Declarative, Code-Based Forms UI

RYAN DAVIS

Queensland C# Mobile Developers Meetup

2019 06 25

whoami

- Ryan Davis
- Professional ~~Mobile~~ LINQPad Developer

ryandavis.io rdavis_au

rdavisau

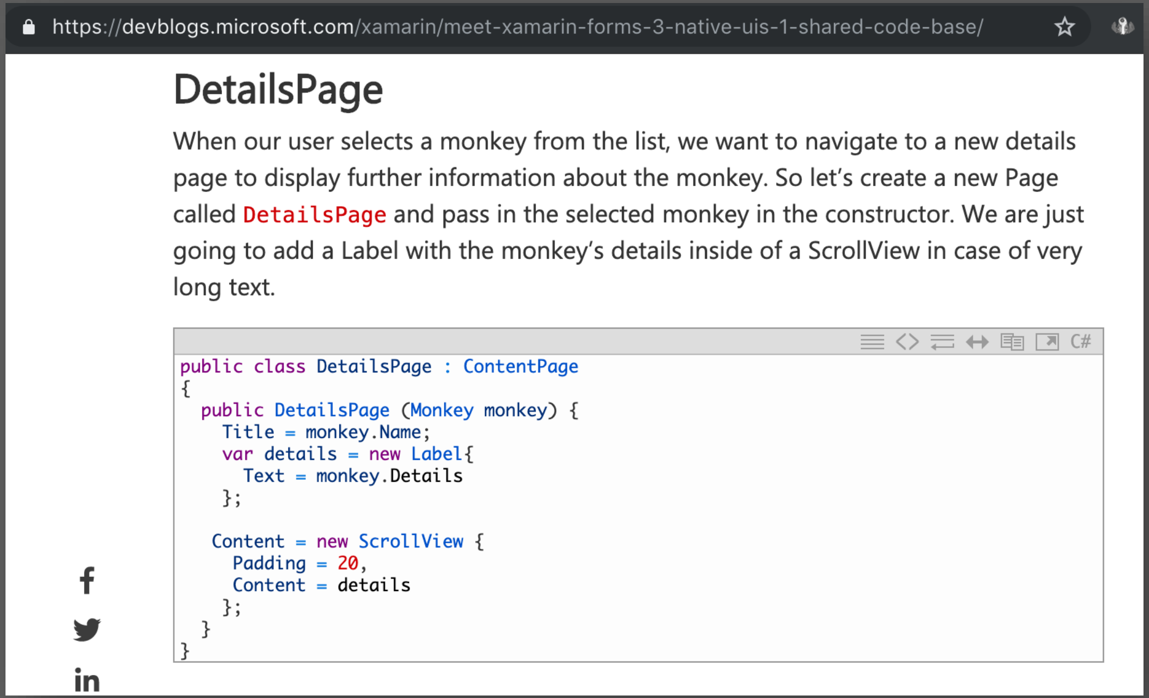
- **essential-interfaces** – use DI/mocking with Xamarin.Essentials
- **dumpeditable-linqpad** – extensible inline object editor for LINQPad
- **jsondatacontext-linqpad** – json data context driver for LINQPad
- **sockets-for-pcl, sockethelpers** – socket comms in a PCL
  (today you should use netstandard sockets why are you all still installing this)

to cover

- the story of code and xaml
- the hero coded ui needs
- demos, samples
- resources

in the beginning, code was the favourite

When Xamarin.Forms was announced (late May 2014), it shipped with support for code and XAML. In the early days, blog posts featured C#-based samples.



https://devblogs.microsoft.com/xamarin/meet-xamarin-forms-3-native-uis-1-shared-code-base/

DetailsPage

When our user selects a monkey from the list, we want to navigate to a new details page to display further information about the monkey. So let's create a new Page called `DetailsPage` and pass in the selected monkey in the constructor. We are just going to add a Label with the monkey's details inside of a ScrollView in case of very long text.

```
public class DetailsPage : ContentPage
{
  public DetailsPage (Monkey monkey) {
    Title = monkey.Name;
    var details = new Label{
      Text = monkey.Details
    };

   Content = new ScrollView {
     Padding = 20,
     Content = details
    };
  }
}
```

Code sample from the original Xamarin.Forms announcement post

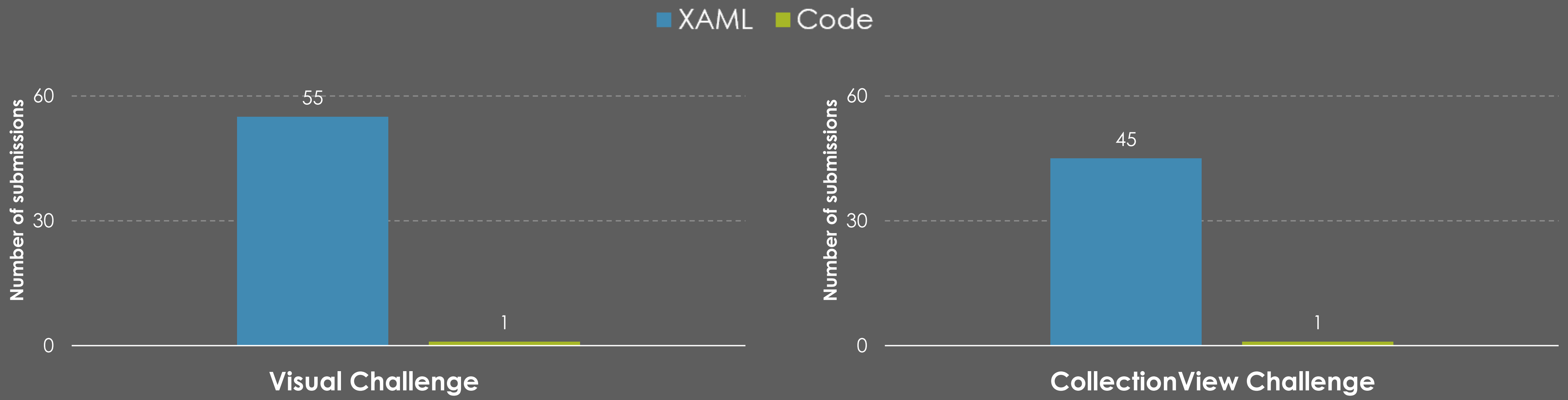
however, the tide quickly turned

For a variety of reasons, XAML became favoured by a majority of Forms developers.

