

Inhalt

Kurs 3 Apps mit Xamarin – User Interface Konzept	2
2. Teil des UI	2
Erste kleine eigene Seite ColorChanger	2
Seitennavigation.....	5
TabbedPageDemoPage	5
CarouselPageDemoPage	7
MasterDetailPageDemoPage	7
LayoutOptions	10
Image Größe und Positionsdemo.....	12
Data Binding	13
GestureRecognizers.....	13
Alert Box mit DisplayAlert	14
PageImageLayoutAspectohneScroll.cs.....	14
PageImageLayoutAspectScrollboth.cs	14
PageTransRotScaleAnchor.cs	15
PageDemoGestures.cs.....	15
PageGesturesSimple.cs	17

Kurs 3 Apps mit Xamarin – User Interface Konzept

2. Teil des UI

Es werden die folgenden Dokumente vorausgesetzt:

Kurs 2 UI mit Xamarin.Forms – V20.pdf

Erste kleine eigene Seite ColorChanger

Um die Erstellung eigener Seiten etwas zu üben beginnen wir mit einer kleinen Anwendung, die auch im Praktikum benutzt werden soll. Mit zweimal drei Slidern soll die Farbe einer Box und im Labor dann die Farbe einer WLAN- Lampe verstellt werden.

Im Anschluss daran werden die Seitennavigationen vorgestellt.

Aussehen im Emulator:



Öffnen neues Cross- Mobile App Projekt mit AppColorChangerV1, Leer, + UWP. Hinzufügen neuer Seite C#- Inhaltsseite, Name PageColChange.

Dort aus Codesnippets die einzelnen Teile einfügen oder per Hand hinzufügen:

Außerhalb Konstruktor:

```
Label lab;
```

```
BoxView box;
Button btnRGB, btnHSL;
Slider slR, slG, slB, slHue, slSat, slLum;
```

Innerhalb:

```
Label LabTitle = new Label
{
    Text = "Demo Page for Color Changing",
    FontSize = 20.0,
};
lab = new Label
{
    Text = "Top R--G--B, Bottom: Hue, Saturation, Luminosity",
    VerticalOptions = LayoutOptions.Start
};
btnRGB = new Button
{
    Text = "Change RGB- Values",
};
btnHSL = new Button
{
    Text = "Change HSL - Values",
};
slR = new Slider
{
    Maximum = 255,
    HorizontalOptions = LayoutOptions.Fill,
    VerticalOptions = LayoutOptions.Start,
};
slR.ValueChanged += SlR_ValueChanged;
slG = new Slider
{
    Maximum = 255,
    HorizontalOptions = LayoutOptions.Fill,
    VerticalOptions = LayoutOptions.Start
};
//sl2.ValueChanged += Sl2_ValueChanged;

slB = new Slider
{
    Maximum = 255,
    HorizontalOptions = LayoutOptions.Fill,
    VerticalOptions = LayoutOptions.Start
};
//sl3.ValueChanged += Sl3_ValueChanged;

slHue = new Slider
{
    Maximum = 255,
    HorizontalOptions = LayoutOptions.Fill,
    VerticalOptions = LayoutOptions.Start,
};
//sl4.ValueChanged += Sl4_ValueChanged;

slSat = new Slider
{
    Maximum = 255,
    HorizontalOptions = LayoutOptions.Fill,
    VerticalOptions = LayoutOptions.Start,
};
// sl5.ValueChanged += Sl5_ValueChanged;
```

```

s1Lum = new Slider
{
    Maximum = 255,
    HorizontalOptions = LayoutOptions.Fill,
    VerticalOptions = LayoutOptions.Start,
};

//s16.ValueChanged += S16_ValueChanged;

box = new BoxView
{
    VerticalOptions = LayoutOptions.FillAndExpand,
    //Aspect = Aspect.AspectFit,
    HorizontalOptions = LayoutOptions.Fill,
    BackgroundColor = Color.Gray,
};
Content = new StackLayout
{
    Children =
    {
        LabTitle, lab, btnRGB, s1R, s1G, s1B,
        box, btnHSL, s1Hue, s1Sat, s1Lum
    }
};
this.Padding = new Thickness(20, 20, 20, 20);

```

Aussehen ist jetzt OK, aber noch keine Funktion. Jetzt bei den Slidern mittels Tab- Taste Event-Funktionen erzeugen und bei den oberen ergänzen :

```
box.BackgroundColor = Color.FromRgb(s1R.Value / 255.0, s1G.Value / 255.0, s1B.Value / 255.0);
```

in den unteren:

```
box.BackgroundColor = Color.FromHsla(s1Hue.Value / 255.0, s1Sat.Value / 255.0, s1Lum.Value / 255);
```

Jetzt funktioniert schon alles, aber schöner wäre es, wenn bei RGB- Verstellung die HSL- Regler aktualisiert würden und umgekehrt. Das geht dann so: Es wird eine boolsche Variable ergänzt:

```
bool rgb;
```

In den beiden Buttons im Clickevent wird umgeschaltet:

```
Button RGB: rgb = true;
```

```
Button HSL: rgb = false;
```

in den RGB- Slidern:

```

if (rgb)
{
    s1Hue.Value = box.BackgroundColor.Hue * 255;
    s1Sat.Value = box.BackgroundColor.Saturation * 255;
    s1Lum.Value = box.BackgroundColor.Luminosity * 255;
    box.BackgroundColor = Color.FromRgb(s1R.Value / 255.0, s1G.Value /
255.0, s1B.Value / 255.0);
}

```

In den HSL- Slidern:

```

if (!rgb)
{
    slR.Value = box.BackgroundColor.R * 255;
    slG.Value = box.BackgroundColor.G * 255;
    slB.Value = box.BackgroundColor.B * 255;
    box.BackgroundColor = Color.FromHsla(slHue.Value / 255.0, slSat.Value
/ 255.0, slLum.Value / 255);
}

```

Fertig. Alle Werte RGB und HSL sind in Color double zwischen 0 und 1.

