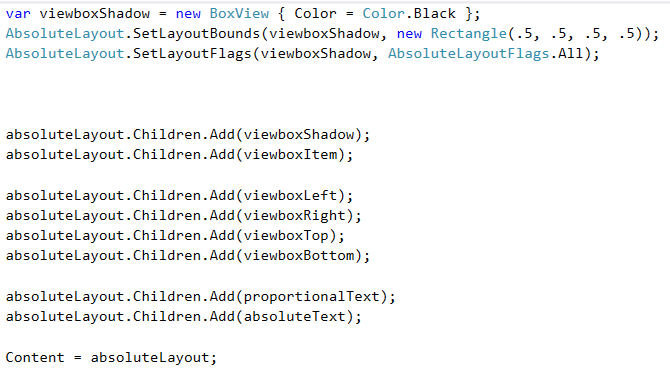
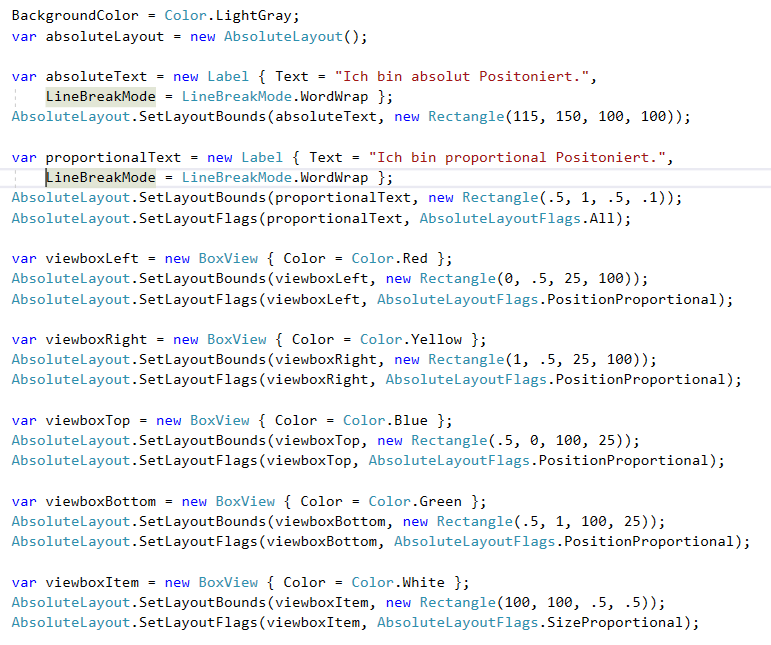
und gibt ein Objekt der Klasse SizeRequest zurück

**Beispiel**

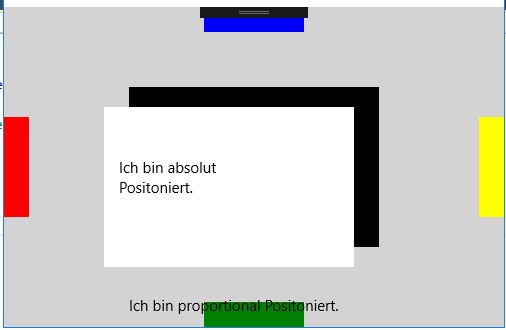
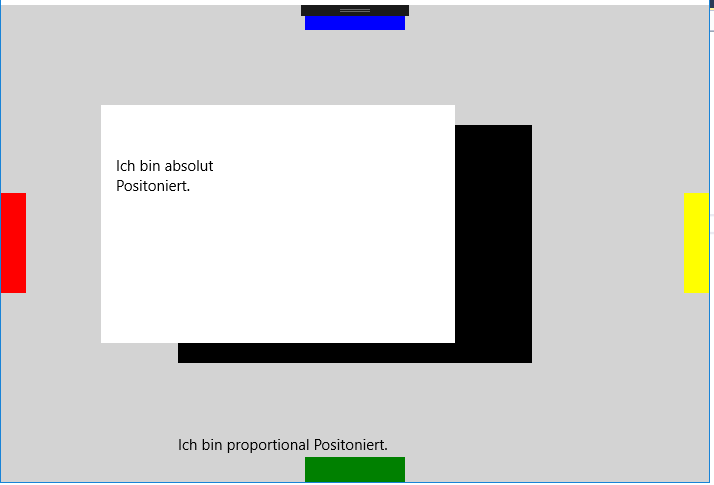
**XAML**

****

**C#**

****

**Ausgabe**

****

## 

## Grid

Mit dem Grid kann man schnell Reihen und Spalten erstellen.

Mit dem GridLength Typ “Star” kann man einfach Zeilen proportional zueinander machen.

Dies Layout ist hervorragend für tabellenartige Seiten geeignet.

**Attribute für innerhalb des Layouts**

Grid.ColumnDefinitions Die Spalten für das Grid werden definiert.

Grid.RowDefinitions Die Reihen für das Grid werden definiert.

ColumnSpacing Der Abstand zwischen zwei Spalten.

RowSpacing Der Abstand zwischen zwei Reihen

**Attribute für Views innerhalb des Grids**

Grid.Row Wird verwendet um dem View in dem Grid zu sagen in

welcher Reihe es sich befindet

Grid.RowSpan Wird verwendet um dem View die Anzahl der Reihen

mitzugeben, über das sie sich erstreckt.

Grid.Column Wird verwendet um dem View in dem Grid zu sagen in

welcher Spalte es sich befindet

Grid.ColumnSpan Wird verwendet um dem View die Anzahl der Spalten

mitzugeben, über das sie sich erstreckt.

**Methoden/Events**

**Zugehörige Klassen**

GridLength

**Attribute**

GridUnitType Dabei Handelt es sich um eine Enumeration, die

bestimmt wie GridLength.Value gedeutet werden

soll.

Absolute: Die GridLength.Value wird als absoluter Wert

interpretiert

Auto: Die GridLength.Value wird ignoriert und ein Wert

wird gewählt, der das Children Objekt umschließt.

Star: Die GridLength.Value wird als proportionaler

Wert gedeutet, der erst nach den Elementen mit

Absolute und Auto bestimmt wird.

**Beispiel**

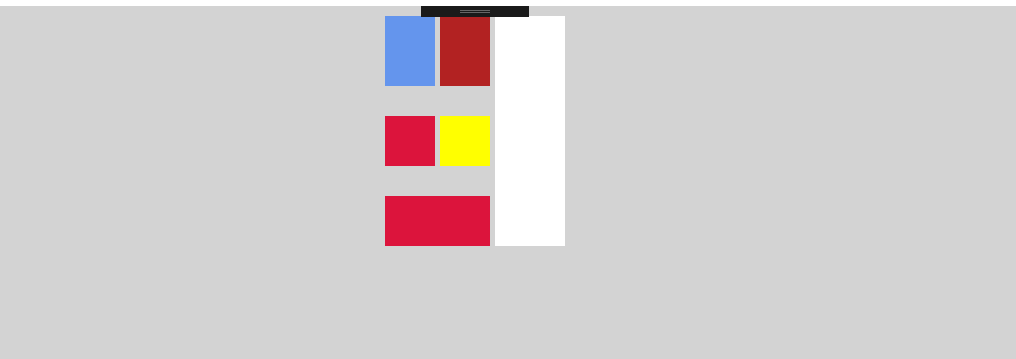
**XAML:**

****

**C#:**

****

**Ausgabe:**

****

## RelativeLayout

**Erklärung:**

Wird verwendet um Ui’s zu bauen, die zu jeder Bildschirmgröße passen.

Man richtet die Position und Größe der einzelnen Views relativ an einem der

SiblingViews an oder am Layout selbst aus.

Dabei müssen die beiden Views nicht auf der selbene Ebene sein.