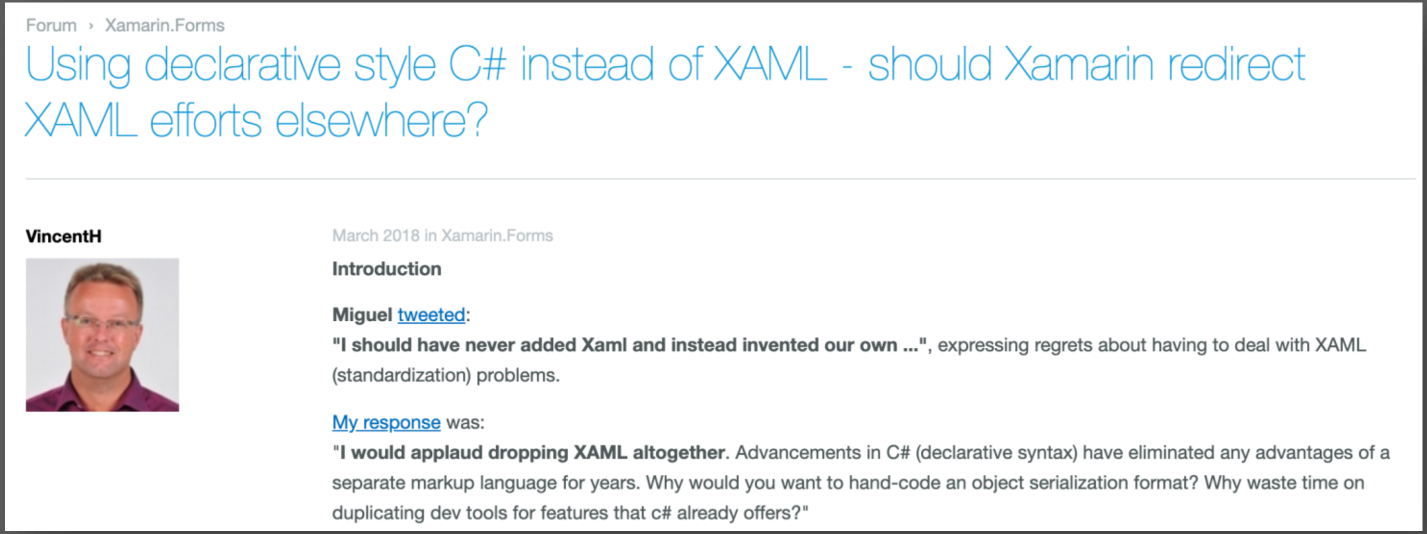
the xaml popularity was unstoppable

n=102 But Possibly Real World Representative Comparison of Popularity Between XAML and C# based on Recent Xamarin.Forms Challenges

(higher is better)



still, a dedicated few stayed hopeful for C#



Forum › Xamarin.Forms

Using declarative style C# instead of XAML - should Xamarin redirect XAML efforts elsewhere?

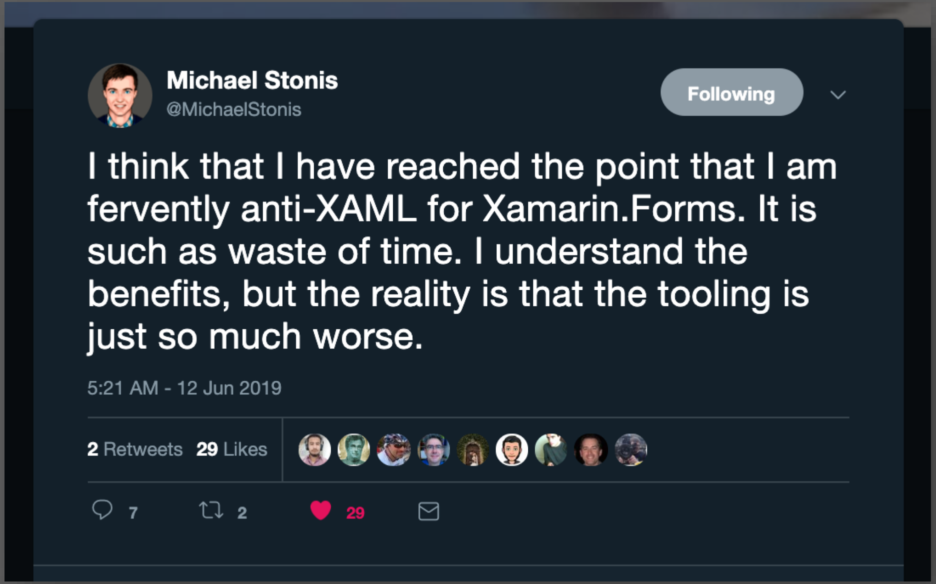
VincentH

March 2018 in Xamarin.Forms

Introduction

Miguel tweeted:
"I should have never added Xaml and instead invented our own ...", expressing regrets about having to deal with XAML (standardization) problems.

My response was:
"**I would applaud dropping XAML altogether**. Advancements in C# (declarative syntax) have eliminated any advantages of a separate markup language for years. Why would you want to hand-code an object serialization format? Why waste time on duplicating dev tools for features that c# already offers?"



Michael Stonis
@MichaelStonis

I think that I have reached the point that I am fervently anti-XAML for Xamarin.Forms. It is such as waste of time. I understand the benefits, but the reality is that the tooling is just so much worse.