Seitennavigation

Deswegen habe ich folgendes neues Projekt (bei mir AppBayFinal2019) zur einfachen Demonstration der TabbedPage und der CarouselPage zusammengebaut:

TabbedPageDemoPage

1. Start mit Blank cross- platform Name: AppBayFinal2019.
2. Hinzufügen von einer C#- Seite Page1, diese wird die Main- Page
3. Hinzufügen von 3 weiteren C#- Seiten aus Appsimple.zip
4. Bild regcsharp.jpg auf Build: eingebettete Resource.
5. In App.xaml.cs ersetzen von MainPage = new App2.**Page1**();
6. In Page1 jetzt **public class Page1 : TabbedPage** statt ContentPage
7. Im Konstruktor von Page1 jetzt alles ersetzen durch:

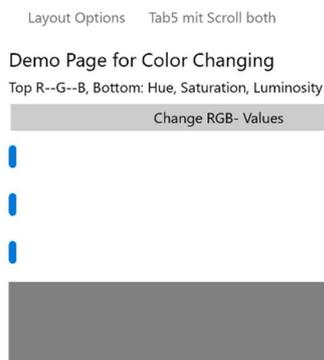
```

Children.Add(new ColChanger());
Children.Add(new PageLO());
Children.Add(new PageTrans());

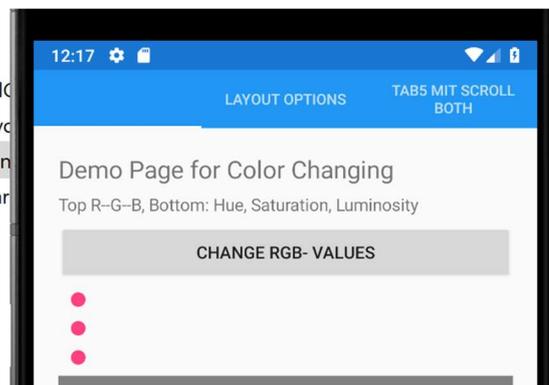
```

In allen Tab- Pages muss jetzt im Konstruktor ein Title gesetzt werden, der dann in den Tabs erscheint: **this.Title="Tab1-4"**; ebenfalls zum Erkennen der richtigen Seite in den Labels **"Dies ist Seite Tab1-4"**, dann sollte es funktionieren! ColChanger hat noch keinen Title!

UWP:



Android:



Man erkennt, dass bei UWP der Titel ganz ausgeschrieben wird mit Pfeilen, um nach links und rechts zu gehen, bei Android sind die Tabs oben fest und die Texte werden abgeschnitten.

Jetzt will ich noch die Datenübergabe zwischen den Seiten zeigen. Das funktioniert exakt genauso wie bei der Übergabe von zwei Fenstern in Windows-Forms. Die zwei Seiten -Dateien ergänzen:

PageBayReuter.cs
PageFetchData.cs

In der Steuerseite Page 1 ergänzen:

```
Children.Add(new PageBay());
Children.Add(new PageTab6());
```

Änderungen in Datei PageFetchData:

Man schafft sich eine Referenz auf Page1, also der Main-Page. Und im Konstruktor wird diese Referenz übergeben:

```
Label lab;
Page1 meinMain;
public PageTab1(Page1 m)
{
    meinMain = m;
```

Dann kann man in der But10.Clicked von der Reuterpage einen Wert auslesen und in Lab ausgeben mit:

```
private void But10_Clicked(object sender, EventArgs e)
{
    lab.Text=meinMain.pb.labOutT1.Text;
}
```

Dabei zuvor noch die Reuter- Page PageBayReuter als public deklarieren und den labOutT1 auch.

Also in Reuter: **public** Label labOutT1;
Und in Page1:

```
public PageBay pb;
public Page1()
{
    pb = new PageBay();
    Children.Add(new PageTab6(this));
    ...
    Children.Add(pb);
    ...
```

Dann Ergebnis:

The screenshot shows a mobile application interface with a title bar containing tabs: Tab1, Tab2, Tab3, Tab4, and Reuter- Ident. The main content area is titled 'App zur Reuter' and contains two input fields labeled 'Eingabe Tu:' and 'Eingabe Tg:'. The 'Eingabe Tu:' field contains the value '0123' and the 'Eingabe Tg:' field contains '0234'. Below these fields is a yellow button labeled 'Berechne Reut'. To the right of the input fields is a grey button labeled 'Hole Zahl T1 von Reuter- Page'. Below the input fields, the following text is displayed: 'T1= 108.786600', 'T2= 65.262600', and 'Td= 99.600000'. To the right of this text, the value 'T1= 108.786600' is displayed again, followed by the text 'und auf Tab1:'.

Bei iOS muss immer vorher in info.plist der Bundle- ID geändert werden auf com.bay....:

Bundle-ID:

com.bay.AppBayFinal2019

CarouselPageDemoPage

Man ersetzt einfach in der Main- Seite (bei mir Page1) oben die TabbedPage durch CarouselPage:

```
public class Page1 : CarouselPage
```

Der Rest ist identisch. Der einzige Unterschied zwischen beiden ist einfach, dass bei TabbedPage oben (oder bei IOS unten) die Tabs mit dem Seitentitel erscheinen, bei der CarouselPage nicht. Bei beiden kann man die Seiten hin- und herwischen. Bei UWP konnte man früher pausenlos nach rechts oder links wischen, erschien die letzte Seite in einer Richtung, folgte die erste von links und umgekehrt. Aber heute geht das nicht mehr, sondern wie bei Android: Bei Android kann man nur bis zum Ende der Seiten wischen, dann muss man wieder zurückwischen.

