Inhalt

Gute Referenz zu XAMARIN (26.11.2020) : https://next-munich.com/blog/faqs-xamarin/

Apps mit Xamarin – User Interface Konzept

Intro

Es werden die folgenden Dokumente vorrausgesetzt:

Start Xamarin.Forms 2020.pdf

Start Reuter App Cross Platform 2020.pdf

Weitere vorbereitende Beispielprogramme

Xaml- Beispiel

Aus „Code snippetts aus Xamarin.Forms 2019.txt“ den Teil stackLayout in MainPage.XAML überschreiben und in App.xaml.cs zurück auf MainPage1 stellen:

```
<StackLayout Padding="10,0">
      <Label Text="Hello,Bay- Xamarin.Forms!"
         FontSize="Large"
         VerticalOptions="CenterAndExpand"
         HorizontalOptions="Center" />
      <Button Text = "Click Me!"
          VerticalOptions="CenterAndExpand"
          HorizontalOptions="Center" />
      <Switch VerticalOptions="CenterAndExpand"
          HorizontalOptions="Center" />
      <Slider VerticalOptions="CenterAndExpand" />
  </StackLayout>
```

Wenn man dann in Button ein Clicked ergänzt:

```
    <Button Text = "Click Me!"
       VerticalOptions="CenterAndExpand"
       HorizontalOptions="Center"
           Clicked="Button_Clicked"/>
    <Switch VerticalOptions="CenterAndExpand"
```

Und dann „gehe zur Definition“, dann wird der entsprechende Handler in MainPage1.xaml.cs erzeugt.

Aber das Kombinieren beider Seiten ist umständlich. Besser alles in einer Seite. Mit dem Snippet aus obiger Datei alles in neuer Seite Page1 einfügen mit dem gleichen Ergebnis.

Scroll View

Weiteres Beispielprogramm aus dem Petzold-Buch, das bei jedem Druck auf einen Button einen neuen Label mit der aktuellen Zeit erstellt. Dabei lernt man auch die Elemente StackLayout und ScrollView kennen. Wird die Seitenlänge überschritten, gibt es die Möglichkeit zu scollen.

Code hier und in „Code snippetts aus Xamarin.Forms Buch.txt“:

```
public class PageBay : ContentPage
{
    StackLayout loggerLayout = new StackLayout();
```

```
        public PageBay()
        {
            // Create the Button and attach Clicked handler.
            Button button = new Button
            {
                Text = "Log the Click Time",
                BorderColor = Color.Blue,
                BackgroundColor = Color.Red
            };
            button.Clicked += Button_Clicked;

              // Assemble the page.
            this.Content = new StackLayout
            {
                Children =
                {
                    button,
                    new ScrollView
                    {
                            VerticalOptions = LayoutOptions.FillAndExpand,
                            Content = loggerLayout
                    }
                }
            };
        }
        void Button_Clicked(object sender, EventArgs args)
        {
            // Add Label to scrollable StackLayout.
            loggerLayout.Children.Add(new Label
            {
                Text = "Button clicked at " + DateTime.Now.ToString("T")
            });
        }
    }
```

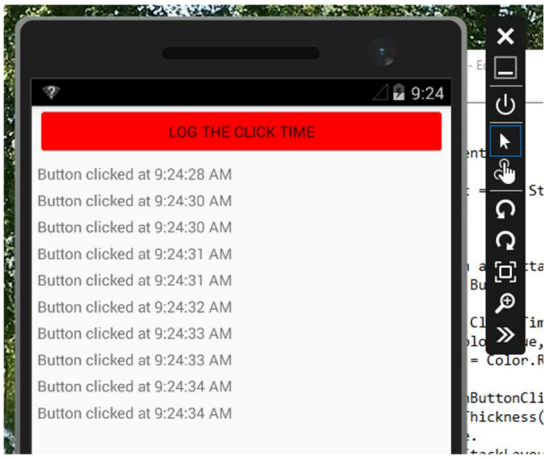
Man erkennt jetzt eine wichtige Eigenschaft der Programmierart in Xamarin.Forms: Die Eigenschaften werden in den jeweiligen Konstruktoren festgelegt und mit Kommata getrennt aufgelistet. Beim Button habe ich gegenüber dem Buch einfach mal drei Farbeigenschaften ergänzt.

Die Eventhandler für die Komponenten werden einfach wie gehabt mit

```
            button.Clicked += Button_Clicked;
```

definiert. Dann muss es außerhalb von Page1() aber natürlich diese Funktion geben, wird mit der Tab- Taste automatisch erzeugt.

Nach 10 Klicks auf den Button sieht das dann auf dem Android-Emulator so aus:



Klickt man weiter, so wird irgendwann die Liste zu lang und man kann scrollen.

User Interface

User Interface Guide im Internet

https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/

Gibt eine gute Übersicht über alle Elemente.

Weitere Beispiele auf Github

https://github.com/xamarin/xamarin-forms-samples

Auf diesen Seiten werden open- Source- Projekte veröffentlicht. Klingt gut. Am 13.11.2019 sind dort über 50 Verzeichnisse mit Projekten zu finden. Der Download ist öffentlich. Man kann nur alles komplett downloaden, was ziemlich lange dauert (ca. 1 h)

File: xamarin-forms-samples-master.zip (bei mir erhältlich, allerdings zu groß für Moodle)

User Interface Demo Beispiel FormsGallery

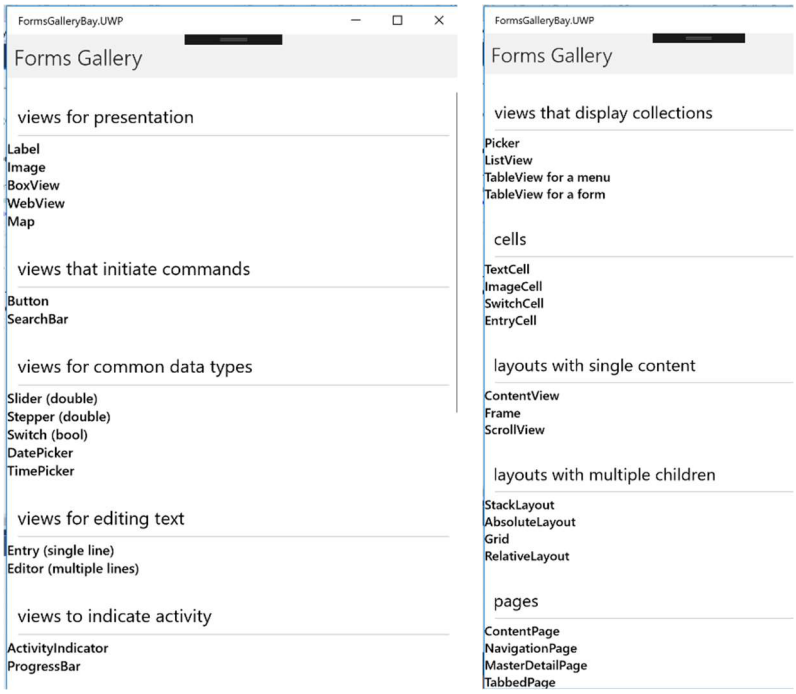
Dazu bitte die Datei „FormsGalleryBay18.zip“ herunterladen und auspacken. Darin befinden sich 39 cs- Dateien mit Beispielen für UI- Elemente, didaktisch hilfreich vom Autor in Github aufbereitet.