5:21 AM - 12 Jun 2019

2 Retweets **29** Likes

7 2 29

https://forums.xamarin.com/discussion/123771/using-declarative-style-c-instead-of-xaml-should-xamarin-redirect-xaml-efforts-elsewhere?

using XF’s API from C# can be awkward

- Some tasks that are ‘easy’ in XAML feel heavy handed in C#
- Xamarin.Forms API will often force you to adopt imperative code
- People look at you funny

but we can use C# to make using C# better

- Helper methods that wrap or abstract awkward tasks
- Syntax that encourages a more declarative usage
- DSL-like methods that minimise boilerplate

enter CSharpForMarkup by VincentH.NET

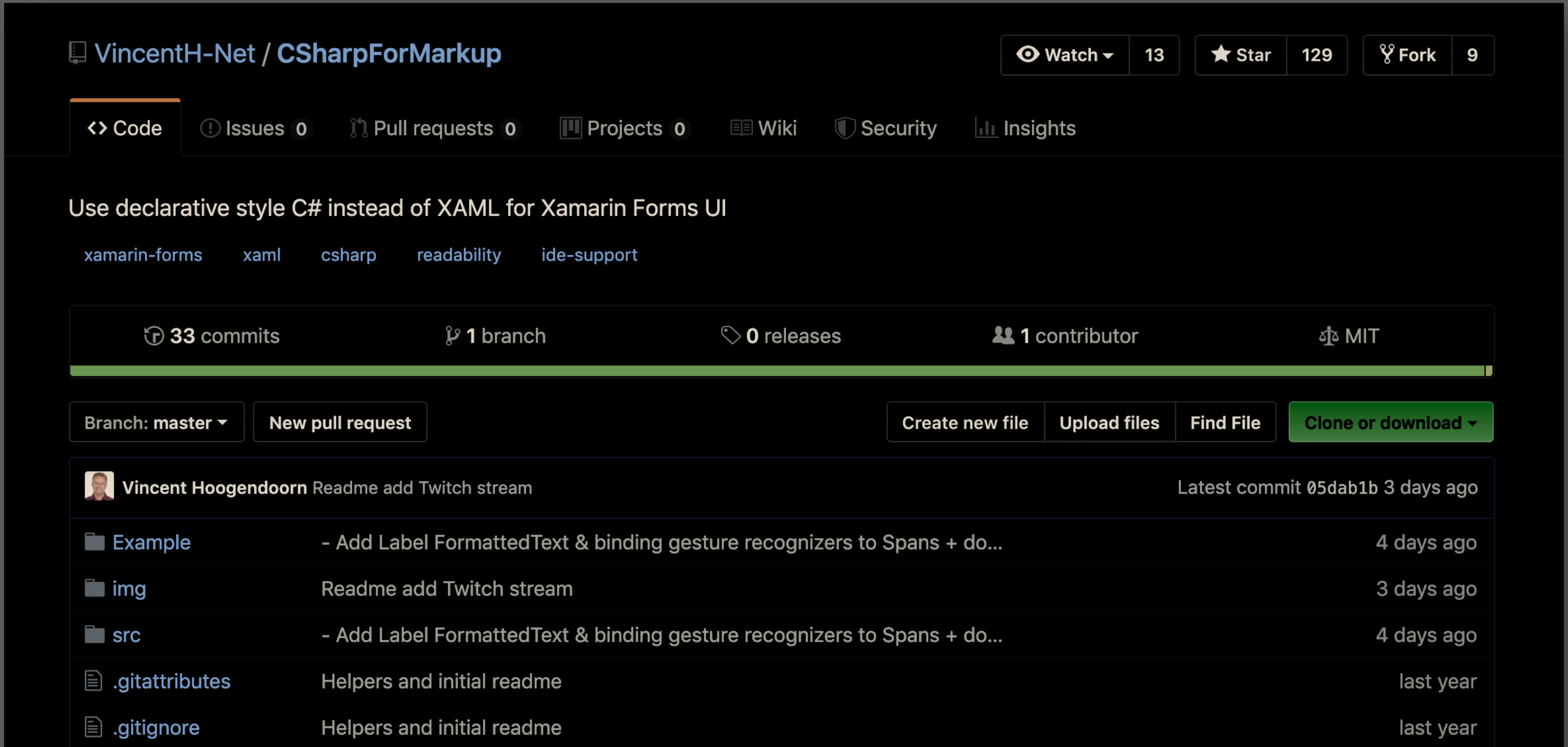
A set of extension methods and helper functions that allow the use of a declarative style of C# instead of XAML for the creation of Xamarin Forms UI.

https://github.com/VincentH-Net/CSharpForMarkup

- Supports the creation of **concise**, **declarative**, **readable** UI definitions
- **Appropriately constrains** clever code solutions in UI definitions
- Enables greater use of **type-safety** than raw code or XAML alone
- **Battle-tested**, used in production apps
- Lets you write code that **makes you feel good** when you look back at it

just download one file from the repo

Read the README for good examples, tips, tricks and guidelines



VincentH-Net / CSharpForMarkup

Use declarative style C# instead of XAML for Xamarin Forms UI

xamarin-forms xaml csharp readability ide-support

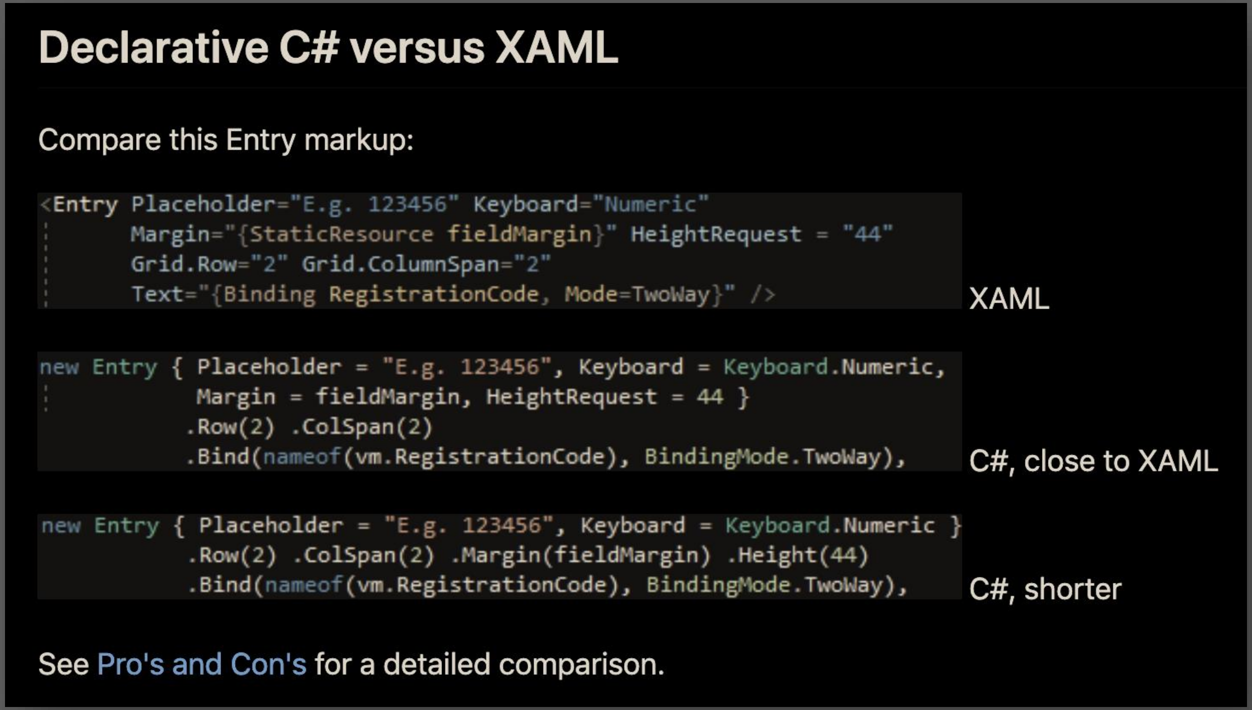
33 commits | 1 branch | 0 releases | 1 contributor | MIT