**Attribute / Constraints**

Die Positionierung innerhalb des RelativeLayouts erfolgt über

**Constraints**.

**ConstraintExpression Properties**:

Type: Gibt an ob der Constraint relativ zu einem

anderen View oder dem Elternview ist.

RelativeToView

RelativeToParent

Property: Die Eigenschaften des Bezugsviews,

die als Referenz genommen wird.

Factor: Der Faktor, der mit dem Wert der Eigenschaft

verrechnet wird.

Constant: selbst festlegbare Differenz zum

Bezugs-Property.

ElementName: Der Name des Elementes zu dem dieser View

relativ ist.

**Attribute für Views innerhalb des RelativeLayouts**

RelativeLayout.XConstraint: Der Constraint, der für die X-Position des

Views zuständig ist.

RelativeLayout.YConstraint: Der Constraint, der für die Y-Position des

Views zuständig ist.

RelativeLayout.WidthConstraint: Der Constraint, der für die Weite des

Views zuständig ist.

RelativeLayout.HeightConstraint: Der Constraint der für die Höhe des

Views zuständig ist.

**Die Views können sowohl über die Constraints, als auch über die**

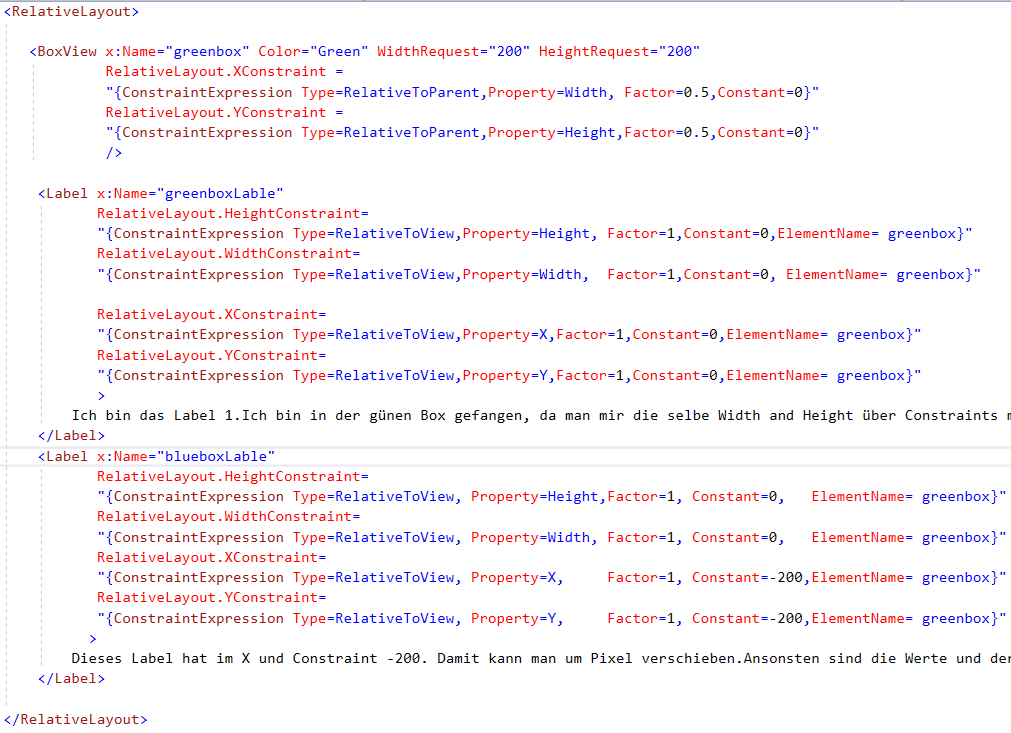
**herkömmlichen Attribute HeightRequest und WidthRequest ihre Werte**