MasterDetailPageDemoPage

Hier ist die Steuerung etwas komplizierter, dafür die Bedienung später bei mehreren Seiten viel einfacher. Das Beispiel in FormsGallery kann man vergessen, zum Erlernen immer noch viel zu kompliziert. Auch dazu habe ich ein eigenes Beispiel erzeugt, auch wieder mit unseren schon existierenden fünf Seiten. Hauptunterschied zu den beiden obigen Steuerungen: Man hat eine Hauptseite mit den Auswahlelementen für die Detailseiten. Dort muss man eine Detailseite wählen und dann gibt es eine „Zurücktaste“ zur Hauptseite. Das macht nur Sinn, wenn man ganz viele Detailseiten hat und man sich mit den Tabs und Carousel zu Tode wischen würde. Die Auflistung mit Selektionsmöglichkeit geht über eine ListView.

Hinzufügen weiterer 7 Seiten aus AppRestFinal2019.zip

Also Änderung in Page1, der neuen Startseite, alle Pages vorher deklarieren :

```
public class Page1 : MasterDetailPage
{
    public PageBay pb;
    PageColChanger pcc;
    PageGesture pg;
    PageL0 plo;
    PageTab2 pt2;
    PageTab3 pt3;
    PageTab4 pt4;
    PageTab5 pt5;
    PageTab6 pt6;
    PageTab9 pt9;
    PageGestureSimple pgs;
    PageTrans ptr;
}
```

Dann im Konstruktor alle Seiten instanzieren, Label, StringArray und Listview erzeugen:

```
pb = new PageBay(); // PageBayReuter.cs
pt6 = new PageTab6(this); //PageFetchData.cs
pg = new PageGesture(); //PageDemoGestureslocalImage.cs
pt2 = new PageTab2(); //PageImageLayoutAspectScrollVert.cs
pt3 = new PageTab3(); //PageImageLayoutAspectohneScroll.cs
pt4 = new PageTab4(); //PageAlertBox.cs
pt5 = new PageTab5(); //PageImageLayoutAspectScrollboth.cs
pt9 = new PageTab9(); //PageColorChanger.cs
ptr = new PageTrans(); //PageTransRotScaleAnchor.cs
pcc = new PageColChanger(); // PageColChanger
pgs = new PageGestureSimple(); //PageGestureSimple
plo = new PageL0(); //PageLayoutOptions.cs
```

```

        Label header = new Label
    {
        Text = "MasterDetailPage",
        FontSize = 30,
        FontAttributes = FontAttributes.Bold,
        HorizontalOptions = LayoutOptions.Center
    };

    string[] pages =
    {
        "Reuter", "Fetch Data", "Gesture Demo", "vert scroll", "ohne
scroll", "Alert", "both scroll", "Pic Tap",
        "Trans etc", "Color Changer", "Layout options"
    };
    // Create ListView for the master page.
    ListView listView = new ListView
    {
        ItemsSource = pages
    };

    listView.ItemSelected += ListView_ItemSelected;

    this.Master = new ContentPage
    {
        Title = "Page List", // Title required!
        Content = new StackLayout
        {
            Children = {
                header,
                listView
            }
        }
    };
    //listView.SelectedItem = 1;
    this.Detail = pt2; // Startseite
    this.IsPresented = true;

```

Dann werden die Seiten Master und Detail festgelegt, voreingestellt ist pt2. Damit als erstes beim Starten die Masterseite gezeigt wird, wird IsPresented auf true, sonst startet das Programm mit Seite pt1.

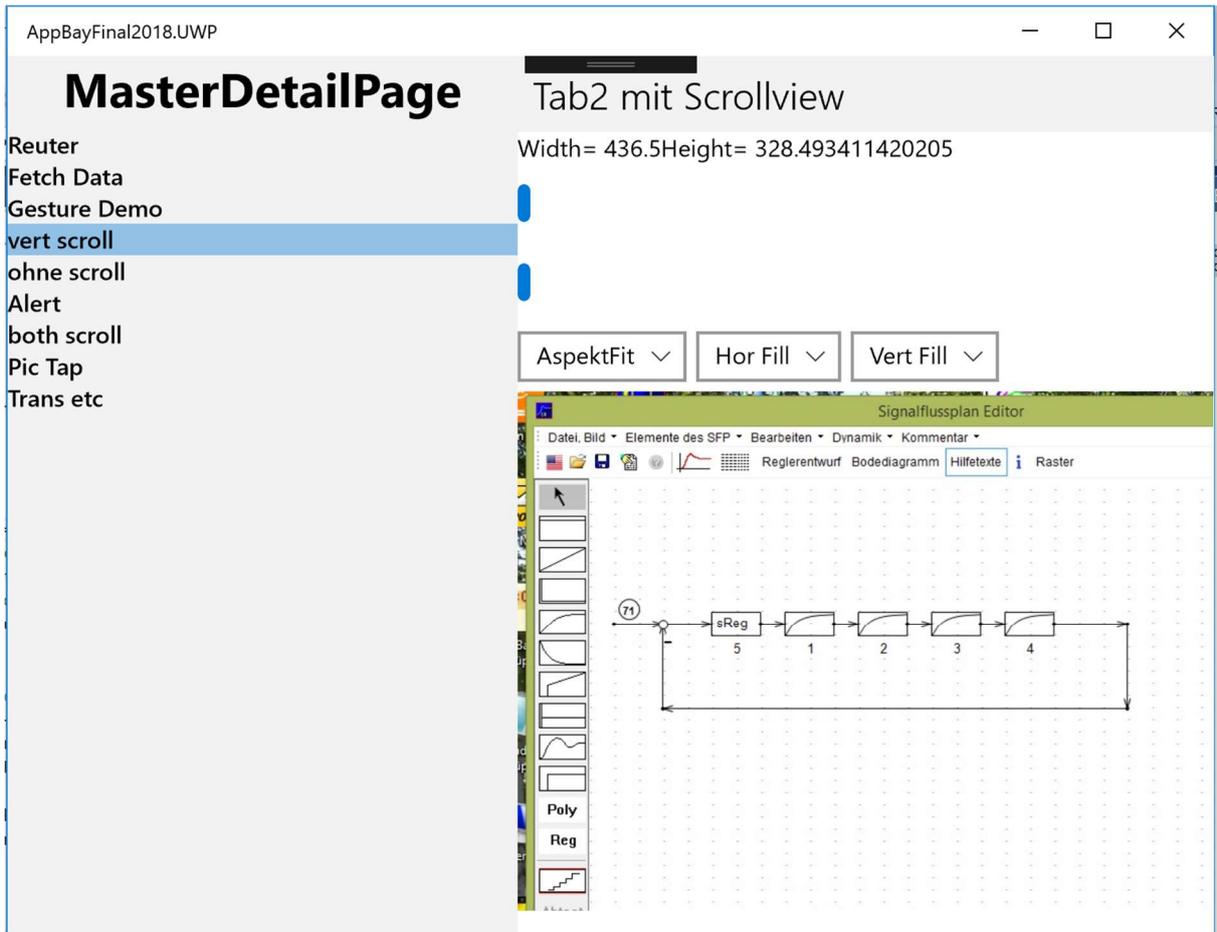
In dem Ereignis beim Selektieren:

```

private void ListView_ItemSelected(object sender, SelectedItemChangedEventArgs e)
{
    switch (e.SelectedItemIndex)
    {
        case 0: this.Detail = pb; break;
        case 1: this.Detail = pt6; break;
        case 2: this.Detail = pg; break;
        case 3: this.Detail = pt2; break;
        case 4: this.Detail = pt3; break;
        case 5: this.Detail = pt4; break;
        case 6: this.Detail = pt5; break;
        case 7: this.Detail = pt9; break;
        case 8: this.Detail = ptr; break;
        case 9: this.Detail = pcc; break;
        case 10: this.Detail = pgs; break;
        case 11: this.Detail = plo; break;
    }
    this.IsPresented = false;
}