1. Start mit der neusten Vorlage der Cross- Platform, ich habe sie FormsGalleryBay genannt. Mit UWP und leerer Inhaltsseite.
2. Hinzufügen aller Seiten aus FormsGalleryBay18.zip außer App in das Hauptprojekt.
3. In App.xaml.cs dann auf neue Hauptseite „HomePage“ verweisen, allerdings diesmal mit „NavigationPage“:
4. `MainPage = new NavigationPage(new HomePage());`

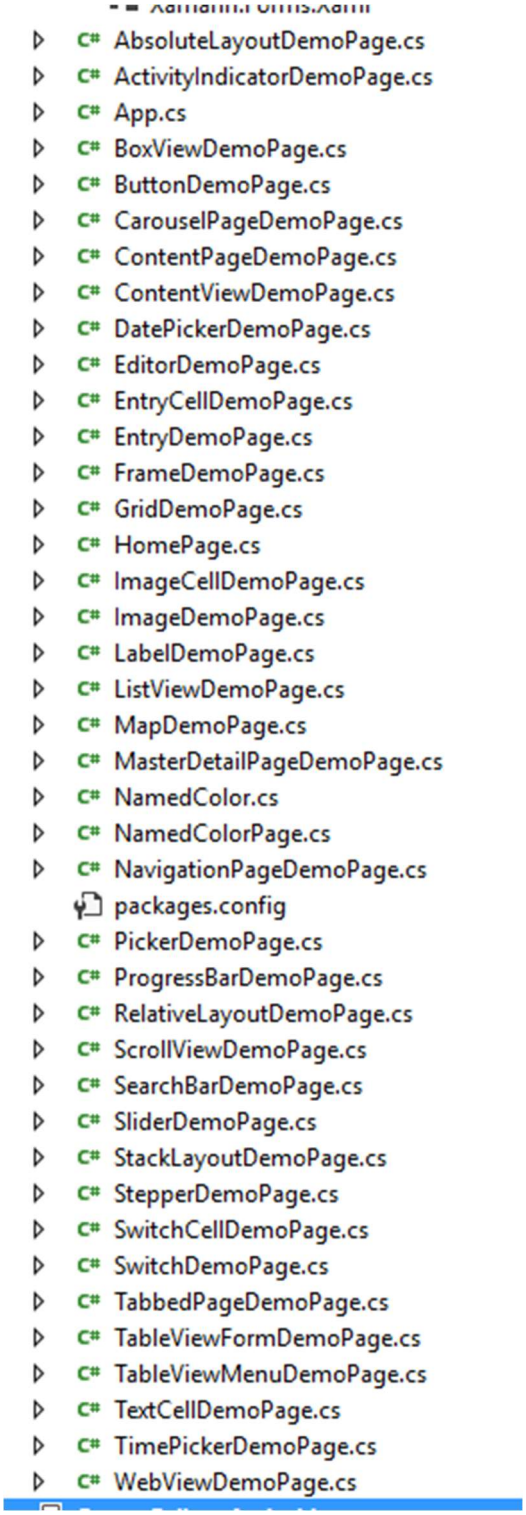
Mit folgendem Ergebnis:

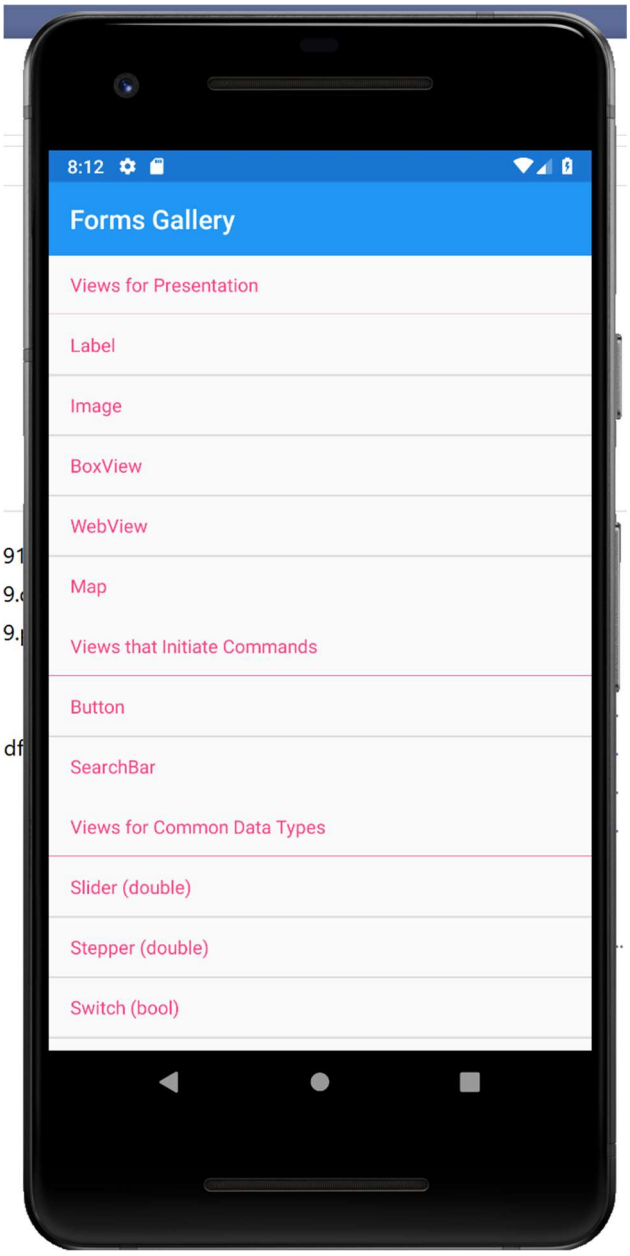
```
namespace FormsGalleryBay
{
    10 Verweise
    public partial class App : Application
    {
        3 Verweise
        public App ()
        {
            InitializeComponent();

            MainPage = new NavigationPage (new HomePage());
        }
```

UWP- Screens:



Klickt man ein Bespiel an, so wird eine Demoseite aufgerufen. Diese liegt dann als cs vor und man weiß, wie man sie in C# programmieren muss. Einfach super. Hier die Liste mit den Seiten, Daneben die Android- Version mit einem Pixel Pie- Emulator:


AbsoluteLayoutDemoPage.cs
ActivityIndicatorDemoPage.cs
App.cs
BoxViewDemoPage.cs
ButtonDemoPage.cs
CarouselPageDemoPage.cs
ContentPageDemoPage.cs
ContentViewDemoPage.cs
DatePickerDemoPage.cs
EditorDemoPage.cs
EntryCellDemoPage.cs
EntryDemoPage.cs
FrameDemoPage.cs
GridDemoPage.cs
HomePage.cs
ImageCellDemoPage.cs
ImageDemoPage.cs
LabelDemoPage.cs
ListViewDemoPage.cs
MapDemoPage.cs
MasterDetailPageDemoPage.cs
NamedColor.cs
NamedColorPage.cs
NavigationPageDemoPage.cs
packages.config
PickerDemoPage.cs
ProgressBarDemoPage.cs
RelativeLayoutDemoPage.cs
ScrollViewDemoPage.cs
SearchBarDemoPage.cs
SliderDemoPage.cs
StackLayoutDemoPage.cs
StepperDemoPage.cs
SwitchCellDemoPage.cs
SwitchDemoPage.cs
TabbedPageDemoPage.cs
TableViewFormDemoPage.cs
TableViewMenuDemoPage.cs
TextCellDemoPage.cs
TimePickerDemoPage.cs
WebViewDemoPage.cs


8:12
Forms Gallery
Views for Presentation
Label
Image
BoxView
WebView
Map
Views that Initiate Commands
Button
SearchBar
Views for Common Data Types
Slider (double)
Stepper (double)
Switch (bool)

UI von Xamarin Forms

Hat man sich schön an die Toolbox von der normalen Windows- Programmierung gewöhnt und die pixelweise Gestaltung der Oberflächen lieben gelernt, so folgt jetzt ein ernüchternder Schock. Es gibt keinen Designer mehr, keine Toolbox und das Oberflächen – UI (User-Interface) - Konzept folgt komplett einer total anderen Philosophie. Das muss man jetzt alles neu lernen. Übrigens ist der Bezeichner UI (User Interface) einer von vielen: HMI= Human-Machine Interface, GUI=Grafical User Interface, MMI = Mensch-Maschine Interface oder BO= Bedienoberfläche ……

Aber das ist die Zukunft!!!