**bekommen, diese können im Zweifel aber von den Constraints**

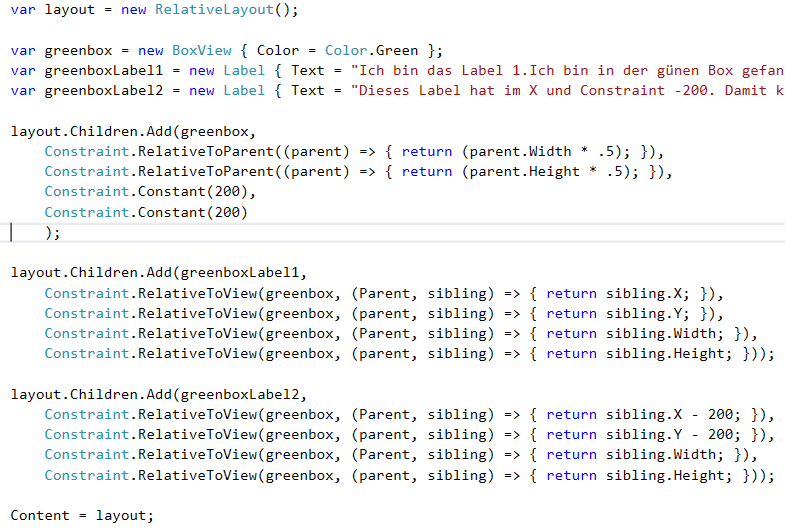
**überschrieben werden.**

**Beispiele**

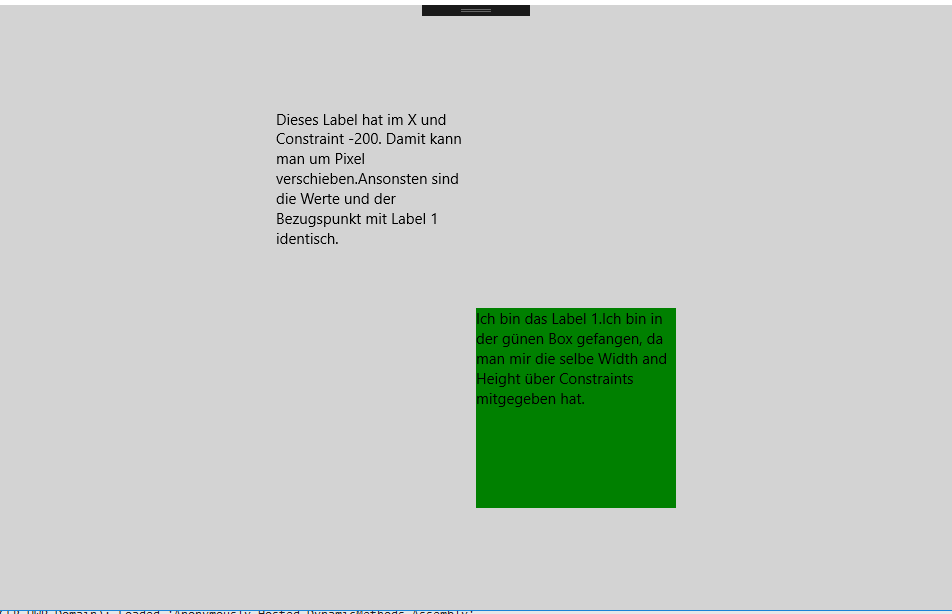
**XAML**

****

**C#**

****

**Ausgabe**

****

## Stacklayout

**Das Stacklayout ist dafür geeignet Elemente entlang einer horizontalen oder vertikalen**

**Linie zu erstellen.**

**Dabei bestimmt die Reihenfolge, wie die Children hinzugefügt wird, auch über die Reihenfolge in der Darstellung.**

**Attribute**

Spacing: Der Abstand zwischen den Views

Orientation: Gibt an ob die Views horizontal oder vertikal stecken

“Vertical”: Platziert die Views in einer vertikalen Reihenfolge

“Horizontal”: Platziert die Views in einer horizontalen Reihenfolge

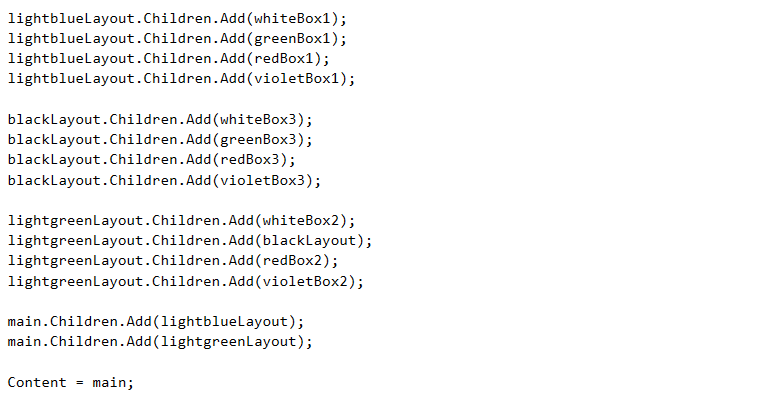
**Attribute für Elemente Innerhalb des Layouts:**

**Beispiele**

**XAML**

****

**C#**

****

**Ausgabe**

****

## ScrollView

**Erklärung:**

Wird vor allem als Ergänzung dann verwendet, wenn das verwendete Layout

nicht mehr auf eine Seite passt.

Man sollte das ScrollView immer als oberstes Layout, der Views die man scrollen möchter,verwenden falls man ein verschachteltes Layout plant.

Momentan sind von den vier anderen Layout alle außer dem

StackLayout mit der ScrollView ohne Probleme verwendbar.

Views die Bereits von sich aus scrollen können sind ebenfalls problematisch, das

**Attribute**

Content: Der Inhalt des ScrollViews

ContentSize(read only): Gibt die Größe des Views als Size mit Höhe und

Breite aus.

Orientation(Enumeration): Gibt an ob sich das View Horizontal, Vertical,

oder in beide Richtungen (Both) scrollen lässt.

ScrollX(read only): Gibt die aktuelle Scroll-Position auf der X-Achse

an.

ScrollY(read only): Gibt die aktuelle Scroll-Position auf der Y-Achse

an.

**Methoden/Events**

ScrollToAsync()Der View wird entweder mit Koordinaten auf

eine bestimmte Position gescrollt

oder man kann auch durch das Benennen des Elementes

und der ScrollToPosition-Enumeration bis zu diesem Element scrollen.

Zusätzlich kann noch mithilfe eines Booleans bestimmt werden, ob es sich bei der Bewegung zum Element um eine flüssige Animation handeln soll

**scroll.ScrollToAsync(0, 150, true);**

**scroll.ScrollToAsync(label, ScrollToPosition.Start, true);**

ScrollToPosition (Enumeration).

Center Scrollt das Element zum Zentrum des

sichtbaren Views.

End Scrollt das Element zum Ende des sichtbaren

Views

MakeVisible Scrollt das Element so, dass es im View

sichtbar wird.

Start Scrollt das Element zum Anfang des sichtbaren

Views

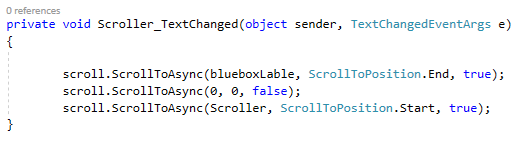
Scrolled Das Event wird ausgelöst, wenn der aktuelle View fertig ist mit

scrollen

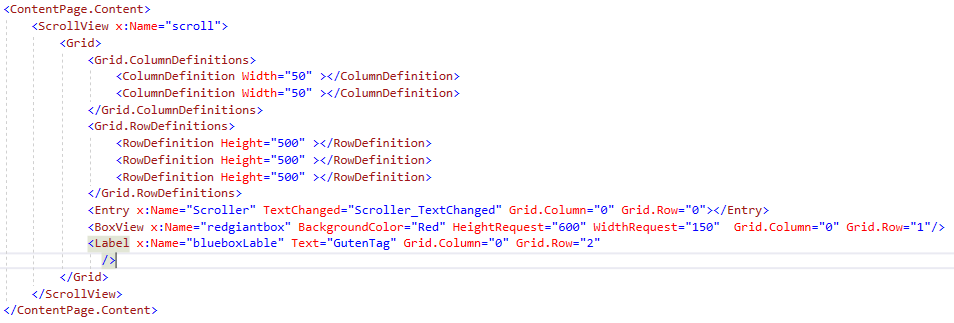
In dem **ScrolledEventArgs** für den Eventhandler werden auch die aktuellen X und Y Werte der Position über **ScrollX** und **ScrollY** mitgegeben.

**Beispiel**

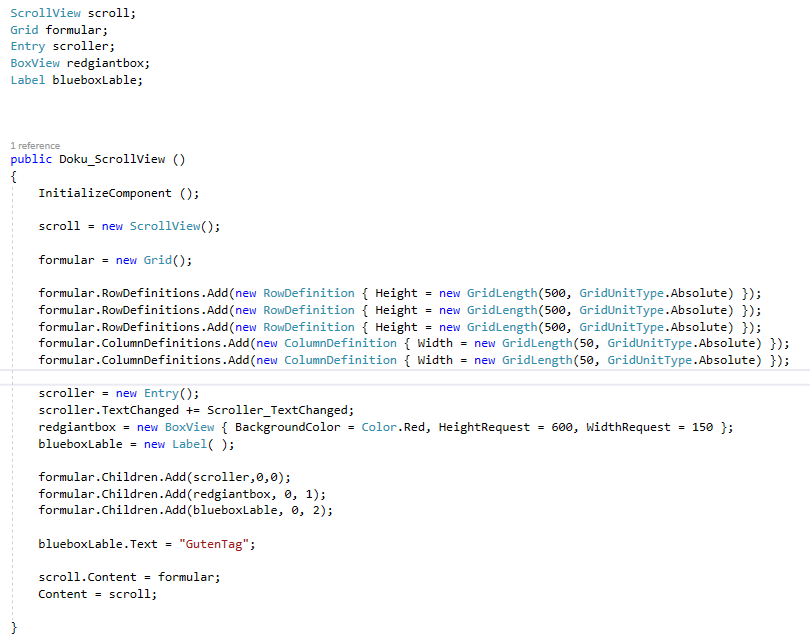
**Eventmethode**

****

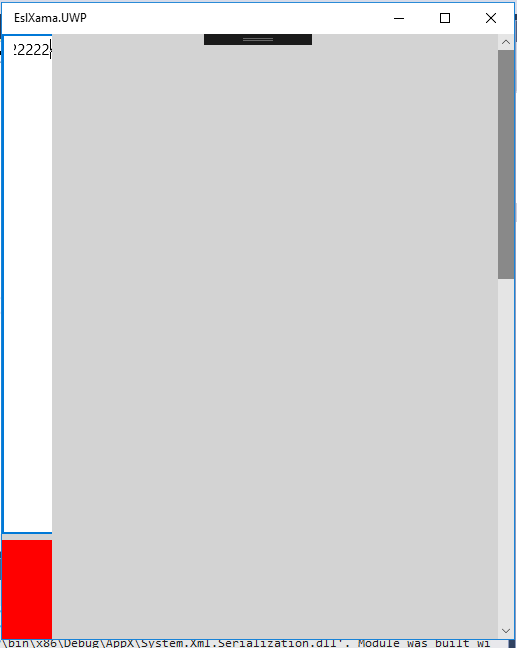
**XAML**

****

**C#**

****

**Ausgabe**

****

# 

# Sonstiges zu Xamarin

## Data Binding

**Erklärung:**

Data Binding ermöglicht das Verbinden zweier Properties miteinander. Bei

einer Veränderung einer dieser Properties, wird auch automatisch das andere

angepasst.

