# Layout Example:

This example shows creating a Grid in XAML, then putting it inside a nested stack.

**Star “\*” means take up as much room as possible.**

Start with the grid:

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="\*" />

<RowDefinition Height="\*" />

<RowDefinition Height="\*" />

<RowDefinition Height="\*" />

<RowDefinition Height="\*" />

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="\*" />

</Grid.ColumnDefinitions>

<Button Text="1" Grid.Row="0" Grid.Column="0" />

<Button Text="2" Grid.Row="0" Grid.Column="1" x:Name="btn1"/>

<Button Text="3" Grid.Row="0" Grid.Column="2" x:Name="btn1"/>

<Button Text="4" Grid.Row="1" Grid.Column="0" />

<Button Text="5" Grid.Row="1" Grid.Column="1" />

<Button Text="6" Grid.Row="1" Grid.Column="2" />

<Button Text="7" Grid.Row="2" Grid.Column="0" />

<Button Text="8" Grid.Row="2" Grid.Column="1" />

<Button Text="9" Grid.Row="2" Grid.Column="2" />

<Button Text="0" Grid.Row="3" Grid.Column="1" />

<Button Text="+" Grid.Row="4" Grid.Column="0" />

<Button Text="-" Grid.Row="4" Grid.Column="1" />

<Button Text="=" Grid.Row="4" Grid.Column="2" />

</Grid>

Add the nested stack layouts, maybe add the margins and other options after the first run:

This will go before the grid definition, add a closing tag after the grid.

<StackLayout VerticalOptions="Center" Margin="30,30,30,30">

<StackLayout Orientation="Horizontal">

<Label Text="Output:" HorizontalOptions="Start" />

<Label Text="" HorizontalOptions="CenterAndExpand" x:Name="lblResult"/>

</StackLayout>

Linking up to the buttons on the grid (and doing some nonsense):

MainPage = new XamarinLayouts.MainPage();

Button btn1 = MainPage.FindByName<Button>("btn1");

Button btn2 = MainPage.FindByName<Button>("btn2");

Label lblResult = MainPage.FindByName<Label>("lblResult");

btn1.Clicked += (sender, e) => { lblResult.Text += btn1.Text; };

btn2.Clicked += (sender, e) => { lblResult.Text += btn2.Text; };

Exercise: Create the above using the code approach.

# Layout Options Demonstration

This demonstrates what the different LayoutOptions really do. Using a StackLayout as the Expansion settings only matter with it. This example is used on the Xamarin Developer documentation site.

Create a stack layout at the top of the App class:

static StackLayout stackLayout = new StackLayout

{

BackgroundColor = Color.Gray,

Padding = 2,

Spacing = 2,

};

Create a method that will just add children based upon some text and a layout option setting:

private static void AddElement(string text, LayoutOptions layoutOption)

{

stackLayout.Children.Add(new Label

{

Text = text,

BackgroundColor = Color.White,

HorizontalTextAlignment = TextAlignment.Center,

VerticalTextAlignment = TextAlignment.Center,

HorizontalOptions = layoutOption,

VerticalOptions = layoutOption,

WidthRequest = 160, // sets how wide we would like it to be

HeightRequest = 25,

});

stackLayout.Children.Add(new BoxView // a flattened box to show in between

{

HeightRequest = 1,

Color = Color.Yellow,

});

}

Then call in the constructor that for each of the 8 options and add the stack layout to the Main Page.

AddElement("Start", LayoutOptions.Start);

AddElement("Center", LayoutOptions.Center);

AddElement("End", LayoutOptions.End);

AddElement("Fill", LayoutOptions.Fill);

AddElement("StartAndExpand", LayoutOptions.StartAndExpand);

AddElement("CenterAndExpand", LayoutOptions.CenterAndExpand);

AddElement("EndAndExpand", LayoutOptions.EndAndExpand);

AddElement("FillAndExpand", LayoutOptions.FillAndExpand);

MainPage = new ContentPage

{

Padding = new Thickness(0, Device.OnPlatform<int>(20, 0, 0), 0, 0),

Content = stackLayout,

};

Things to note: Additional space is taken up evenly by the “Expand” boxes. Only Fill will make the label larger if there is room, expand won’t (see CenterAndExpand).