```

Ergebnis UWP nach Selektieren des vierten Eintrags:

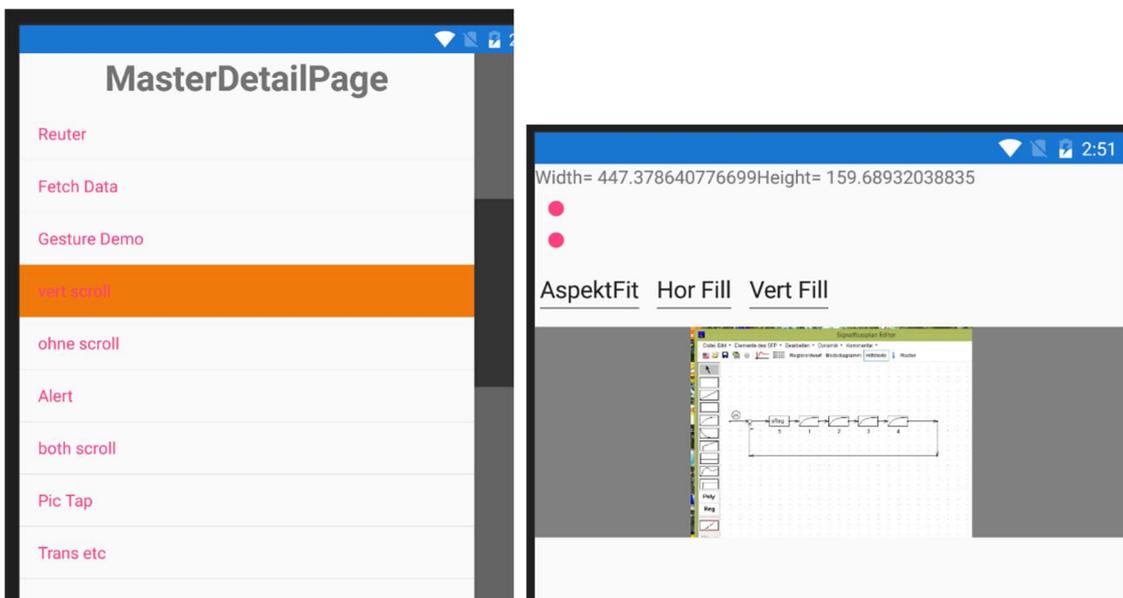


Auf dem lokalen Computer hat man viel Platz und die Masterseite bleibt immer sichtbar.

Das Ereignis ItemSelected hat den Nachteil, dass wenn man auf ein bereits selektiertes Item klickt, nichts passiert. Das ist doof. Besser den Event ItemTapped nehmen, dann geht es immer, also

```
listView.ItemTapped += ListView_ItemTapped;
```

Ergebnis Android, wenn vierter Eintrag selektiert wird:



Es gibt keinen Zurück- Button!! Zurück kommt man mit Wischen vom ganz linken Rand nach rechts.

Das war es mit den Elementen View / Layout / Page

Bitte zur Realisierung dieser folgenden Demonstrationen die Seiten AppBayFinal2019 in eine leere cross-App einbinden und die schon bisher öfter durchgeführten Aktionen zur Anpassung wiederholen:

1. Hinzufügen aller Seiten und jpg-Bild
2. In App.xaml.cs auf Page1 umschalten
3. Bild regCharp.jpg auf „Eingebettete Resource“ stellen

LayoutOptions

Link: <https://developer.xamarin.com/guides/xamarin-forms/user-interface/layouts/layout-options/>

Siehe Menüpunkt

Diese sind ja schon laufend benutzt worden und sollen hier etwas näher erläutert werden. Die Möglichkeiten sind:

.Center

.CenterAndExpand

.End

.EndAndExpand

.Fill

.FillAndExpand

.Start

.StartAndExpand

Die vier Alignment- Options Center, End, Fill und Start können mit der „Expansion“ kombiniert werden. Die AndExpand- Zusätze funktionieren aber nur im StackLayout, sonst sind sie unwirksam. Die Default LayoutOption ohne zusätzliche Angabe ist „Fill“.

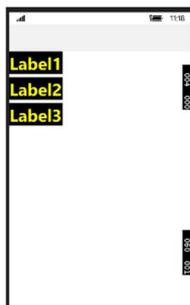
Da mit den LayoutOptions nur die Position eines Elementes definiert wird, ist eigentlich einfach zu verstehen, was passiert. Center setzt Element in die Mitte, End nach unten oder rechts, Start nach oben oder links, Fill füllt aus.