**BindingMode:**

Es gibt folgende Varianten von BindingModes:

Default Dies ist der Standard BindingMode der Property

TwoWay Die Daten gehen zwischen Ziel- und Ursprungsobjekt

in beide Richtungen

OneWay Die Daten gehen von Ursprungs- in das Zielobjekt

OneWayToSource Die Daten gehen vom Ziel- in das

Ursprungsobjekt

Bei den meisten Properties ist der Standard BindingMode OneWay, jedoch sind

bei folgenden Properties der Standard TwoWay:

* + - * “Date” Property der Klasse “DatePicker”
      * “Text” Property der Klassen “Editor”, “Entry”, “SearchBar”, und “EntryCell”
      * “IsRefreshing” Property der Klasse “ListView”
      * “SelectedItem” Property der Klasse “MultiPage”
      * “SelectedIndex” und “SelectedItem” Properties der Klasse “Picker”
      * “Value” Property der Klassen “Slider” und “Stepper”
      * “IsToggled” Property der Klasse “Switch”
      * “On” Property der Klasse “SwitchCell”
      * “Time” Property der Klasse “TimePicker”

**Binding Path:**

Hiermit kann man auf einen Objekt einer Collection oder auf

Properties zugreifen, die in andere Properties stecken.

**Binding Value Converter:**

Sollten die verknüpften Properties nicht zueinander konvertierbar sein, muss

man eine “Value Converter”- Klasse dafür erstellen.

Diese muss von dem Interface IValueConverter (aus dem Xamarin.Forms Namespace) erben.

Die Convert - Methode ist dabei für die Konvertierung vom Ursprungs- zum Zieproperty verantwortlich, wohingegen ConvertBack für die Konvertierung vom Ziel- zum Ursprungsproperty zuständig ist.

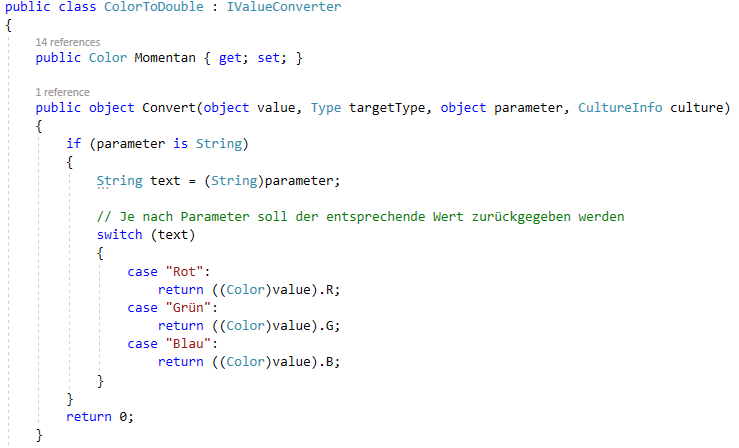
Dem Converter kann man auch einen ConverterParameter übergeben, da aber

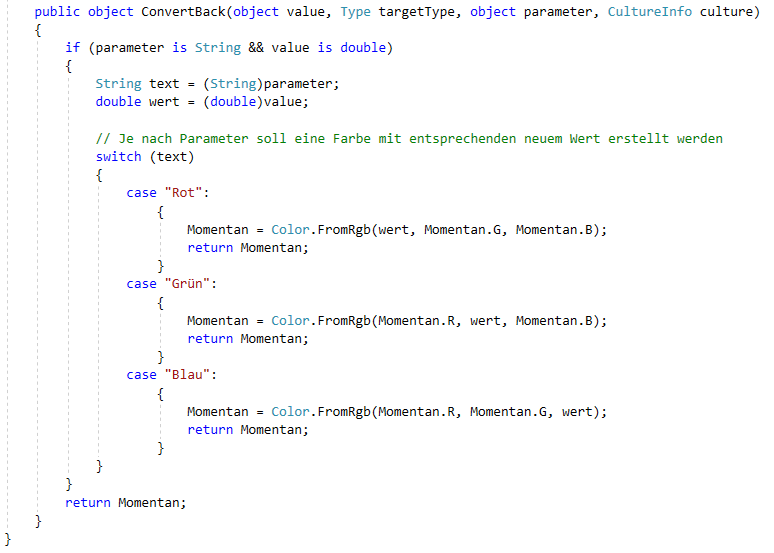
XAML und C# diesen Parameter eventuell unterschiedlich deuten könnten, wäre es sinnvoll vor der Konvertierung den Parameter zu überprüfen.

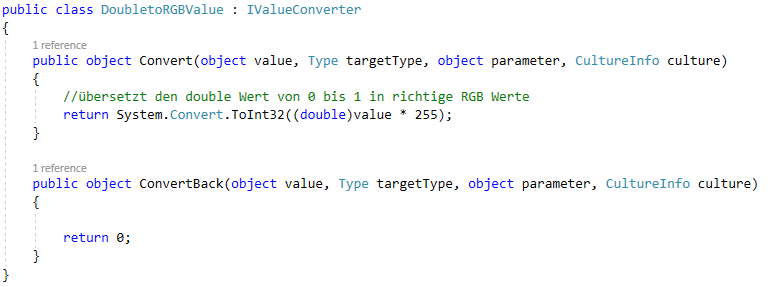
Die erstellten “Value Converter” Klassen können auch Properties und generische Parameter besitzen.