Vincent Hoogendoorn Readme add Twitch stream — Latest commit 05dab1b 3 days ago

Example	- Add Label FormattedText & binding gesture recognizers to Spans + do...	4 days ago
img	Readme add Twitch stream	3 days ago
src	- Add Label FormattedText & binding gesture recognizers to Spans + do...	4 days ago
.gitattributes	Helpers and initial readme	last year
.gitignore	Helpers and initial readme	last year

just download one file from the repo

Read the README for good examples, tips, tricks and guidelines



Declarative C# versus XAML

Compare this Entry markup:

```
<Entry Placeholder="E.g. 123456" Keyboard="Numeric"
       Margin="{StaticResource fieldMargin}" HeightRequest = "44"
       Grid.Row="2" Grid.ColumnSpan="2"
       Text="{Binding RegistrationCode, Mode=TwoWay}" />
```

XAML

```
new Entry { Placeholder = "E.g. 123456", Keyboard = Keyboard.Numeric,
            Margin = fieldMargin, HeightRequest = 44 }
           .Row(2) .ColSpan(2)
           .Bind(nameof(vm.RegistrationCode), BindingMode.TwoWay),
```

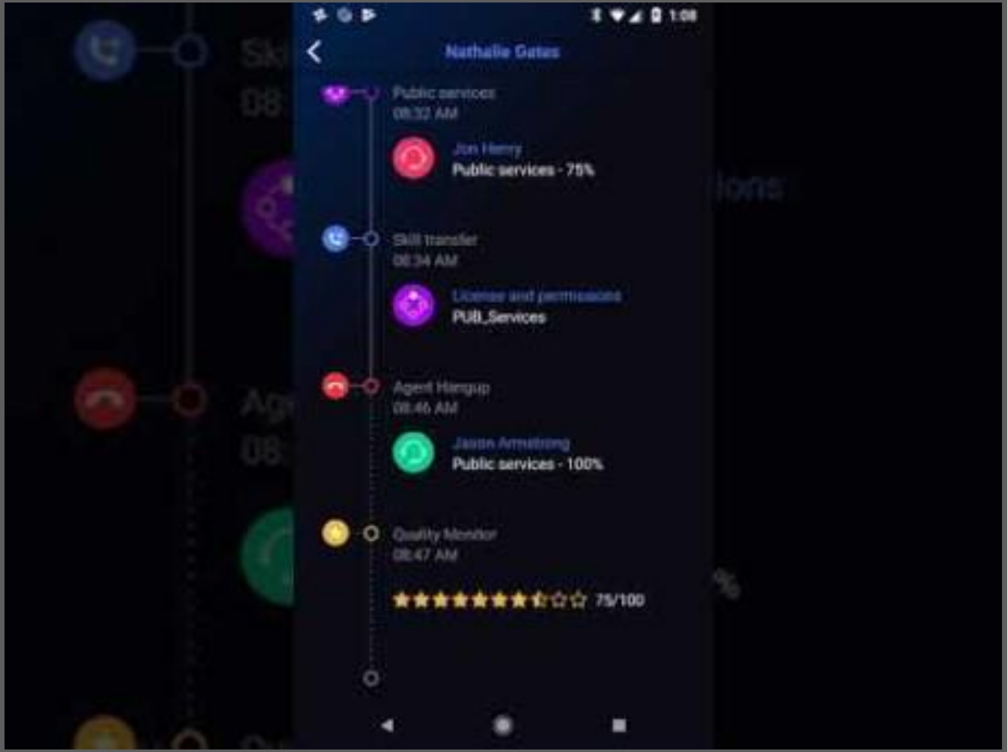
C#, close to XAML

```
new Entry { Placeholder = "E.g. 123456", Keyboard = Keyboard.Numeric }
           .Row(2) .ColSpan(2) .Margin(fieldMargin) .Height(44)
           .Bind(nameof(vm.RegistrationCode), BindingMode.TwoWay),
```

C#, shorter

See Pro's and Con's for a detailed comparison.

Basic example of various helpers in use



Video of a shipping app using these helpers (see Resources)