Sind z.B. drei Label in einem Stacklayout, so sehen diese mit Horizontal- und VerticalOptions so aus:

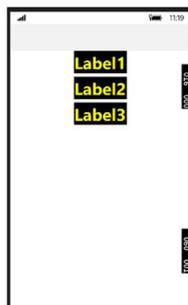
Beide Fill:



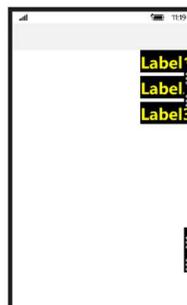
Start:



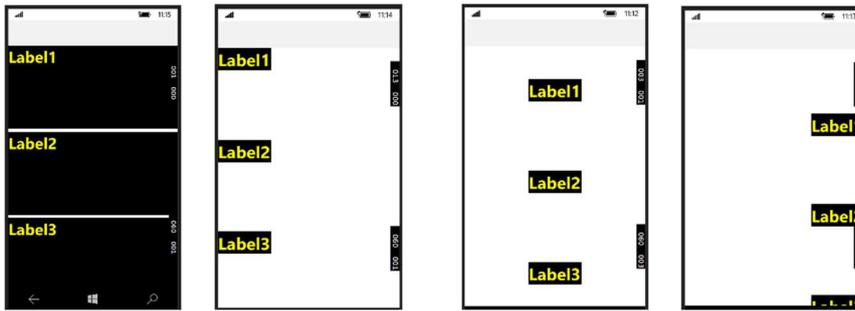
Center:



End:



FillAndExpand: StartAndExpand CenterAndExpand EndAndExpand:

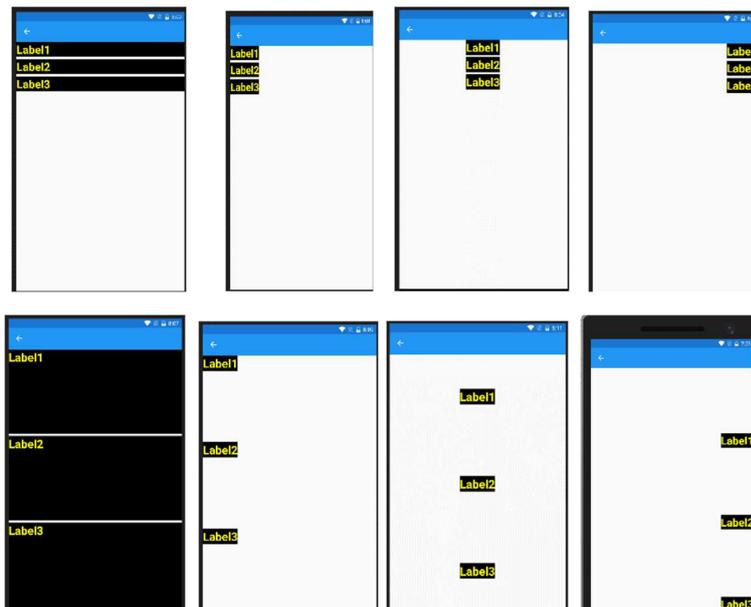


Bei Expand werden beim StackLayout die Elemente mit zusätzlichem Platz gleichmäßig über die verfügbare Fläche verteilt. Offensichtlich ist bei EndAndExpand die Fläche außerhalb des sichtbaren Bereiches, weil bei UWP bei der HomePage oben ein Titel steht und diese Fläche nicht mehr freigegeben wird.

Bei der horizontalen Positionierung hat Expand keine Wirkung. Bei dem StackLayout ist bei der vertikalen Positionierung ohne Expand kein Unterschied zu sehen.

Es werden für die drei Elemente drei gleich große Bereiche reserviert. Bei StartAndExpand werden die Elemente oben an die Bereiche, bei CenterAndExpand in der Mitte von den Bereichen und bei EndAndExpand am unteren Rand der Bereiche platziert.

Bei Android:



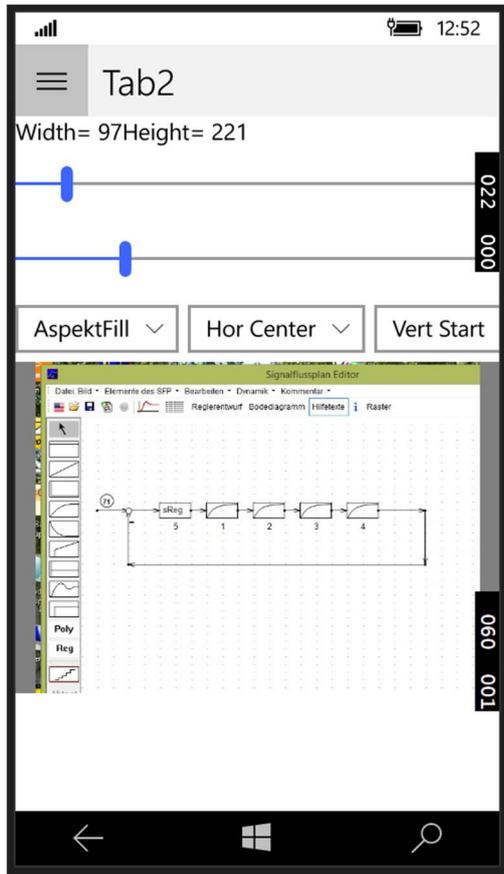
Damit der Unterschied zwischen Fill und Start sichtbar wird, habe ich den Hintergrund eines Labels Schwarz und die Schriftfarbe gelb gestellt:

```
TextColor =Color.Yellow,
BackgroundColor = Color.Black
```

Damit man auch mal sieht, wie ein „Zurück-Button“ funktionieren könnte habe ich diese Funktion auf das untere Label als Click- Ereignis mal programmiert.