**Beispiel**

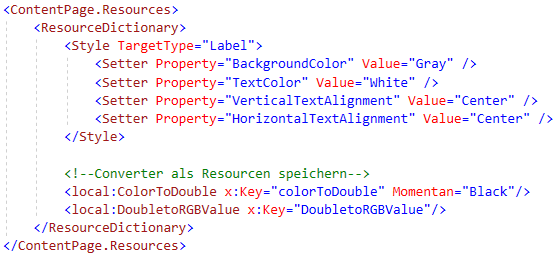
**Converter-Klassen:**

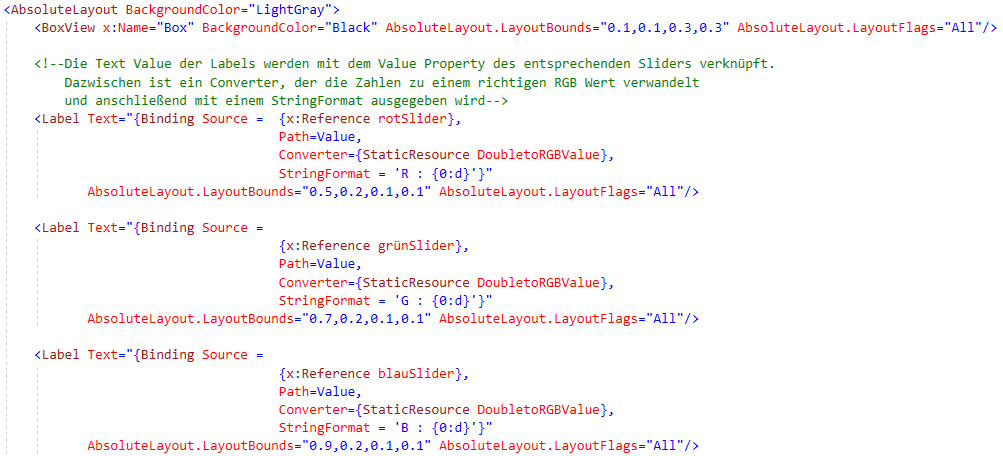
****

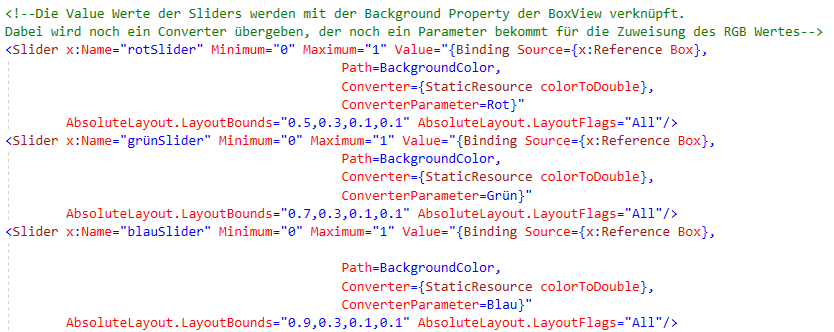
****

****

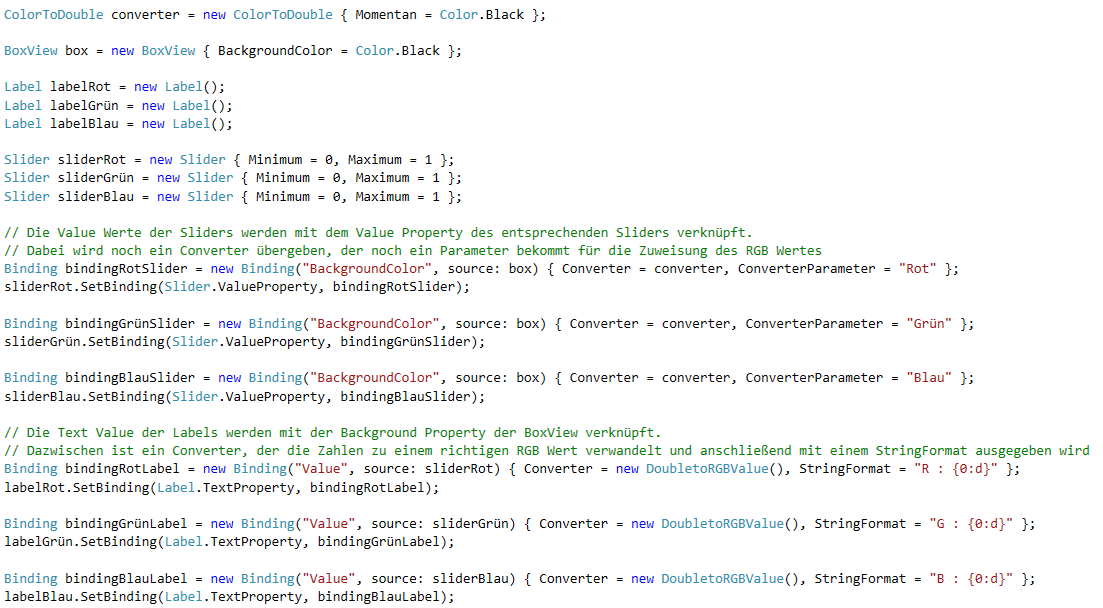
**XAML:**

****

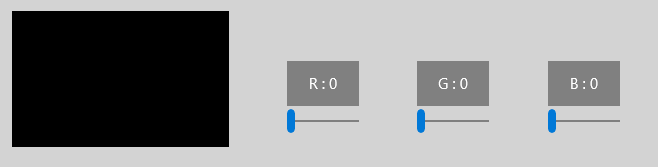
****

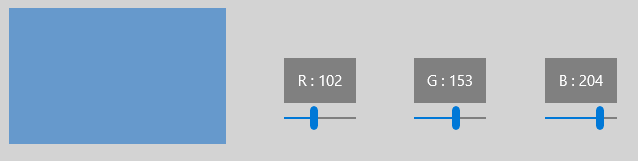
****

**C#:**

****

**Ausgabe:**

****

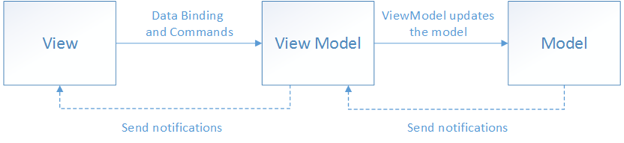
****

## 

## 

## 

## Model-View-ViewModel (MVVM)

****

**Erklärung:**

Das MVVM-Konzept ist eine Methode um die UI und die Programmlogik

voneinander zu trennen.

Dadurch vereinfacht man das Testen, sowie die weitere

Entwicklung und Erhaltung des Programmes. Ebenfalls kann man dadurch

auch einfacher Code wiederverwenden.

Dabei teilt man es in folgende drei Komponenten auf:

**View:**

Die View wird idealerweise in XAML geschrieben und ist für das Layout und

die Optik verantwortlich. Hier wird auch die optische Logik, wie z.B. das

Disablen von Buttons, mit hilfe von Triggers, Data Bindings und dergleichen

verwaltet.

Die Properties der Elemente werden dabei mit den Properties des ViewModels

verknüpft.

**View Model:**

Hier werden die Eigenschaften und Commands, an denen die View sich bindet,

und deren Logik festgestellt.

Das View Model ist ebenfalls für die Kommunikation zwischen View und den

Models verantwortlich.

**Model:**

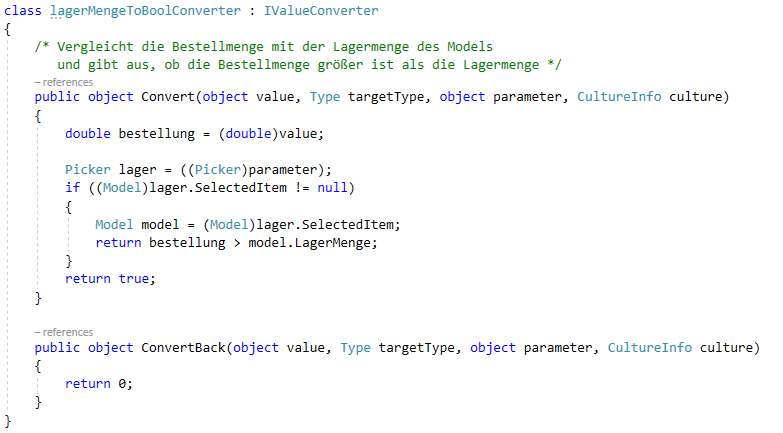
Models sind abgekapselte Klassen, die mit der grafischen Oberfläche nichts zu

tun haben und für die Daten zuständig sind. Meist sind diese in Verbindung mit

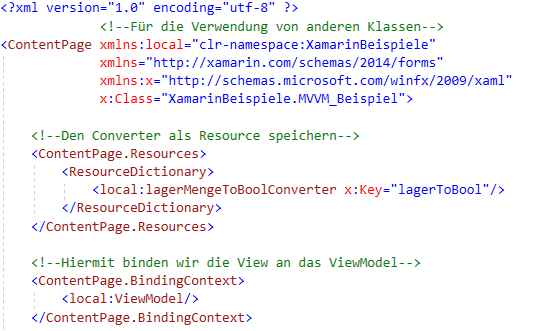
Datenbanken oder ähnliche Dienste.