Erstens: die Touch- Bedienung hat sich immer mehr durchgesetzt, da die Hardware billig geworden und die Bedienung einfach intuitiver ist. Die Mausbedienung, die 30 Jahre lang dominierte, wird sterben. In 10 Jahren gibt es nur noch Touch- Bildschirme, Tablets und Phones. Und es wird ja auch durch Win IoT die Bedienung vom Raspberry mit dieser Oberfläche möglich. Es gibt einfach alte Controls (z.B. die NumericUpDown), die sich nur mit Maus, aber nicht mit Touch bedienen lassen.

Zweitens: Und wenn man überhaupt Apps programmieren will, dann muss man sich mit dieser neuen Philosophie einfach anfreunden, da Android, IOS und UWP einfach auf diesem Prinzip basieren. Da die Bildschirme aller dieser Geräte völlig unterschiedlich sein können und die Pixel- Zahlen gewaltig variieren, hat eine pixelweise Gestaltung einfach keinen Sinn. Die Gestaltung entspricht einfach mehr den bekannten Internetseiten (Html, CMS etc.).

Die UI- Philosophie

Im Buch von Petzold hat er die Struktur folgendermaßen bildhaft beschrieben: Zitat Seite 1020:

> If you think of a Xamarin.Forms application as a building, then you construct this building from bricks that take the form of views and elements. You arrange them into walls using layout classes, and then form them into rooms with `ContentPage`, with passages from room to room made possible with navigational functions structured around `NavigationPage`.

Also die Views und Elements sind die Steine eines Hauses, die zusammen werden die Wände mit den verschiedenen Layouts. Diese zusammen werden dann mit ContentPage die Räume und mit NavigationPage kann man dann von Raum zu Raum navigieren. Alle diese Elemente lassen sich in Ruhe in dem Projekt „FormsGalleryBay“ kennen lernen. Und dieses wollen wir im Folgenden tun.

Die folgende Struktur in der Darstellung von FormsGallery ist ganz hilfreich:

UI-Übersicht

Views zum Präsentieren

- Label
- Image
- BoxView
- WebView

Views zum Auslösen von Ereignissen

- Button
- SearchBar

Views zum Ändern von Daten

- Slider

- Stepper
- Switch
- DatePicker
- TimePicker

Views zur Eingabe von Text

- Entry
- Editor

Views zur Anzeige von Aktivität

- ActivityIndicator
- ProgressBar

Views zur Anzeige von Collections (Listen)

- Picker
- ListView
- TableView für ein Menü
- TableView für eine Seite

Zellen

- TextCell
- ImageCell
- SwitchCell
- EntrryCell

Layouts mit einfachem Inhalt

- ContentView
- Frame
- ScrollView

Layouts mit mehreren Views (Children)

- StackLayout
- AbsolutLayout
- Grid
- RelativeLayout

Pages

- ContentPage
- NavigationPage
- MasterDetailPage
- TabbedPage
- CarouselPage

Für alle diese Elemente / Komponenten findet man in FormsGallery ein Beispiel und den entsprechenden C#- Code.

Dazu wird jetzt kurz erläutert, wie FormsGallery aufgebaut ist, damit man an die einzelnen Informationen gut herankommt: Es ist insgesamt eine „NavigationPage“:

In App.xaml.cs erfolgt der Aufruf von HomePage ja so:

```
MainPage = new NavigationPage(new HomePage());
```

Die HomePage ist als TableView aufgebaut, (this.Content = new TableView), dort gibt es in der Root einige TableSections (new TableSection("Views for Presentation"), in diesen Sections dann TextCell für die einzelnen Elemente, die angeklickt /getouched werden können.

Der Aufruf der Seiten über „navigateCommand“ ist für mich momentan sehr strange und nicht nachvollziehbar. Aber er funktioniert ja tatsächlich. In den dann aufgerufenen einzelnen Seiten kann man dann den SourceCode zur Benutzung erkennen. Z.B. LabelDemo:

Man braucht diese Art der Navigation aber nicht, da aus meiner Sicht alles mit MasterDetail einfach gesteuert werden kann (mein Favorit siehe weiter unten)

LabelDemoPage

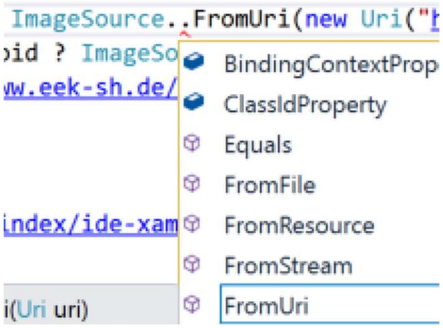
Öffnet man die Seite **LabelDemoPage**, so ist diese als ContentPage aufgebaut, es werden zwei Label deklariert und definiert, die dann in einem StackLayout als Children genannt werden. Das kennt man nun schon allmählich.

ImageDemoPage

In **ImageDemoPage** kommt jetzt das Image dazu, der Link zeigt ein Bild auf meiner Homepage.

```
Image image = new Image
            {
                Source = ImageSource.FromUri(new Uri("http://www.joergbayerlein.de/wp-content/uploads/2015/05/joerg.jpg")),
                VerticalOptions = LayoutOptions.CenterAndExpand
            };
```

In Eigenschaft Source gibt man die Quelle an, entweder wie im Text FromUri oder FromResource, wobei da der Dateizugriff weiter unten noch erarbeitet wird.



BoxViewDemoPage

In **BoxViewDemoPage** kommt die Box dazu:

```
BoxView boxView = new BoxView
  {
      Color = Color.Accent,
      WidthRequest = 150,
      HeightRequest = 150,
      HorizontalOptions = LayoutOptions.Center,
      VerticalOptions = LayoutOptions.CenterAndExpand
  };
```

WebViewDemoPage

In der **WebViewDemoPage** wird diese View so definiert:

```
WebView webView = new WebView
  {
      Source = new UrlWebViewSource
      {
          Url = "https://www.th-luebeck.de",
      },
      VerticalOptions = LayoutOptions.FillAndExpand
  };
```

ButtonDemoPage

In **ButtonDemoPage** wird wie schon im Reuter- Beispiel ein Button mit zwei Label in einem Stacklayout gesammelt, Click- Event wie schon erläutert. Die Formatierung der Ausgabe im Label im Click-Event ist in für mich etwas schwer lesbaren Form:

```
clickTotal += 1;
label.Text = String.Format("{0} button click{1}",
                           clickTotal, clickTotal == 1 ? "" : "s");
```

Man kann es so machen, das ist die Art, wie in speicherbegrenzten Minisystemen Bytes gespart werden. Besser lesbar ist meine Version:

```
clickTotal += 1;
label.Text = clickTotal.ToString() + " button click";
if (clickTotal > 1) label.Text += "s";
```

SearchBarDemoPage

In **SearchBarDemoPage** wird eine SearchBar mit Ereignis definiert:

```
SearchBar searchBar = new SearchBar
 {
     Placeholder = "Xamarin.Forms Property",
 };
 searchBar.SearchButtonPressed += OnSearchBarButtonPressed;
```

Im StackLayout wird zusätzlich zu dem bisher Bekannten unter Children ein ScrollView hinzugefügt, so dass der Inhalt unter ScrollView (also die Ergebnisausgabe) im Falle eines längeren Textes gescrollt werden kann:

```
Children =
 {
     header,
     searchBar,
     new ScrollView
     {
         Content = resultsLabel,
         VerticalOptions = LayoutOptions.FillAndExpand
     }
 }
```

Im Beispieltext der Aktion selbst sind für mich neu die Klassen `Assembly`, `var list = new List<Tuple<Type, Type>>();` und vor allen Dingen das `Tuple`.