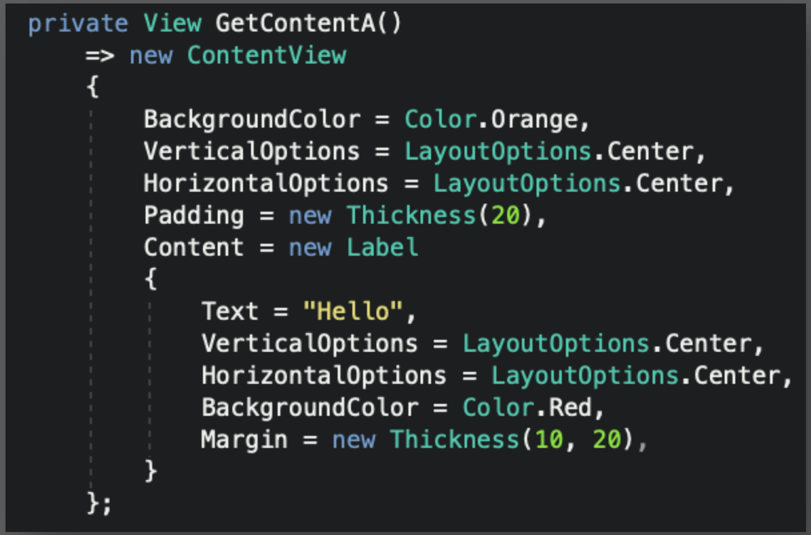
-= declarative code-based xf ui =-

CSharpForMarkup walkthrough

fluent, declarative control positioning

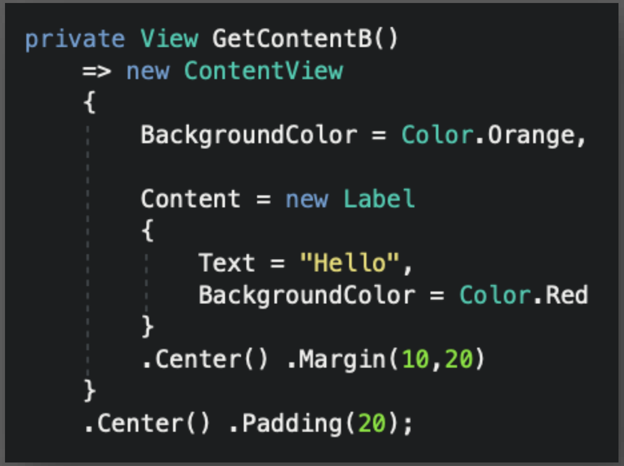
Specifying property names and values for common layout properties significantly increases the verbosity of C# based UI code.

CSharpForMarkup includes most common properties in concise helper methods.



```
private View GetContentA()
    => new ContentView
    {
        BackgroundColor = Color.Orange,
        VerticalOptions = LayoutOptions.Center,
        HorizontalOptions = LayoutOptions.Center,
        Padding = new Thickness(20),
        Content = new Label
        {
            Text = "Hello",
            VerticalOptions = LayoutOptions.Center,
            HorizontalOptions = LayoutOptions.Center,
            BackgroundColor = Color.Red,
            Margin = new Thickness(10, 20),
        }
    };
```

Layout options can often be specified without sacrificing declarative style but are quite verbose



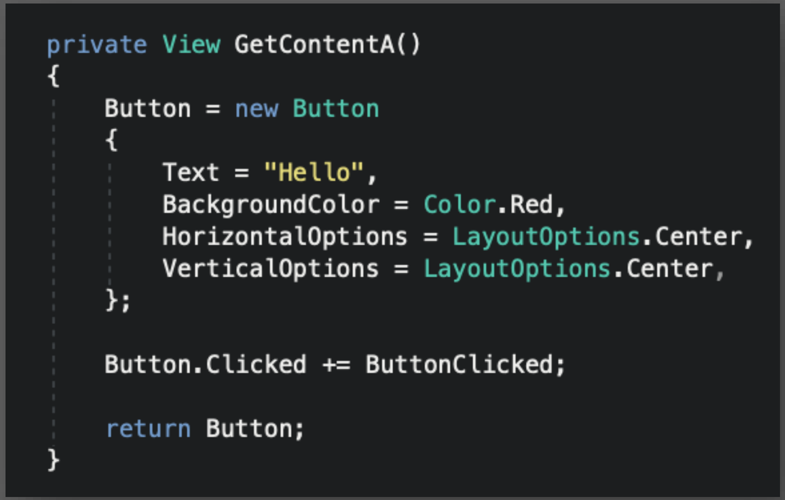
```
private View GetContentB()
    => new ContentView
    {
        BackgroundColor = Color.Orange,

        Content = new Label
        {
            Text = "Hello",
            BackgroundColor = Color.Red
        }
        .Center() .Margin(10,20)
    }
    .Center() .Padding(20);
```

CSharpForMarkup simplifies this code by reducing repeated code and providing additional helpers like .Center, which covers both vertical and horizontal positioning at once

simple side effects made easy

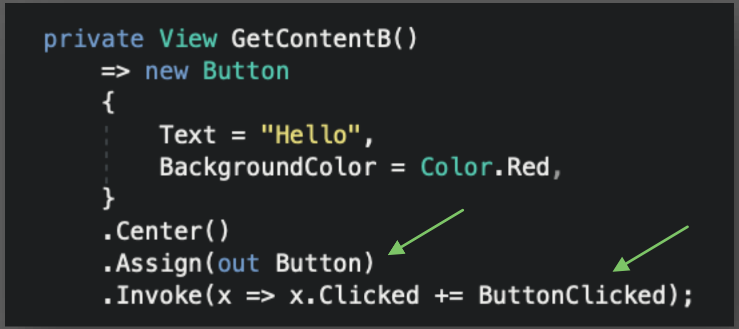
Ordinarily, actions like assigning to a field, attaching an event handler or calling a method on a control would all require dedicated statements.

.Assign and **.Invoke** allow these without sacrificing declarative code.



```
private View GetContentA()
{
    Button = new Button
    {
        Text = "Hello",
        BackgroundColor = Color.Red,
        HorizontalOptions = LayoutOptions.Center,
        VerticalOptions = LayoutOptions.Center,
    };

    Button.Clicked += ButtonClicked;

    return Button;
}
```

Side-effects like assignment and setting event handlers typically force an imperative style



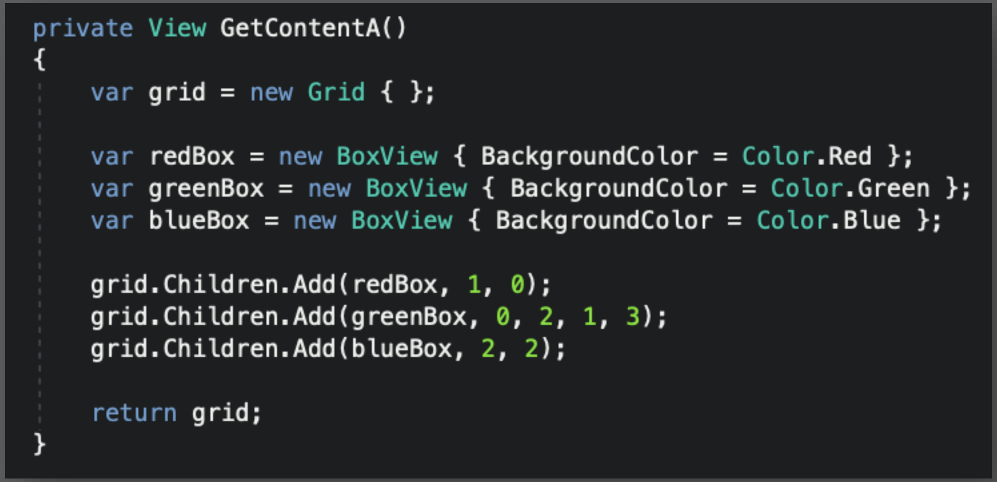
```
private View GetContentB()
    => new Button
    {
        Text = "Hello",
        BackgroundColor = Color.Red,
    }
    .Center()
    .Assign(out Button)
    .Invoke(x => x.Clicked += ButtonClicked);
```