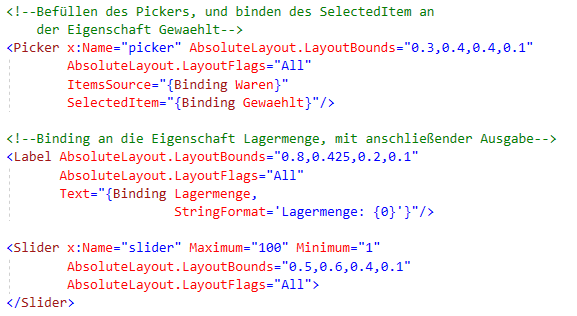
**Beispiel**

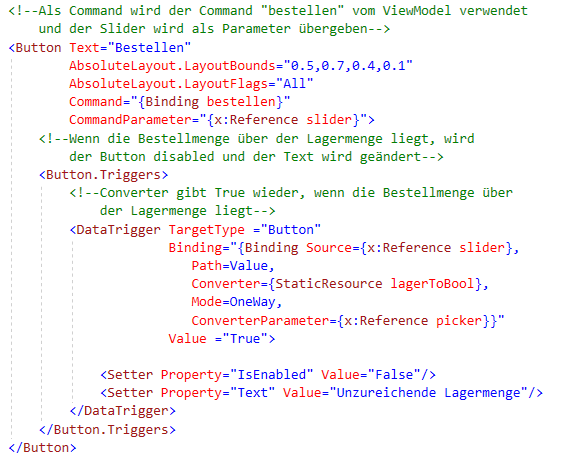
**Converter**

****

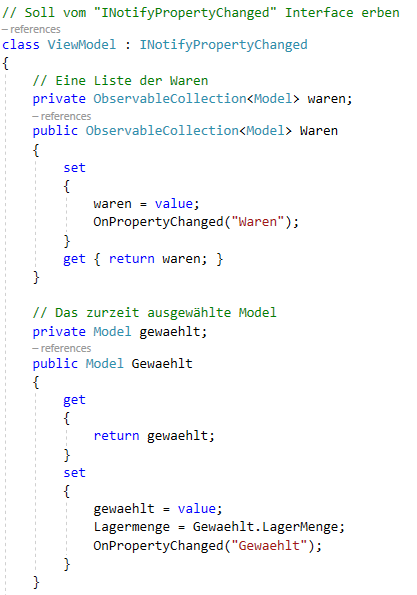
**View**

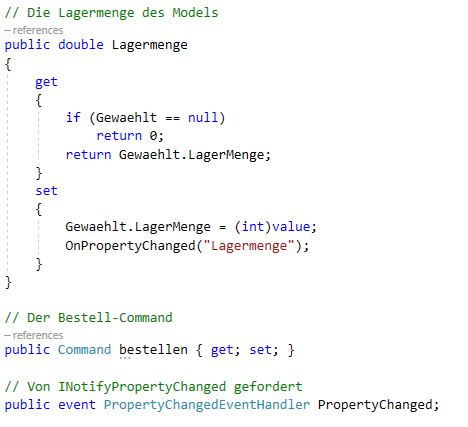
****

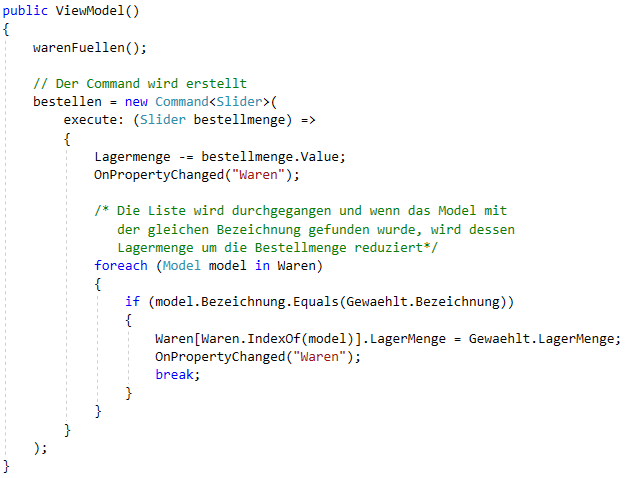
****

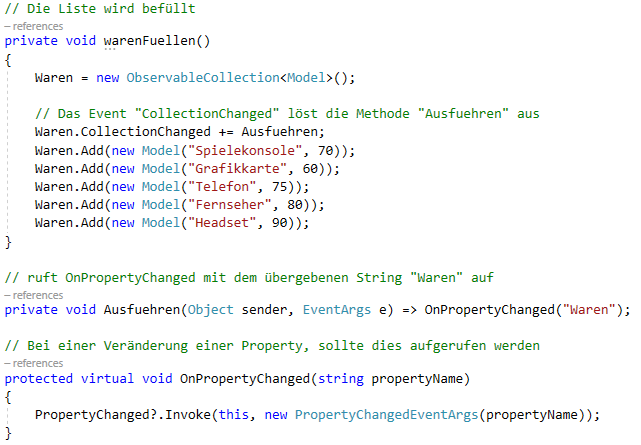
****

**ViewModel**

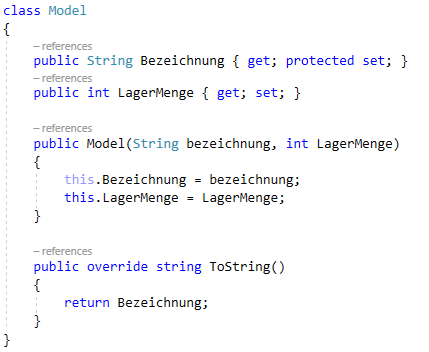
****

****

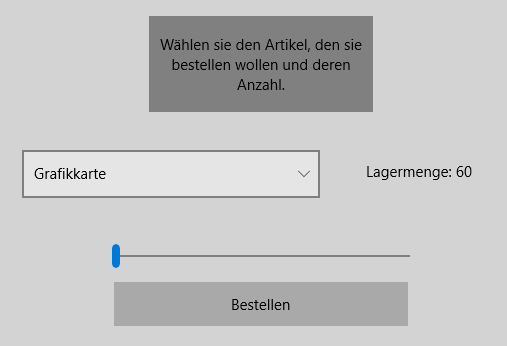
****

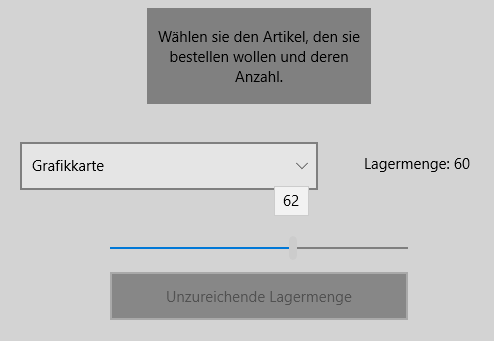
****

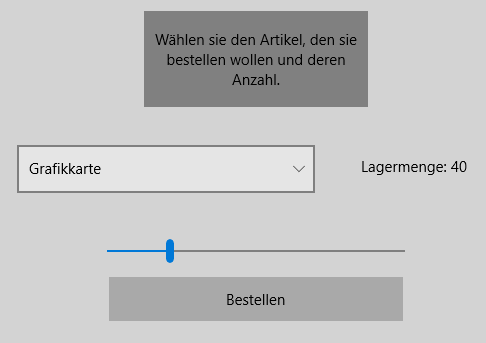
**Model**

****

**Ausgabe**

****

****

****

## 

## Styles

**Erklärung:**

Durch Styles kann man für verschiedene Elemente

einen einheitlichen Style erstellen.

Die Styles kann man auf dem Application - , Page - und Control - Level

definieren.

Styles lassen sich in folgende Kategorien einstufen:

Explizite Styles:

Gilt für die Elemente, die diesen Style zugewiesen werden.

Implizite Styles:

Gilt für alle Elemente ohne Zuweisung des Styles

Global Styles:

Diese Styles werden in der App Klasse/XAML erstellt und

können dann auf jeder Seite verwendet werden. Globale Styles

können explizit oder implizit sein.

Style Vererbung:

Styles können voneinander erben, um Duplikationen zu

verhindern und für eine bessere Wiederverwendung.