Image Größe und Positionsdemo

Um alle Möglichkeiten der Größe und Position bei einem Image View auszuprobieren habe ich eine kleine Demo dazu geschrieben. In einer leeren neuen Content Page soll folgendes so angeordnet werden: das Bild kommt von meiner Seite: <http://www.joergbayerlein.de/wp-content/uploads/2015/06/regcSharp.jpg> und ist 683*514 Pixel groß.



Mit zwei Slidern sollen WidthRequest und HeightRequest des Image View verstellt werden, mit drei Pickern sollen dann alle Kombinationen der Eigenschaften Aspect und Horizontal- und Vertical – Options einstellbar sein.

Man kann dann in zwei Versionen ausprobieren wie das dann in einem normalen Stacklayout mit (Tab2) und ohne (Tab3) ScrollView aussieht.

Das alles in der AppBayFinal2019 soweit programmiert. Die beiden Seiten werden mit der MasterDetail- Page aus der obigen Demo aufgerufen.

Details:

Alle Slider:

```
Slider s1 = new Slider
{
    Maximum = 1000,
    HorizontalOptions = LayoutOptions.Fill,
    VerticalOptions = LayoutOptions.Start
};
```

In den ValueChange- Ereignissen:

```
private void S12_ValueChanged(object sender, ValueChangedEventArgs e)
{
    image.HeightRequest = e.NewValue;
    lab.Text = "Height= " + image.Height;
}
```

Oder der andere entsprechend für WidthRequest. Mit Width und Height kann man dann die tatsächlichen Dimensionen des Image auslesen.

Die Picker werden mit Text-Items gefüllt:

```
pic1 = new Picker
{
    VerticalOptions = LayoutOptions.Start,
};
pic1.Items.Add("AspektFit");
pic1.Items.Add("AspektFill");
pic1.Items.Add("Fill");
pic1.SelectedIndex = 0;
```

Der Index wird zu Anfang auf 0 gesetzt, damit der erste Eintrag angezeigt wird. Die anderen beiden Picker:

```
picV0 = new Picker
{
    VerticalOptions = LayoutOptions.Start,
```

```

};
picV0.Items.Add("Vert Fill");
picV0.Items.Add("Vert Start");
picV0.Items.Add("Vert End");
picV0.Items.Add("Vert Center");
picV0.Items.Add("Vert Fill+Exp");
picV0.Items.Add("Vert Start+Exp");
picV0.Items.Add("Vert End+Exp");
picV0.Items.Add("Vert Cent+Exp");

```

und entsprechend picHO. In den SelectedIndexChanged- Methoden werden dann in einer Switch – Anweisung die entsprechenden Werte gesetzt:

```

switch (picV0.SelectedIndex)
{
    case 0: image.HorizontalOptions = LayoutOptions.Fill; break;
    case 1: image.HorizontalOptions = LayoutOptions.Start; break;

```

usw. oder

```

switch (pic1.SelectedIndex)
{
    case 0: image.Aspect = Aspect.AspectFit; break;

```

So kann man in Ruhe schnell alle Kombinationen ausprobieren. Es ist sehr komplex und undurchschaubar. Aber man bekommt alles wie gewünscht hin.

Data Binding

Hier noch eine einfachere Data Binding, das in FormsGallery schon oft benutzt wurde. Damit soll jetzt im Beispiel oben ein Slider mit einem Label zum Anzeigen des Slider- Wertes benutzt werden. Zuerst definiert man eine Instanz von Binding mit Source und Path:

```

Binding binding = new Binding
{
    Source = slider,
    Path = "Value"
};

```

In Source steht das Objekt, von dem die Daten kommen, in Path als String der Name der Eigenschaft, die übergeben werden soll.

In einem weiteren Statement folgt dann die Bindung:

```
label.SetBinding(Label.TextProperty, binding);
```

Dort wird dann gesagt, wohin der übergebene Wert gehen soll. Leider geht das in AppBayFinal2019 auf der Seite Page3 nur einmal, dann ist die Bindung vorbei. Fehlersuche später.

GestureRecognizers

Damit kann man Elementen Reaktionen auf Touch des Users programmieren. Das geht so:

```

TapGestureRecognizer tap = new TapGestureRecognizer();
tap.Tapped += Tap_Tapped;
image.GestureRecognizers.Add(tap);

```

In der ersten Zeile wird ein Recognizer tap deklariert. In der zweiten Zeile wird ein Ereignis erzeugt, das beim tap ausgelöst wird. Und in der dritten Zeile wird dies dem Bild image zugewiesen.

Damit man sieht, dass etwas passiert:

```

private void Tap_Tapped(object sender, EventArgs e)
{

```

```

        image.WidthRequest = image.Width + 50;
        image.HeightRequest = image.Height + 50;
    }

```

Pro Touch auf das Bild wird es um 50 Einheiten in beide Richtungen größer. Man kann dieses Tap-Ereignis auch gleichzeitig an andere Elemente hängen. Dies funktioniert auch:

```
lab.GestureRecognizers.Add(tap);
```

Jetzt wird das Bild auch größer, wenn man auf den Label tapped.