CSharpForMarkup helpers handle these nicely

concise, readable grid positioning

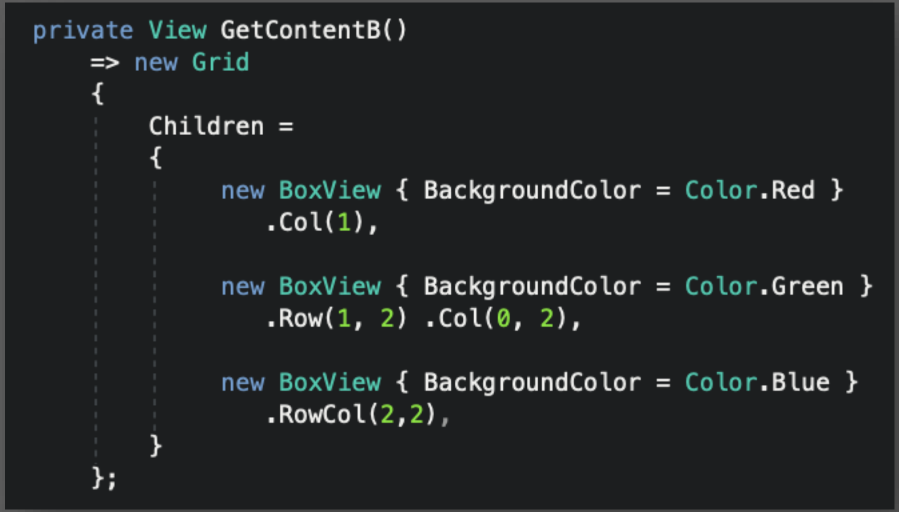
Grid positioning is particularly awkward from code.

.Row, **.Col**, **.RowCol** dramatically improve readability.



```
private View GetContentA()
{
    var grid = new Grid { };

    var redBox = new BoxView { BackgroundColor = Color.Red };
    var greenBox = new BoxView { BackgroundColor = Color.Green };
    var blueBox = new BoxView { BackgroundColor = Color.Blue };

    grid.Children.Add(redBox, 1, 0);
    grid.Children.Add(greenBox, 0, 2, 1, 3);
    grid.Children.Add(blueBox, 2, 2);

    return grid;
}
```

Assigning grid position normally sux, and again favours an imperative style



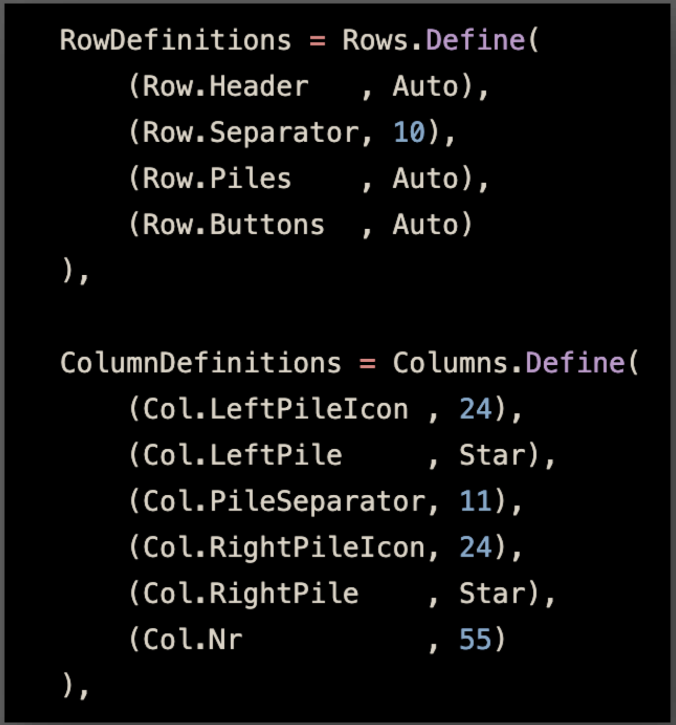
```
private View GetContentB()
    => new Grid
    {
        Children =
        {
            new BoxView { BackgroundColor = Color.Red }
               .Col(1),

            new BoxView { BackgroundColor = Color.Green }
               .Row(1, 2) .Col(0, 2),

            new BoxView { BackgroundColor = Color.Blue }
               .RowCol(2,2),
        }
    };
```

Using CSharpForMarkup, grids positioning can be performed cleanly and easily

type-safe, flexible grid ordering

Grid re-ordering is painful whether using XAML or code

CSharpForMarkup allows you to use enums to describe your grid positioning. By tieing interface definition to enum values, the “domino effect” associated with grid re-ordering can be removed.