Dabei muss man die Hierarchie der Styles beachten.

Das heißt Application Levels können nur untereinander erben,

Page Levels untereinander und von Application Level und

Control Level von allen Styles.

Dynamische Styles:

Im Gegensatz zu den anderen Styles passen sich

diese Styles an Veränderungen an.

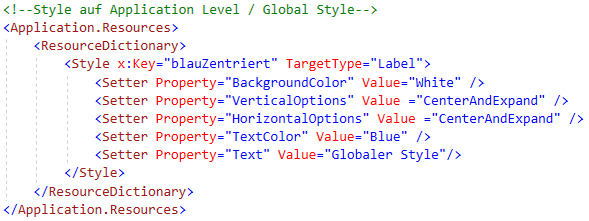
Device Styles:

Diese folgenden Styles sind exklusiv für die Labels:

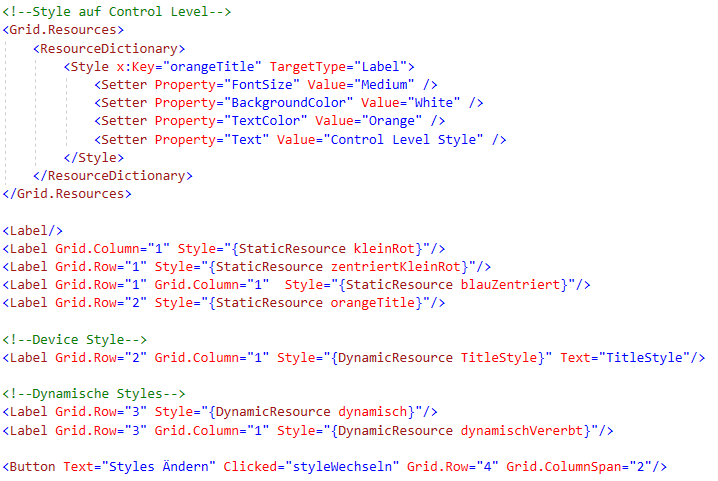
* + BodyStyle
  + CaptionStyle
  + ListItemDetailTextStyle
  + ListItemTextStyle
  + SubtitleStyle
  + TitleStyle

**Beispiel**

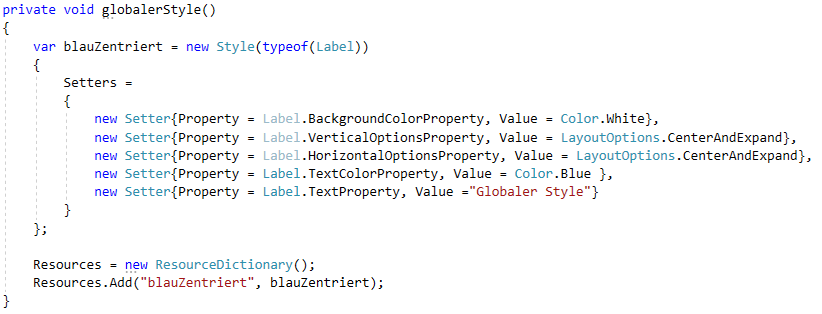
**XAML:**

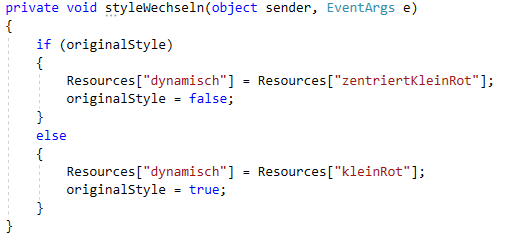
****



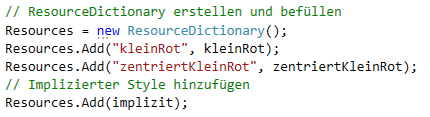


**C#:**

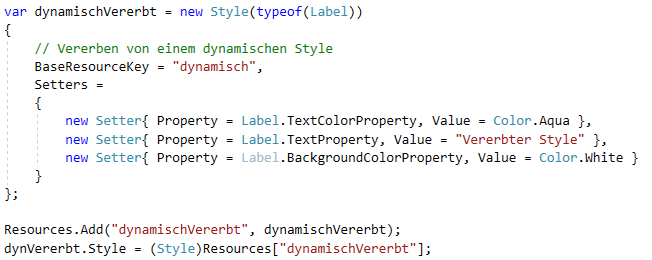
****

****

****

****

****

****

**Ausgabe:**

****

****

## Trigger

**Erklärung:**

Mithilfe von Triggers kann man das Aussehen verändern nachdem eine

bestimmte Situation aufgetreten ist. Triggers können auch in einem Style

festgelegt werden.

Diese sind die Bestandteile einer Trigger-Deklaration:

TargetType: Das Element, welches vom Trigger angesprochen wird

Property: Der zu überwachende Property

Value: Der Wert der Property, der den Trigger auslöst

Setter: Hier werden die optischen Veränderungen festgelegt

Enter- ExitActions: Rufen TriggerAction-Klassen am Anfang/Ende

einer Action auf

EnterActions und ExitActions können zusammen mit Settern definiert werden,

jedoch werden die Setters unabhängig von denen sofort aufgerufen.

Es gibt folgende Arten an Triggers:

**Property Trigger:**

Dieser Trigger wird ausgelöst, wenn die Property einen

festgelegten Wert besitzt.

**Data Trigger:**

Ähnlich wie der Property Trigger, jedoch werden mithilfe von Data Bindings

Properties von anderen Elementen überwacht.

**Event Trigger:**

Event Triggers werden ausgelöst, sobald ein festgelegtes Event ausgeführt wird.

Im Gegensatz zu den anderen Triggers wird hier jedoch eine Klasse aufgerufen,

die vom Interface TriggerAction erbt. In der zu überschreibenden Methode

“Invoke” werden dann die Veränderungen festgelegt. Dieser Trigger unterstützt

EnterActions und ExitActions nicht.

**Multi Trigger:**

Dieser Trigger ist ähnlich aufgebaut wie der Property- und Datatrigger, jedoch

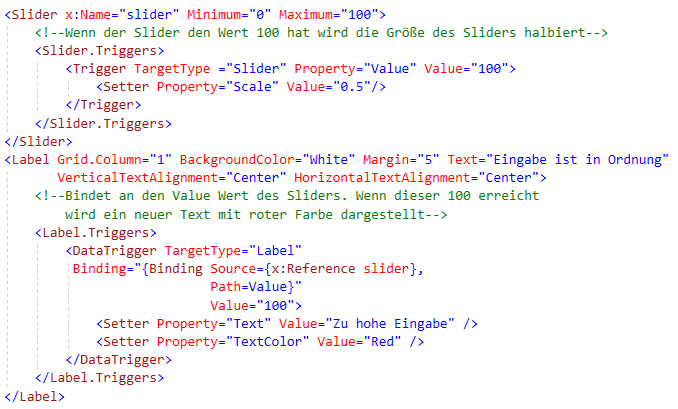
kann man hier mehrere Bedingungen definieren, die erfüllt werden müssen um

den Trigger auszulösen.