Alert Box mit DisplayAlert

Analog zur ShowMessage – Box gibt es auch eine solche in Xamarin.Forms. Aufruf z.B. mit einer Async – function, die ich hinter einem Button- Clicked gelegt habe (auf TabPage4):

```

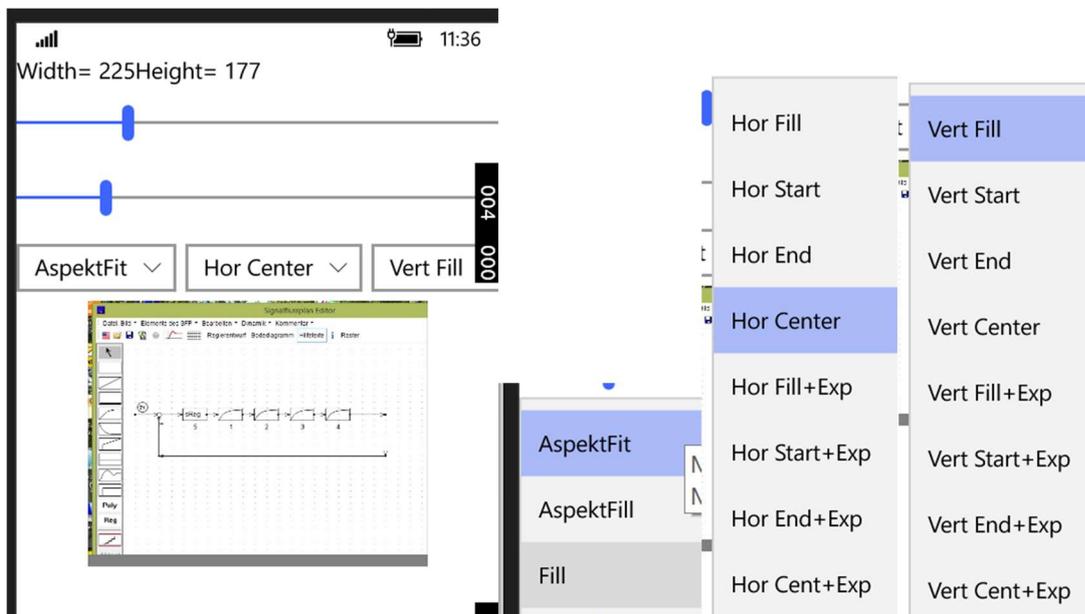
private async void But_Clicked(object sender, EventArgs e)
{
    bool alert = await DisplayAlert("Alert", "Are you sure to change the
background color of page?", "Yes", "No");
    if (alert)
    {
        this.BackgroundColor = Color.Blue;
    }
    else
        this.BackgroundColor = Color.Red;
}

```

Wenn alert nach Druck auf den „Yes“- Button true ist, wird der Hintergrund blau, sonst rot.

PageImageLayoutAspectohneScroll.cs

Hier wird aus dem Internet von meiner Homepage ein Bild geladen und in einer Image auf einer ContentPage dargestellt. Man kann alle Aspect – Werte und alle horizontalen und vertikalen Layoutoptions einstellen und Width und Height- Werte des Bildes verstellen und die Reaktion beobachten. Screen:

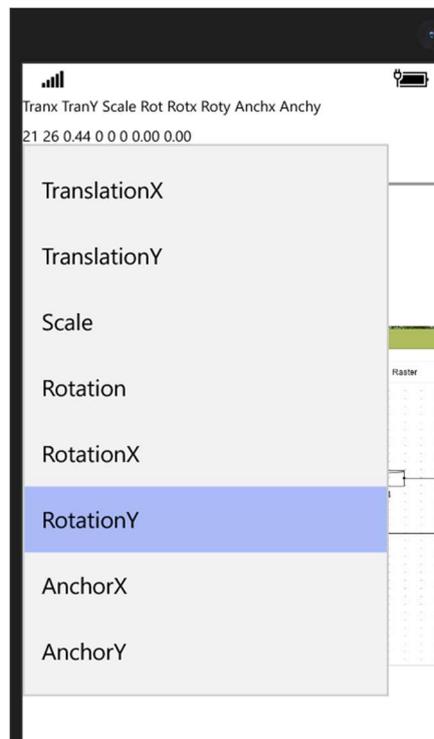


PageImageLayoutAspectScrollboth.cs

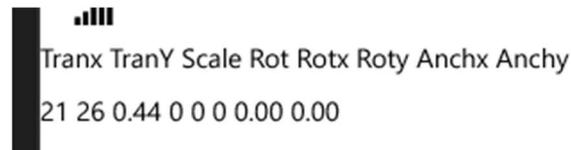
Praktisch identisch wie vorherige Seite, nur befindet sich das Bild jetzt in einem Scollview mit Orientation „Both“, so dass man in beide Richtungen scrollen kann, wenn das Bild zu groß wird.

[PageTransRotScaleAnchor.cs](#)

Hier kann man die Transformationseigenschaften ausprobieren. Mit einem Slider werden Werte



eingestellt, mit einem Picker wird eingestellt, welche Eigenschaft man mit diesem Wert verstellen will: Mit Translation kann man das Bild in x und y- Richtung verschieben, dort sind Werte zwischen -1000 und 1000 einstellbar. Alle aktuellen Werte werden oben in einem Label angezeigt:



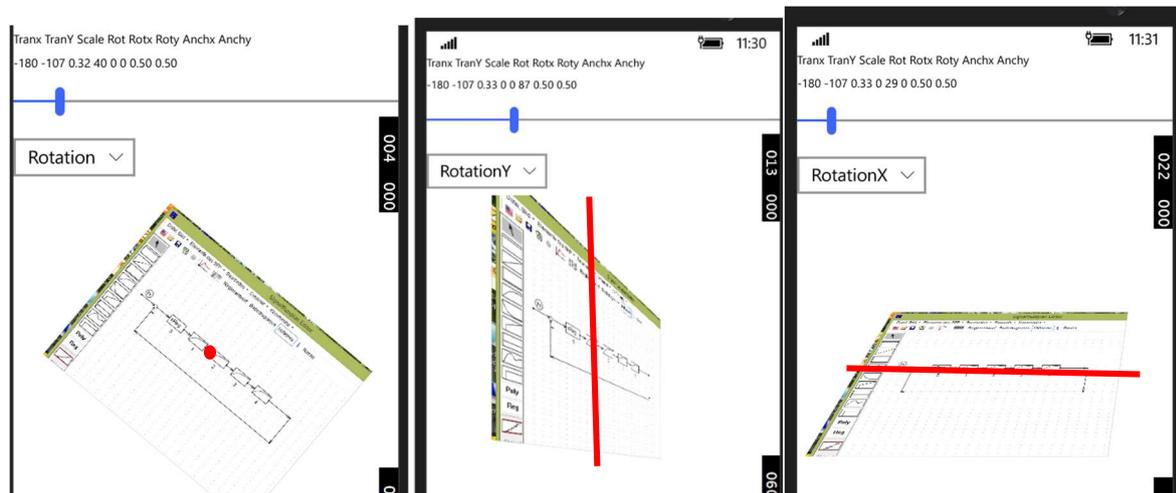
Es sind 8 Werte einstellbar, die man im Picker links sieht.