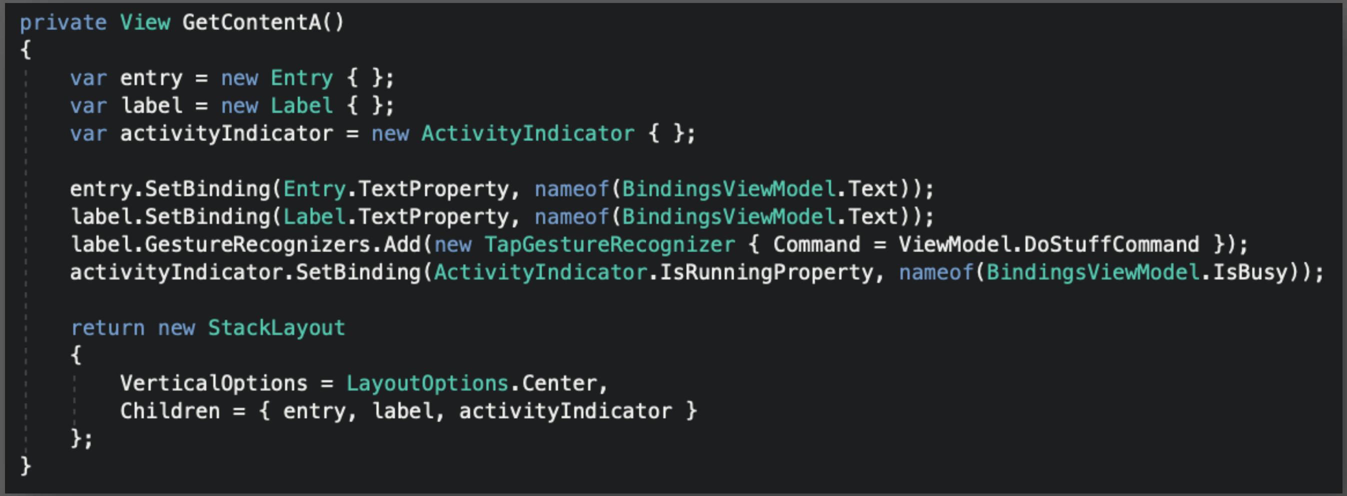
```
RowDefinitions = Rows.Define(
    (Row.Header    , Auto),
    (Row.Separator, 10),
    (Row.Piles     , Auto),
    (Row.Buttons   , Auto)
),

ColumnDefinitions = Columns.Define(
    (Col.LeftPileIcon , 24),
    (Col.LeftPile      , Star),
    (Col.PileSeparator, 11),
    (Col.RightPileIcon, 24),
    (Col.RightPile     , Star),
    (Col.Nr            , 55)
),
```

Strongly typed row and column definitions makes changes later much easier

simple – or sophisticated – inline bindings

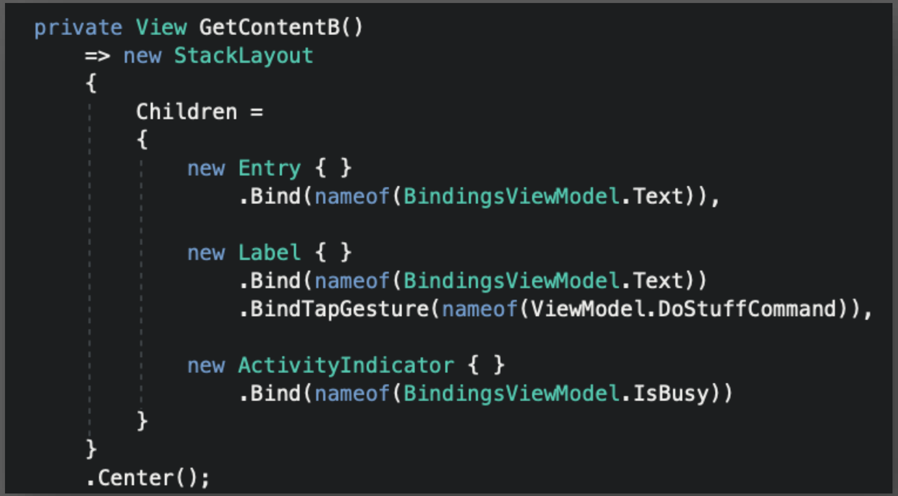
Code-based Xamarin.Forms bindings can involve verbose ceremony.

CSharpForMarkup includes inline **.Bind** helpers with default target properties to reduce the need for boilerplate, and a **.BindTapGesture** helper that attaches a TapGestureRecogniser to a control, bound to the specified command.



```
private View GetContentA()
{
    var entry = new Entry { };
    var label = new Label { };
    var activityIndicator = new ActivityIndicator { };

    entry.SetBinding(Entry.TextProperty, nameof(BindingsViewModel.Text));
    label.SetBinding(Label.TextProperty, nameof(BindingsViewModel.Text));
    label.GestureRecognizers.Add(new TapGestureRecognizer { Command = ViewModel.DoStuffCommand });
    activityIndicator.SetBinding(ActivityIndicator.IsRunningProperty, nameof(BindingsViewModel.IsBusy));

    return new StackLayout
    {
        VerticalOptions = LayoutOptions.Center,
        Children = { entry, label, activityIndicator }
    };
}
```

Code-based bindings also typically require imperative code style and multiple statements



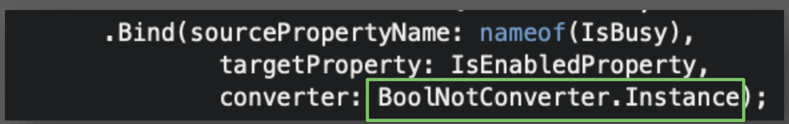
```
private View GetContentB()
    => new StackLayout
    {
        Children =
        {
            new Entry { }
                .Bind(nameof(BindingsViewModel.Text)),

            new Label { }
                .Bind(nameof(BindingsViewModel.Text))
                .BindTapGesture(nameof(ViewModel.DoStuffCommand)),

            new ActivityIndicator { }
                .Bind(nameof(BindingsViewModel.IsBusy))
        }
    }
    .Center();
```

CSharpForMarkup helpers improve brevity and readability dramatically

type-safe inline valueconverters

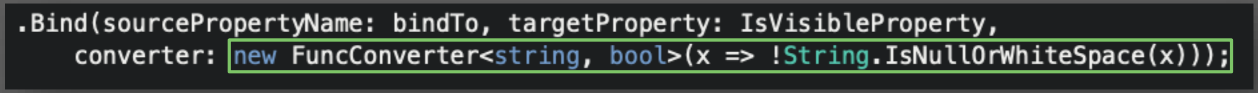
ValueConverters can be specified directly within **.Bind** calls when needed.