**Beispiel**

**XAML**

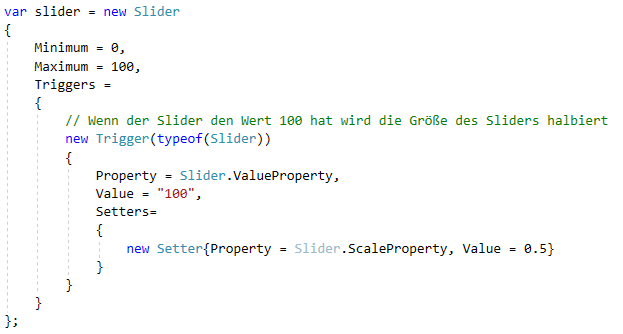
****

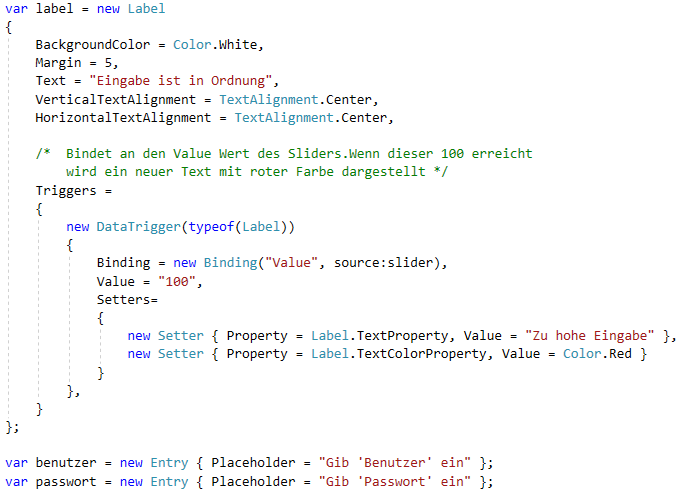
****

****

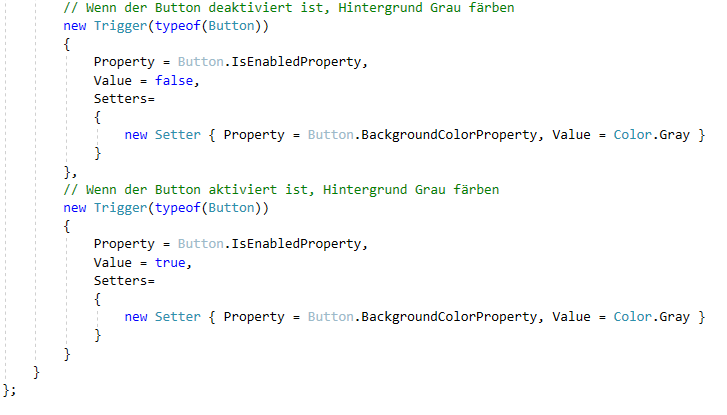
**C#**

****

****

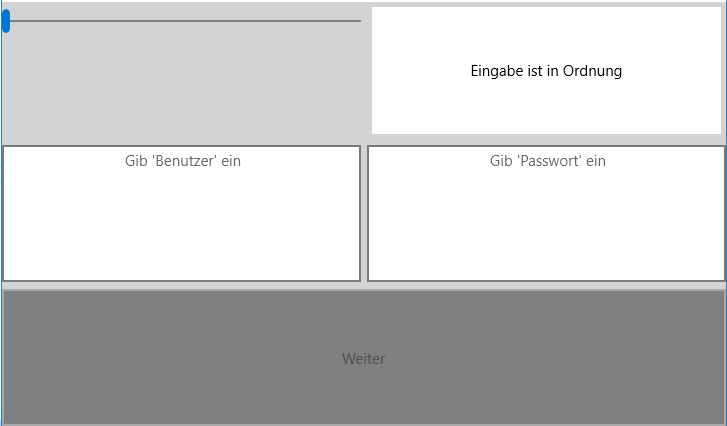
****

****

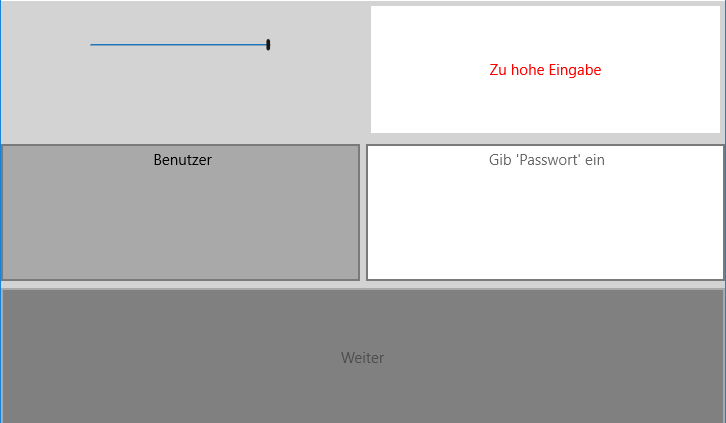
****

**Ausgabe:**

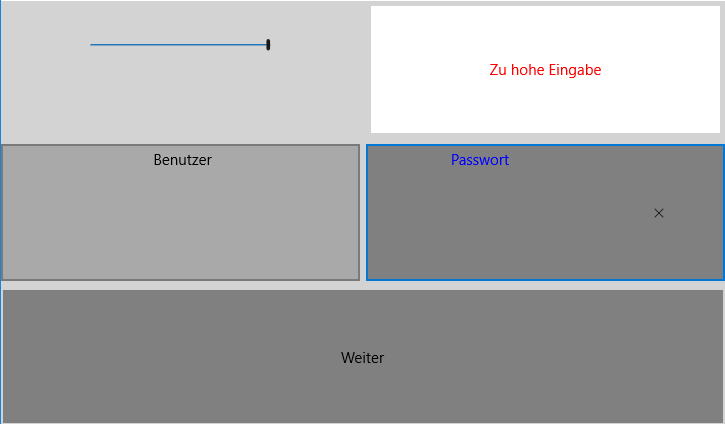
Anfang



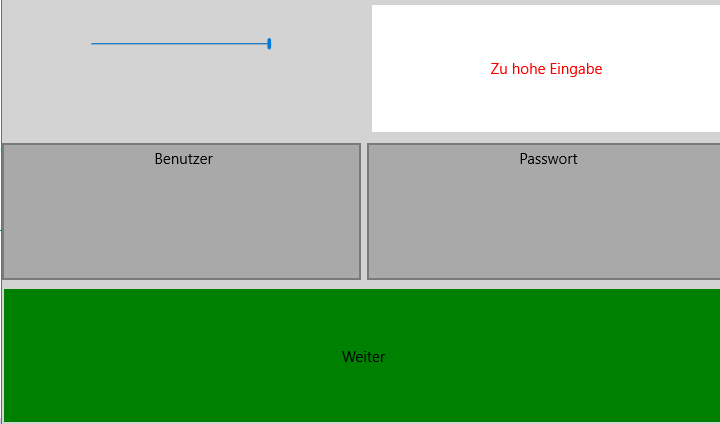
Slider bewegt und im linken Entry Text eingegeben



Im rechten Entry Text eingegeben und fokussiert, Button wurde enabled



Button wurde geklickt



# 

# Notizen:

Trigger:

<https://developer.xamarin.com/guides/xamarin-forms/application-fundamentals/triggers/>

StringFormat ausführlich

<https://msdn.microsoft.com/en-us/library/system.string.format(v=vs.110).aspx>

**StringFormat Überblick:** [**http://blog.stevex.net/string-formatting-in-csharp/**](http://blog.stevex.net/string-formatting-in-csharp/)

mit On Platform - Platform spezifisch z.b. On Platform=”UWP”

listviews in tableviews nicht möglich

Für Navigationsanfang:

MainPage = new NavigationPage(new XamarinESL.Hauptmenue());

Um Seiten der Navigation hinzuzufügen:

Navigation.PushAsync(Contentpage page);

data binding itemssource:

<https://stackoverflow.com/questions/28738090/how-to-set-itemssource-of-listview>

Übersicht der Views:

<https://developer.xamarin.com/guides/xamarin-forms/user-interface/controls/views/>

Farbverlauf?

mit custom renderer (daheim anschaun)