Was auch immer, gibt man z.B. „Text“ ein so ist auf UWP- Emulator folgendes Ergebnis zu sehen (ich habe gegenüber dem ursprünglichen Text ein „\n\r“ hinzugefügt:

Es werden also alle Eigenschaften aller Xamarin.Forms – Komponenten durchsucht nach der Eigenschaft im Suchtext und aufgelistet mit Angabe des Typs.

Der ursprünglich Text ging so:

```
            foreach (Type type in xamarinFormsAssembly.ExportedTypes)
             {
                 TypeInfo typeInfo = type.GetTypeInfo();

                 // Public types only.
                 if (typeInfo.IsPublic)
                 {
                     // Loop through the properties.
                     foreach (PropertyInfo property in typeInfo.DeclaredProperties)
                     {
                         // Check for a match
                         if (property.Name.Equals(searchText))
                         {
                             // Add it to the list.
                             list.Add(Tuple.Create<Type, Type>(type,
property.PropertyType));
                         }
                     }
                 }
             }
```

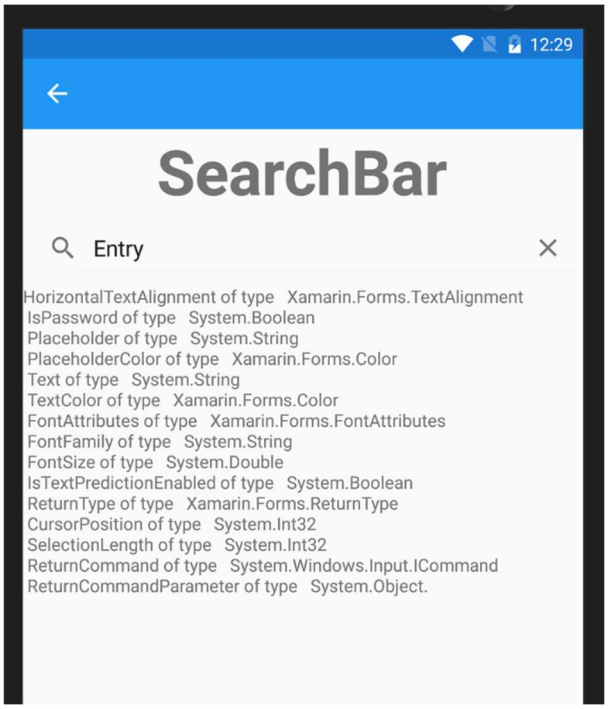
Ich habe ihn etwas umgewandelt, so dass man sich nun von type in Searchtext alle public- Properties anzeigen kann. Lässt man dann noch in der if die Abfrage nach searchtext weg, so werden alle type mit allen Properties aufgelistet. Gibt eine gute Übersicht über die Möglichkeiten, die man hat.

Mein neuer Code: Allerdings musste ich alle Deklarationen von Tuple ändern in `Tuple<String, Type>`

```
            foreach (Type type in xamarinFormsAssembly.ExportedTypes)
              {
                  TypeInfo typeInfo = type.GetTypeInfo();
```

```
                // Public types only.
                if (typeInfo.IsPublic && type.Name.Equals(searchText))
                {
                    // Loop through the properties.
                    foreach (PropertyInfo property in typeInfo.DeclaredProperties)
                    {
                            // Add it to the list.
                            list.Add(Tuple.Create<String, Type>(property.Name,
property.PropertyType));

                    }
                }
            }
```

Ergebnis z.B. für Entry:



Hier lernt man auch noch einmal den ScrollView kennen. Das Suchergebnis in Content wird in einen ScrollView gepackt, so dass bei einem Text größer als Display mit „Wischen“ gescrollt werden kann.

Sehr kompliziert scheint die dann folgende Suchaktion in der ButtonPressed – funktion zu sein. Er durchsucht die „Assembly“, dort die „ExportedTypes“ und speichert diese in einer dynamischen Liste von Tuple, die mir bis dato auch noch nicht bekannt waren. Die Klasse Tuple wird genauer in https://developer.xamarin.com/api/type/System.Tuple/ beschrieben. Das hat aber alles nichts mit UI zu tun.

SliderDemoPage

Das nächste Element ist die **SliderDemoPage.** Hinzufügen mit:

```
            Slider slider = new Slider
             {
                 Minimum = 0,
                 Maximum = 100,
                 VerticalOptions = LayoutOptions.CenterAndExpand
             };
             slider.ValueChanged += OnSliderValueChanged;
```

Hier wird das ValueChanged – Ereignis benutzt, um dort den Slider- Value, der in den EventArgs e übergeben wird, in einem Label anzuzeigen:

```
void OnSliderValueChanged(object sender, ValueChangedEventArgs e)
  {
      label.Text = String.Format("Slider value is {0:F1}", e.NewValue);
  }
```

Der Value wird als Typ double übergeben.

StepperDemoPage

In der **StepperDemoPage** wird der „Stepper“ vorgestellt:

```
Stepper stepper = new Stepper
 {
     Minimum = 0,
     Maximum = 10,
     Increment = 0.1,
     HorizontalOptions = LayoutOptions.Center,
     VerticalOptions = LayoutOptions.CenterAndExpand
 };
 stepper.ValueChanged += OnStepperValueChanged;
```

SwitchDemoPage

In der **SwitchDemoPage** wird ein Switch hinzugefügt:

```
Switch switcher = new Switch
{
    HorizontalOptions = LayoutOptions.Center,
    VerticalOptions = LayoutOptions.CenterAndExpand
};
switcher.Toggled += switcher_Toggled;
```

Funktion ist wie eine CheckBox. Die Schalteigenschaft vom Typ bool steht in Value.

DatePickerDemoPage

In der **DatePickerDemoPage** wird die Datumverstellung aufgerufen:

```
DatePicker datePicker = new DatePicker
  {
      Format = "D",
      VerticalOptions = LayoutOptions.CenterAndExpand
  };
```

Es gibt den Event DateSelected:

```
datePicker.DateSelected += DatePicker_DateSelected;
```

in der Function z.B.:

```
private void DatePicker_DateSelected(object sender, DateChangedEventArgs e)
 {
     date.Text = e.NewDate.Date.DayOfWeek.ToString();
 }
```

kann man den Wochentag eines jeden Datums sich ausgeben lassen.

Link zur Formatanweisung des DatePicker:
https://developer.xamarin.com/api/property/Xamarin.Forms.DatePicker.Format/

TimePickerDemoPage

Desgleichen TimePicker in der **TimePickerDemoPage:**

```
TimePicker timePicker = new TimePicker
 {
     Format = "T",
     VerticalOptions = LayoutOptions.CenterAndExpand
 };
```

Hier wollen wir wieder einen Event hinzufügen:

```
timePicker.PropertyChanged += TimePicker_PropertyChanged;
```

mit der Funktion:

```
       private void TimePicker_PropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
        {
            GepZeit.Text = timePicker.Time.ToString();
        }
```

Und es wird die eingestellte Uhrzeit angezeigt.