Mit Scale kann das Image in der Größe verändert werden, die Eigenschaft verlangt eine Fließkommazahl. 0 heißt Größe Null, 1 ist Originalgröße und z.B. 2 ist doppelte Größe.

Mit AnchorX / Y wird die Position der Drehachse für die Rotationen eingestellt, Standardwerte ist 0,5 und befindet sich in der Mitte des Drehobjektes.

Die Rotationen erwarten Werte zwischen 0 und 360, wobei dies der Grad der Drehung ist. Bei 360 hat das Objekt eine vollständige Drehung gemacht.

Rotation dreht um eine Achse senkrecht zum Bild, RotationY um eine senkrechte y- Achse und RotationX um eine waagerechte X-Achse:



Rotation

RotationY

RotationX

Man muss es mal selbst ausprobieren, dann versteht man es leicht.

[PageDemoGestures.cs](#)

Mit dieser Seite werden die drei möglichen Finger-Bewegungen auf dem schon bekannten Bild demonstriert: Tap, Pan und Pinch.

Tap: auf das Bild tippen ohne Wischbewegung

Pan: mit einem Finger wischen

Pinch: mit zwei Fingern wischen (wie man das zum Vergrößern bei Bildern macht)

Die Seite hat nur einen Label lab (global außerhalb Konstruktor deklariert) und das image.

Tap: Dazu füge man im Konstruktor hinzu

```
TapGestureRecognizer tap = new TapGestureRecognizer();
tap.Tapped += Tap_Tapped;
image.GestureRecognizers.Add(tap);
```

in der Tap_Tapped- Ereignisfunktion mache ich nur simple

```
private void Tap_Tapped(object sender, EventArgs e)
{
    lab.Text = "Image is tapped";
}
```

und man kann dann sehen, dass man getapped hat.

Pan:

Im Konstruktor hinzufügen:

```
PanGestureRecognizer pan = new PanGestureRecognizer();
pan.PanUpdated += Pan_PanUpdated;
image.GestureRecognizers.Add(pan);
```

in der Ereignisfunktion Pan_PanUpdated:

```
private void Pan_PanUpdated(object sender, PanUpdatedEventArgs e)
{
    lab.Text = "Image is panned " + e.TotalX.ToString("f2") + " " + e.TotalY.ToString("f2");
    double x=e.TotalX, y=e.TotalY;
}
```

Hier kann man jetzt die Koordinaten x und y zu entsprechenden Aktionen benutzen. Beim Berühren stehen sie immer auf Null, bewegt man den Finger dann, sieht man die Koordinaten ausgehend vom Startberührungspunkt.

Ich habe einige Befehle ergänzt, um das Bild mit dem Finger zu verschieben. Da beim Abheben nach einer Wischbewegung die beiden Koordinaten sofort wieder auf Null gestellt werden, muss man sich in dem Moment die aktuelle Position des Bildes in xoff und yoff merken. Dazu gibt es aber die beiden Informationen im Event e.StatusType. Dort wird zurückgegeben, ob das Pan beginnt, läuft oder stoppt:

Außerhalb einer Funktion:

```
double xoff = 0, yoff = 0;
```

In Pan_updated ergänzen:

```
if (e.StatusType==GestureStatus.Running)
{
    image.TranslationX = e.TotalX+xoff;
    image.TranslationY = e.TotalY+yoff;
}
if (e.StatusType==GestureStatus.Completed || e.StatusType==GestureStatus.Started)
{
    xoff = image.TranslationX;
    yoff = image.TranslationY;
}
```

```
}
```

Dann folgt das Bild der Fingerbewegung.

Pinch:

Genauere Beschreibung siehe

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

Einfügen:

```
PinchGestureRecognizer pinch = new PinchGestureRecognizer();
pinch.PinchUpdated += Pinch_PinchUpdated;
image.GestureRecognizers.Add(pinch);
```

In Pinch_Updated muss man beim Starten (abfragbar in e.Status) den aktuellen Scale- Faktor des Image festhalten und den aktuellen Mittelpunkt der beiden Finger in e.ScaleOrigin in die Scale- Achse des Bildes stehend in Anchor kopieren. Beim Scaling wird das Bild genau um Anchor herum vergrößert oder verkleinert.

```
if (e.Status == GestureStatus.Started)
{
    startScale = image.Scale;
}
if (e.Status == GestureStatus.Running)
{
    // Calculate the scale factor to be applied.
    currentScale += (e.Scale - 1);
    currentScale = Math.Max(0.5, currentScale);
    // Apply scale factor.
    image.Scale = currentScale;
}
```

Bitte ausprobieren.

Auf meinem Tablet der „Lokale Computer“ funktioniert alles tadellos.

Beim Mobile Emulator geht Tap, Pan, aber Pinch nicht.

Beim Android Emulator 5“ KitKat (4.4) geht Tap, Pan, aber Pinch nicht.

Bei meinem Samsung GT-I9505 geht alles, aber nicht so richtig gut.

Bei Rasp geht Tap und Pan, aber Pinch nicht.

[PageGesturesSimple.cs](#)

Diese Seite ist eine Weiterentwicklung gegenüber dem vorherigen Kapitel. Dort wird insbesondere im Pan und Pinch etwas verbessert. Insbesondere ist das Pan im Android anders organisiert, so dass dort anders gerechnet werden muss. Dies ist in der Zeile

```
image.TranslationX = x*image.Scale+xoff;
image.TranslationY = y*image.Scale+yoff;
if (Device.RuntimePlatform == Device.Android)
{
    xoff = image.TranslationX;
    yoff = image.TranslationY;
}
```

Zu erkennen Bei UWP reichen die ersten beiden Zeilen, bei Android muss bei jeder Fingerbewegung die aktuelle Position in x/yoff gespeichert werden.

Gezeichnet Prof. Dr. Bayerlein Dez. 2020