Der Format – Ausdruck entspricht dem der bekannten DateTime.toString() und kann hier nachgesehen werden:

https://docs.microsoft.com/en-us/dotnet/api/system.datetime.tostring?view=netframework-4.7.2

TimePicker: https://developer.xamarin.com/api/type/Xamarin.Forms.TimePicker/

EntryDemoPage

Wie schon in der Reuter-App benutzt, wie TextBox einzeilig:

```
new Entry
 {
     Keyboard = Keyboard.Email,
     Placeholder = "Enter email address",
     VerticalOptions = LayoutOptions.CenterAndExpand
 },
```

Wenn noch nichts eingegeben oder der Inhalt gelöscht wird, erscheint der Placeholder, Mit Keyboard kann man das auf dem Touch erscheinende Keyboard wählen, das passend für die Eingabe ist, also

```
Keyboard = Keyboard.Email,
```

oder

```
Keyboard = Keyboard.Text,
```

wenn Passworteingabe, dann `IsPassword = true,`

oder wenn Zahlen:

```
Keyboard = Keyboard.Telephone,
```

Noch weitere Keyboards sind: .Default, .Numeric, .Chat, .Plain oder .Url. Es werden passend zur Eingabe in den Tastaturen hilfreiche Tasten angeboten. In Chat sieht man z.B. die Zeile mit den Wortvorschlägen, bei URL wird Taste .COM angeboten etc.

EditorDemoPage

Mehrzeiliger Text möglich. Wenn Keyboard geschlossen, wird der Completed- Event ausgelöst.

```
Editor editor = new Editor
 {
     VerticalOptions = LayoutOptions.FillAndExpand
 };
 editor.Completed += Editor_Completed;
```

dann

```
private void Editor_Completed(object sender, EventArgs e)
  {
      header.Text="Completed";
  }
```

Auch hier kann ein Keyboard gewählt werden. Default ist .Chat.

Activity - ActivityIndicatorDemoPage

```
activityIndicator = new ActivityIndicator
 {
     Color = Device.RuntimePlatform == Device.iOS ? Color.Black :
Color.Default,
     IsRunning = true,
     VerticalOptions = LayoutOptions.CenterAndExpand
 };
```

Die Eigenschaft IsRunning kann gelesen oder beschrieben werden. Auf der Seite habe ich einen Button ergänzt, der diese Eigenschaft ein- und ausschaltet. Die Farbe wird plattformspezifisch gewählt.

ProgressBarDemoPage

```
progressBar = new ProgressBar
 {
     VerticalOptions = LayoutOptions.CenterAndExpand
 };
```

Die Eigenschaft Progress geht von 0 bis 1 und ist vom Typ double. Auf dieser Demo- Seite wird ein Timer benutzt, der alle 100 ms die function `TimerCallback` aufruft.

```
Device.StartTimer(TimeSpan.FromSeconds(0.1), TimerCallback);
```

Dort wird der Progress hochgezählt. Interessant dort ist die Abfrage, ob das Fenster aktiv ist. Die Methoden OnAppearing und OnDisappearing werden mit sich selbst überschrieben plus einer Steuerung einer bool- Variablen, die meinem Programm mitteilen kann, ob dieses Fenster grade aktiv ist. Bei dieser demo ist das nicht so interessant, da die Seite ja bei jedem Aufruf neu erzeugt wird und die ProgressBar immer wieder von vorn startet.

Display Collections → PickerDemoPage

Das ist die Funktion der ComboBox aus Windows- Zeiten. Eine Pfeiltaste oder das Klicken auf den gewählten Text öffnet die Auswahlmöglichkeiten, also die Items.

Hinzufügen Picker mit:

```
Picker picker = new Picker
  {
      Title = "Color",
      VerticalOptions = LayoutOptions.CenterAndExpand
  };
```

Items ist eine Stringliste, es können nur strings hinzugefügt werden mit

```
picker.Items.Add(colorName);
```

In dieser Demo wird eine Klasse „Dictionary“ benutzt. Kannte ich auch noch nicht. Dort werden paarweise Objekte einander zugeordnet aufgelistet. Im Beispiel werden strings Farben zugeordnet. Die ersten Begriffe sind unter .Keys gelistet, die zweiten Eintragungen unter .Values. Aber die zweiten Werte können auch so erreicht werden:

```
string colorName = picker.Items[picker.SelectedIndex];
boxView.Color = nameToColor[colorName];
```

Im Index von NameToColor steht der string!!!

Im Beispiel wird außerdem wieder der Lambda- Operator benutzt bei der Definition des Ereignisses:

```
picker.SelectedIndexChanged += (sender, args) =>
     {
         … Ereignisscode
     };
```

Wer es denn so mag, mag es so tun.

ListViewDemoPage

ListView ist so etwas wie die bekannte ListBox, jedoch mit deutlich mehr Möglichkeiten. Die Demo ist für mich sehr schwer nachvollziehbar. Ich habs versucht: Dort werden verwendet:

Liste mit Namen „people“ mit vorher definierter Klasse „Person“, die drei Properties enthält: `Name,Birthday und FavoriteColor.`
Soweit OK.

Diese wird vorbelegt mit 27 Namen, Geburtstagen und Lieblingsfarben, so dass man eine beispielhafte Liste bekommt.

Die ListView:

```
ListView listView = new ListView
 {
     // Source of data items.
     ItemsSource = people,
```

Nur wenn man nur so die Source festlegt, wird Müll angezeigt. Man benötigt eine Vorlage für jedes Element, also ein „Template“, das mit der Klasse „`DataTemplate`“ erzeugt wird. In diesem Template werden zunächst zwei Label und eine BoxView deklariert und die Inhalte mit SetBinding an die Source gekoppelt. So wird z.B. mit

```
nameLabel.SetBinding(Label.TextProperty, "Name");
```

Der Text des Labels („TextProperty“) an das Element „Name“ in der Liste gebunden. Dann wird mit return ein ViewCell zurückgegeben, in der dann horizontal nebeneinander in einem StackLayout die Box und daneben die beiden Label ausgegeben werden. Die beiden Label sind dann in einem neuen StackLayout (Standard ist vertikale Orientierung) neben die Box platziert.

An solch eine Syntax muss ich mich erst gewöhnen. Ich habe einfach mal alles stak vereinfacht. Aus der Liste wird eine einfache String- Liste:

```
List<string> people = new List<string>
 {
     "Abigail","Bob","Cathy","David","Eugenie","Freddie","Greta",
"Harold","Irene","Jonathan","Kathy","Larry","Monica","Nick","Olive",
```

```
"Pendleton","Queenie","Rob","Sally","Timothy","Uma","Victor","Wendy","Xavier",
"Yvonne","Zachary", "Ben","John","Tom","Bert","Kily","Tini"
            };
```

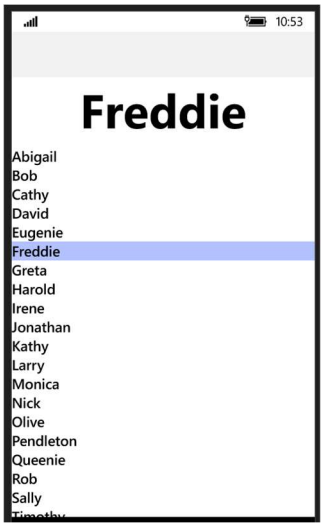
Dann funktioniert eine einfache Deklaration von

```
            ListView listView = new ListView
            {
                // Source of data items.
                ItemsSource = people

            };
```

Ohne DataTemplate bekommt man seine ListBox sogar mit automatischem Scrolleffekt zu sehen. Ich habe dann noch das Ereignis ItemSelected hinzugefügt und dort diesen Code ausgeführt:

```
            header.Text = e.SelectedItem.ToString();
```

Ergebnis:



Im Originaltext folgendes geändert:

Vor // Build the page

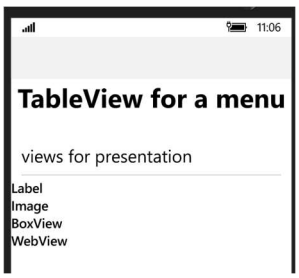
```
listView.ItemSelected += SelItem;
```

Dann außerhalb Konstruktor:

```
private void SelItem(object
sender,SelectedItemChangedEventArgs e)
        {
            Person p = (Person)e.SelectedItem;
            header.Text = p.Name;
        }
```

TableViewMenuDemoPage

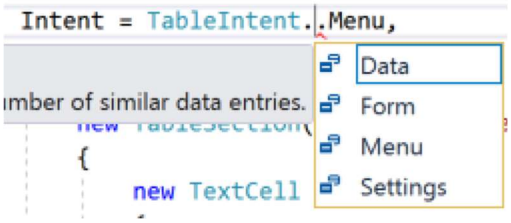
Hier wird eine TableView dazu benutzt, weitere Seiten aufzurufen. Genau dies wird in der Hauptanwendung von FormsGallery in der Datei HomePage.cs ja auch benutzt.



Klickt man auf Label, wird die schon bekannte LabelDemo- Seite aufgerufen usw.

Fügt man eine TableView hinzu, so hat man mit der Eigenschaft „Intent“ vier Möglichkeiten, die Funktion zu bestimmen: Data, Form, Menu oder Settings. Hier wird das Menü demonstriert.

```
        TableView tableView = new TableView
            {
                Intent = TableIntent.Menu,
```



Jetzt folgt eine Tabellenstruktur in root / TableSection und einzelnen TextCells oder anderen Zellen.

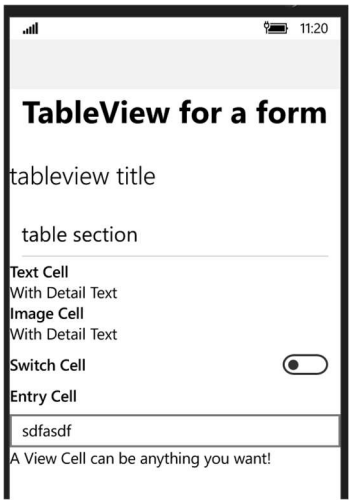
```
Root = new TableRoot
 {
     new TableSection("Views for Presentation")
     {
         new TextCell
         {
             Text = "Label",
```

Der Aufruf einer Seite wird mit einer TextCell möglich, da diese die Eigenschaft Command kennt.

```
Command = new Command(async () =>
    await Navigation.PushAsync(new LabelDemoPage()))
```

Wie gesagt, auch ich muss mich an eine solche Syntax gewöhnen. Bemerkt werden sollte nur noch, dass diese Seiten immer neu erzeugt werden und anschließend wieder verworfen werden.

TableViewFormDemoPage



Hier wird in der tableView der Intent auf

```
Intent = TableIntent.Form,
```

gestellt. Struktur der Tabelle wie oben, nur in einer Section können jetzt verschiedene Zellarten wie ImageCell oder SwitchCell hinzugefügt werden. Bei ImageCell habe ich jetzt wieder die Source geändert auf ein Bild auf meiner Seite. Zu den Zellen siehe nächste Kapitel

http://www.joergbayerlein.de/wp-content/uploads/2015/05/joerg.jpg

TextCellDemoPage

Viele Formatierungsmöglichkeiten hat man hier nicht, auch keine Layoutoptions.

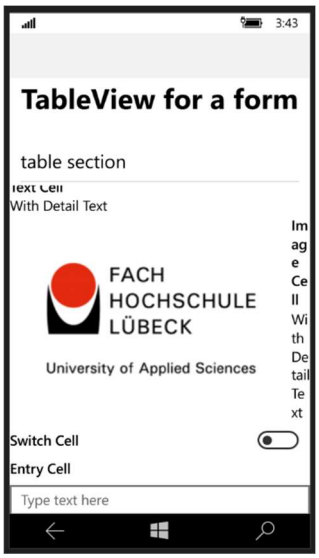
Mit diesem Text erhält man diese Reaktion:

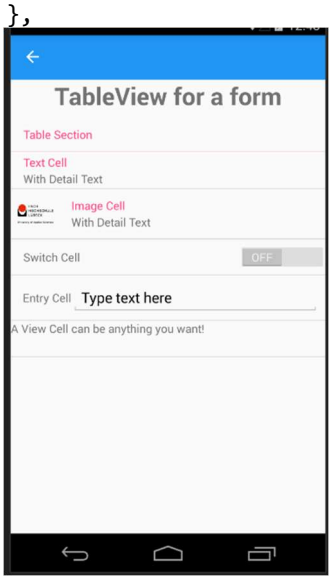
```
new TextCell
 {
     Text = "This is a TextCell",
     Detail = "This is some detail text",
     TextColor=Color.Red,
     DetailColor=Color.Green,
 }
```

ImageCellDemoPage

Siehe TableViewFormDemoPage, dort:

```
                    new ImageCell
                    {
                        // Some differences with loading images in initial
release.
                        ImageSource = ImageSource.FromUri(new Uri("http://www.eek-
sh.de/assets/images/8/fh-luebeck-logo-15fc7ed8.png")),
                        Text = "Image Cell",
                        Detail = "With Detail Text"
                    },
```





UWP Bild zu groß Android: Bild zu klein

SwitchCellDemoPage

Siehe TableViewFormDemoPage, dort benutzt, nichts Neues.

EntryCellDemoPage

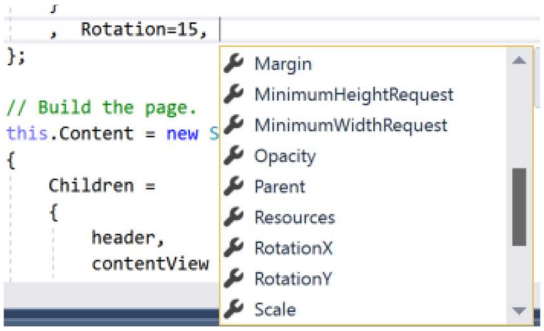
Siehe TableViewFormDemoPage, dort benutzt, nichts Neues.

Single Content Layout → ContentViewDemoPage

Bei einem ContentView kann man z.B. einem Label einen Hintergrund zuweisen oder diverse andere Dinge wie z.B. Rotation damit anstellen. Eine leichte Veränderung der Demo-Page sieht so aus:

```
                    TextColor = Color.Red
                }
                , Rotation=15
```

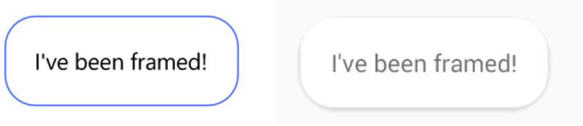
Man bekommt die Liste von Möglichkeiten von Eigenschaften ja bekanntlich mit dem IntelliSense- Vorschlagsfenster. Wenn man z.B. hinter Rotation=15 ein Komma setzt und ein Leerzeichen einfügt sieht man die möglichen Eigenschaften aufgelistet:



FrameDemoPage

```
Frame frame = new Frame
 {
     OutlineColor = Color.Accent,
     HasShadow = true,
     HorizontalOptions = LayoutOptions.Center,
     VerticalOptions = LayoutOptions.CenterAndExpand,
     Content = new Label
     {
         Text = "I've been framed!"
     }, CornerRadius=20
 };
```

Kann einen vielgestaltigen Rahmen um einen Content zeichnen mit Rahmenradius und Farbe variabel. Bei UWP kommt bei mir aber kein Schatten, bei Android schon.



UWP Android

ScrollViewDemoPage

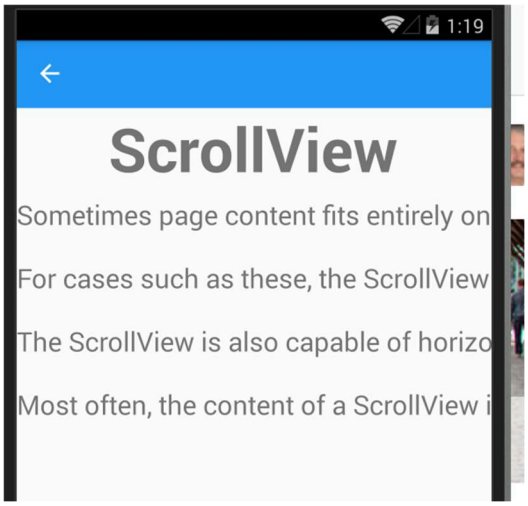
Wenn ein Content größer als eine Seite ist, sollte man ein ScrollView benutzen:

```
ScrollView scrollView = new ScrollView
 {
     VerticalOptions = LayoutOptions.FillAndExpand,
     Content = new Label
     {
         Text = "Sometimes page content fits entirely on "+
       …… langer text,

      }
  };
```

Normal ist ein vertikales Scrollen, es geht aber auch horizontal mit Eigenschaft

```
Orientation =ScrollOrientation.Horizontal,
```

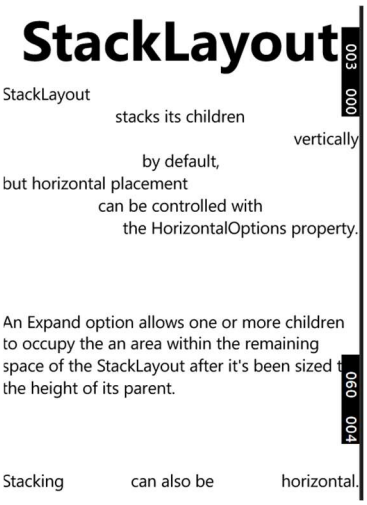
dann ist der Text oben vierzeilig und es sieht so aus:



Layouts mit mehreren Children → StackLayoutDemoPage

In Demo werden in einem StackLayout 8 Label mit verschiedenen HorizontalenLayout- Optionen untereinander gepackt.

Ein zweiter Stacklayout liegt im ersten, jetzt mit horizontaler Orientierung. Da bei UWP der untere Bereich verschwindet, habe ich ein Filler Label ergänzt.



AbsoluteLayoutDemoPage

Hiermit ist eine absolute Positionierung innerhalb eines Containers möglich. Deklaration:

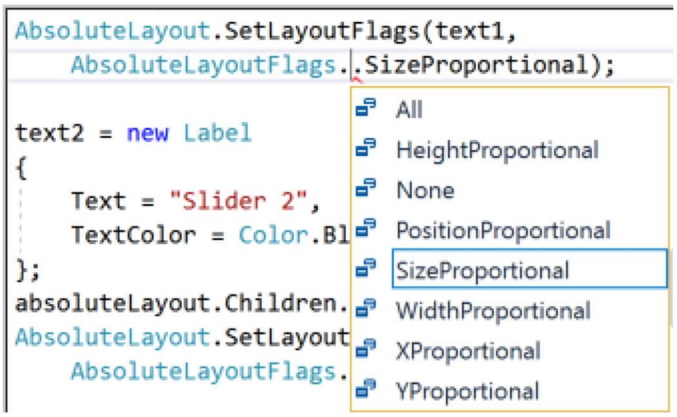
```
AbsoluteLayout absoluteLayout = new AbsoluteLayout
 {
     BackgroundColor = Color.Blue.WithLuminosity(0.9),
     VerticalOptions = LayoutOptions.FillAndExpand
 };
```

Hier wieder neu: Eine Farbeigenschaft „WithLuminosity(x)“, wobei x zwischen 0 und 1 liegen kann. Bei 00 wird jede Farbe ganz schwarz, bei 1 ganz weiß. Hier 0.9 heißt also ein sehr helles fast weißes blau.

Jetzt werden zwei Label zu diesem Layout als Children hinzugefügt mit der Add()- Methode:

```
absoluteLayout.Children.Add(text1);
```

Dann wird festgelegt, was an dem Label geändert werden soll. Die AbsolutLayoutFlags bestimmen, ob Position oder Größe verändert werden soll. Möglich sind:



Ich hab mal mit zwei Slidern die Demo verändert. Slider sl1 verändert die Größe mit „SizeProportional“ und Slider SL2 die `PositionProportional`

Ergebnis mit folgenden Slider- Ereignissen:

```
        private void Sl2_ValueChanged(object sender, ValueChangedEventArgs e)
         {
             AbsoluteLayout.SetLayoutBounds(text2,new Rectangle(e.NewValue/100.0,
e.NewValue/100.0, -1, -1));
             text2.Text = "x= "+ (e.NewValue / 100.0).ToString("f2") + "y=
"+(e.NewValue / 100.0).ToString("f3");
         }
```

Und

```
        private void Sl1_ValueChanged(object sender, ValueChangedEventArgs e)
         {
             AbsoluteLayout.SetLayoutBounds(text1, new Rectangle(0,0,e.NewValue/100.0,
e.NewValue/100.0));
             text1.Text = "x= " + (e.NewValue / 100.0).ToString("f2") + "y= " +
(e.NewValue / 100.0).ToString("f3");

         }
```

Gibt folgendes Ergebnis:

Beide Slider auf 20%, Bei der Größe ist immer Position 0,0, also oben Links und der Label auf 20% Größe gequetscht. Bei der Position ist oben links 0,0. Bei 0,2 ist die Startposition wie im Bild.

Label 1 hat gelbe Hintergrundfarbe. Der Startpunkt von Label 2 ist jetzt nicht 20%20%, sondern so dass bei 100 % der Label noch im Feld ist. Da wird die Länge des Label 2 also berücksichtigt.

GridDemoPage

Zunächst wird definiert, wieviel Zeilen (hier 4) und Spalten (hier 3) die Tabelle haben soll:

```
Grid grid = new Grid {
VerticalOptions = LayoutOptions.FillAndExpand,
```

```
RowDefinitions = {
new RowDefinition { Height = GridLength.Auto },
new RowDefinition { Height = GridLength.Auto },
new RowDefinition { Height = new GridLength (1, GridUnitType.Star) },
new RowDefinition { Height = new GridLength (100, GridUnitType.Absolute) }
   },
ColumnDefinitions = {
new ColumnDefinition { Width = GridLength.Auto },
new ColumnDefinition { Width = new GridLength (1, GridUnitType.Star) },
new ColumnDefinition { Width = new GridLength (100, GridUnitType.Absolute) }
   }
};
```

Es ist die Größe der Spalten und Zeilen auf drei Weisen zu bestimmen: Automatisch vom Inhalt abhängig, Über die Einheit Star und Absolut, beides aber geräteabhängig und keine Pixel. Star ergibt sich, nachdem die Spalten und Zeilen mit Absolut und Auto in der Größe festliegen.

Dann werden Views zu den Zellen mit der Add- Methode hinzugefügt:

```
grid.Children.Add (new Label {
       Text = "Autosized cell",
       TextColor = Color.White,
       BackgroundColor = Color.Blue
}, 0, 1);
```

Wobei die beiden Zahlen nach dem View die Nummer von Spalte / Zeile von oben links ab Null gezählt darstellen, also bei 0,1 ist es Spalte 0 und Zeile 1. Man kann jetzt auch einen View über mehrere Zeilen und Spalten legen. Syntax bei Add mit folgenden vier Zahlen:

Also bei den vier Zahlen a,b,c,d geht das Feldes über die Spalten a bis b-1 und über die Zeilen c bis d-1.

Bei der Überschrift:

```
grid.Children.Add (new Label {
       Text = "Grid",
       FontSize = 50,
       FontAttributes = FontAttributes.Bold,
       HorizontalOptions = LayoutOptions.Center
}, 0, 3, 0, 1);
```

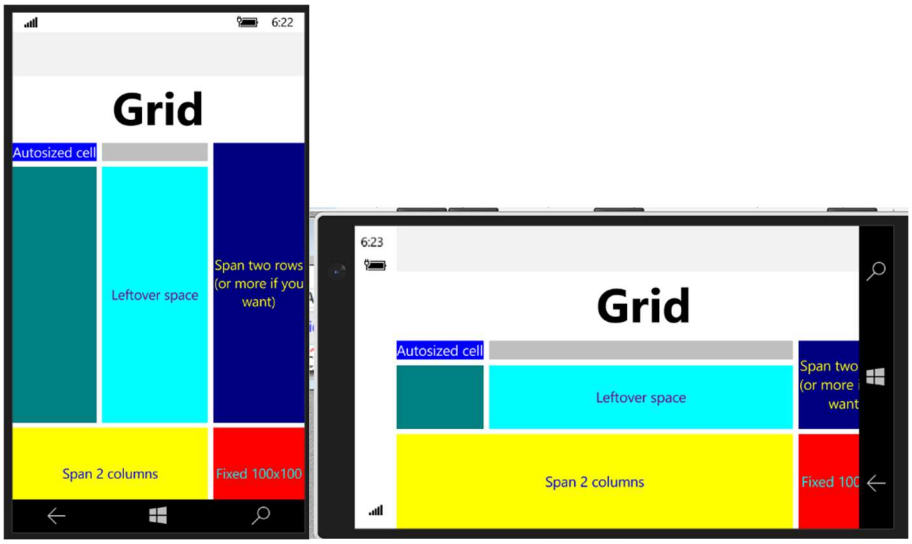
Also von Zeile 0 bis 2 und nur Spalte 0.

Bei dem blauen Feld („Span two rows“):

```
grid.Children.Add (new Label {
       Text = "Span two rows (or more if you want)",
       TextColor = Color.Yellow,
       BackgroundColor = Color.Navy,
       HorizontalTextAlignment = TextAlignment.Center,
       VerticalTextAlignment = TextAlignment.Center
}, 2, 3, 1, 3);
```

Nur Spalte 2 von Zeile 1 bis 2.

Hier die Screens beim UWP gedreht. Die mittlere Spalte ergibt sich, wenn linke und rechte festliegen, sie bekommt also den „Rest“.



RelativeLayoutDemoPage

Der Code ist grausam, wer braucht sowas. Interessant ist der einfache Seitenaufbau mal nicht mit StackLayout, sondern mit einem Grid mit einer Spalte und zwei Zeilen. Oben der Header, der Rest ist das Relative Layout.

```
                // Build the page.
        Grid grid = new Grid {
        RowDefinitions = {
               new RowDefinition { Height = GridLength.Auto },
               new RowDefinition { Height = new GridLength (1, GridUnitType.Star) }
               }
               };
        grid.Children.Add (header, 0, 0);
        grid.Children.Add (relativeLayout, 0, 1);

        Content = grid;
```

Pages → ContentPageDemoPage

Dies wird praktisch bei allen vorherigen Seiten benutzt und ist nichts Neues. Erst die Definition der

Views wie Label usw., dann in einem Layout dies in den Content mit den Children füllen:

```
    class ContentPageDemoPage : ContentPage
     {

……….
            this.Content = new StackLayout
             {
                 Children =
                 {
                     header,
                     label1,
                     label2,
                     label3
                 }
             };
```

NavigationPageDemoPage

Es gibt zwar eine solche Klasse „NavigationPage“, aber bei dieser Demo wird sie gar nicht benutzt, sondern die einfache ContentPage!!! Das liegt daran, dass die HomePage schon als NavigationPage deklariert worden ist. Dann kennt er den Befehl PushAsync, sonst nicht.

Egal, dort werden Buttons definiert und in den Click- functions werden mit Lambda- Operator die Seiten aufgerufen, Code:

```
button2.Clicked += async (sender, args) =>
    await Navigation.PushAsync(new ImageDemoPage());
```

Also nichts Neues.

Zum Abschluss der Beschreibung der Elemente in FormsGallery werden die nächsten drei Steuerungselemente besprochen. Sie benutzen alle drei eine neue Klasse namens NamedColor (siehe NamedColorPage.cs), in der einfach für eine Farbe ein String zugeordnet wird ähnlich wie in einer Dictionary- Klasse.

Dann gibt es eine Klasse NamedColorPage, mit der neue Seiten erzeugt werden können, die dann mit MasterDetail, TabbedPage oder CarouselPage aufgerufen werden können. Das macht das Verständnis dieser Seiten etwas schwierig. Am besten schaue man sich diese drei Seitenaufrufmethoden erst mal in Funktion an. Eine solche Farbseite besteht aus drei Labels, die die RGB- Anteile der gewählten Farbe von 0 bis 1 (0- 100%) anzeigen. Dann kann man ein BoxView mit genau dieser Farbe sehen und darunter werden die drei HSV- Anteile Hue (Farbwinkel oder Farbwert, 0 für Rot, 0.33 für Grün und 0.66 für Blau), Saturation (Farbsättigung: 0 für Grau, 1 für volle Sättigung) und Luminosity (Helligkeit, Weißanteil, Hellwert oder Dunkelstufe : 0 keine Helligkeit, 1 volle Helligkeit) angezeigt. In dem Link
https://developer.xamarin.com/api/type/Xamarin.Forms.Color/
sind alle definierten Farben mit RGB- Werten gelistet, es sind sehr viele (!) Es sind insgesamt 140 verschiedene englische Namen für Farben. In Wiki : https://de.wikipedia.org/wiki/HSV-Farbraum

Der Aufruf `NamedColorPage(false)` erwartet ein bool im Argument, mit dem festgelegt wird, ob eine große Überschrift mit Farbnamen dargestellt wird. Bei der TabbedPage wird er weggelassen (steht ja auf den Tabs drauf) , bei den beiden anderen dargestellt.

Der Seitenaufbau einer neuen Color- Seite ist wieder ganz einfach. Content ist ein StackLayout mit drei Label, dann boxview, dann noch ein StackLayout mit weiteren drei Label. Wenn im Aufruf ein true übergeben wird, wird der „bigLabel“ deklariert und mit

```
(this.Content as StackLayout).Children.Insert (0, bigLabel);
```

Hinzugefügt. So lernt man auch mal, dass so etwas möglich ist.

Die sechs Label werden mit eine Function `CreateLabel` erzeugt, das macht das Ganze dann halt etwas kompliziert. Ich hätte sechs Mal den gleichen Code reinkopiert.

Dazu wird das Schlüsselwort „Func“ benutzt. In der Hilfe sieht man dann diese Information:

```
// Function to create six Labels.
Func<string, string, Label> CreateLabel = (string source, string fmt) => {
    delegate TResult System.Func<in T1, in T2, out TResult>(T1 arg1, T2 arg2)
    To be added.

    T1 ist string
    T2 ist string
    TResult ist Label
                                                                    Label)),
                                                                    ));
    return label;
```

CreateLabel hat zwei Eingabeparameter als string und einen Rückgabewert vom Typ Label. Im ersten Parameter holt sich die Funktion die gewählte Farbe in Color und gibt die R-G-B- Werte aus, im zweiten Parameter wird ein Formatier- String übergeben: Ausgabe Float mit zwei Nachkommastellen.

Die Farbe wird jetzt in einem Binding übergeben.

➔ Aber dieses Beispiel taugt nicht für die einfache Erklärung dieser Mehrfachseitensteuerung. Es ist einfach zu kompliziertes Zeugs.

Deswegen hab ich folgendes neues Projekt (bei mir App2) zur einfachen Demonstration der TabbedPage und der CarouselPage zusammengebaut:

Siehe dazu das Dokument Kurs 3 UI mit Xamarin.Forms V20.pdf

Ende des zweiten Doppelblocks

Gezeichnet Prof. Dr. Bayerlein Dez. 2020