A **FuncConverter<TFrom, TTo>** allows conversions to be defined in line.



```
        .Bind(sourcePropertyName: nameof(IsBusy),
                targetProperty: IsEnabledProperty,
                converter: BoolNotConverter.Instance);
```

CSharpForMarkup includes some
common built in converters

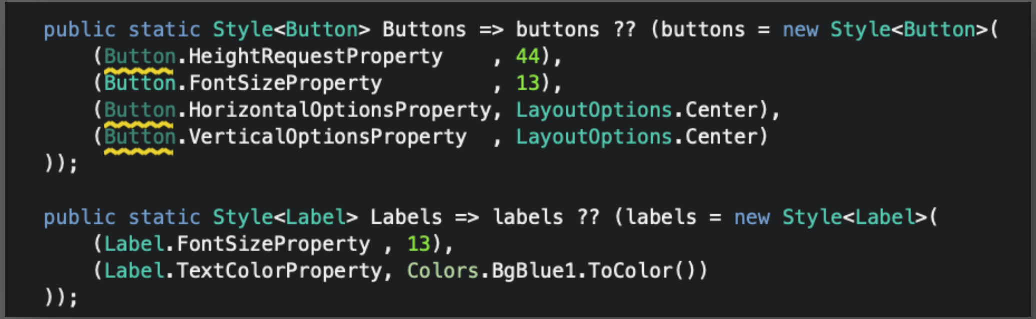


```
.Bind(sourcePropertyName: bindTo, targetProperty: IsVisibleProperty,
    converter: new FuncConverter<string, bool>(x => !String.IsNullOrWhiteSpace(x)));
```

Or you can define your own in a type-safe manner

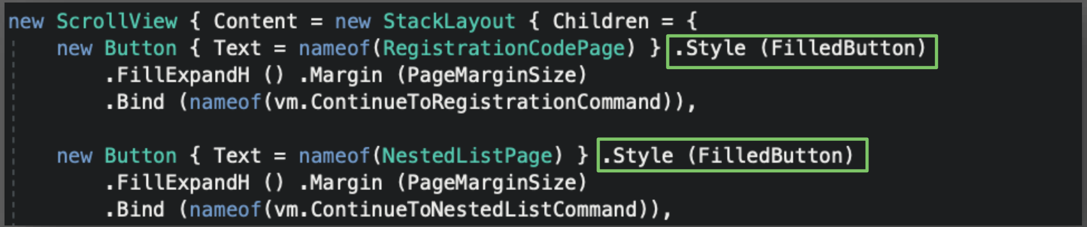
fonts and styles

Styles can be cleanly defined and applied in a type-safe manner using the **Style** helper class and associated methods.



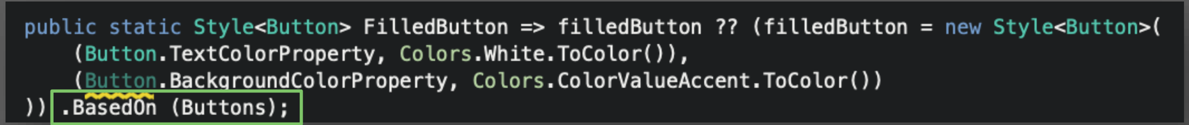
```
public static Style<Button> Buttons => buttons ?? (buttons = new Style<Button>(
    (Button.HeightRequestProperty    , 44),
    (Button.FontSizeProperty         , 13),
    (Button.HorizontalOptionsProperty, LayoutOptions.Center),
    (Button.VerticalOptionsProperty  , LayoutOptions.Center)
));

public static Style<Label> Labels => labels ?? (labels = new Style<Label>(
    (Label.FontSizeProperty , 13),
    (Label.TextColorProperty, Colors.BgBlue1.ToColor())
));
```

Defining styles is straightforward and readable



```
new ScrollView { Content = new StackLayout { Children = {
    new Button { Text = nameof(RegistrationCodePage) } .Style (FilledButton)
        .FillExpandH () .Margin (PageMarginSize)
        .Bind (nameof(vm.ContinueToRegistrationCommand)),

    new Button { Text = nameof(NestedListPage) } .Style (FilledButton)
        .FillExpandH () .Margin (PageMarginSize)
        .Bind (nameof(vm.ContinueToNestedListCommand)),
```

Once define, styles are easily applied using the Style method



```
public static Style<Button> FilledButton => filledButton ?? (filledButton = new Style<Button>(
    (Button.TextColorProperty, Colors.White.ToColor()),
    (Button.BackgroundColorProperty, Colors.ColorValueAccent.ToColor())
)) .BasedOn (Buttons);
```

Styles can derived from other styles

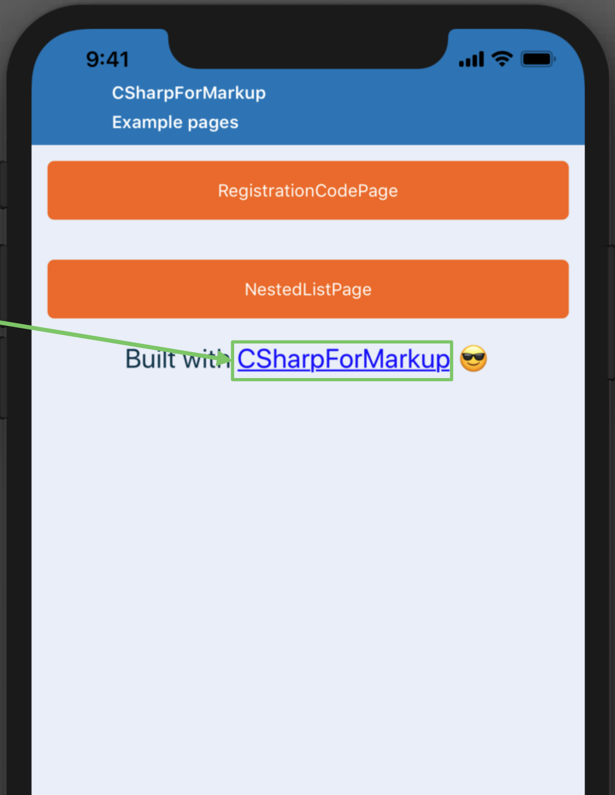
formatted spans are no problem

.FormattedText takes a **params Span[]** argument set

Spans can be bound to taps or gestures

```
new Label { } .FontSize (20) .FormattedText (
    new Span { Text = "Built with " },
    new Span { TextColor = Color.Blue, TextDecorations = TextDecorations.Underline }
    .BindTap (nameof(vm.ContinueToCSharpForMarkupCommand))
    .Bind (nameof(vm.Title)),
    new Span { Text = " \U0001f60e" }
) .CenterH ()
```

Individual span elements have the same binding powers as other views

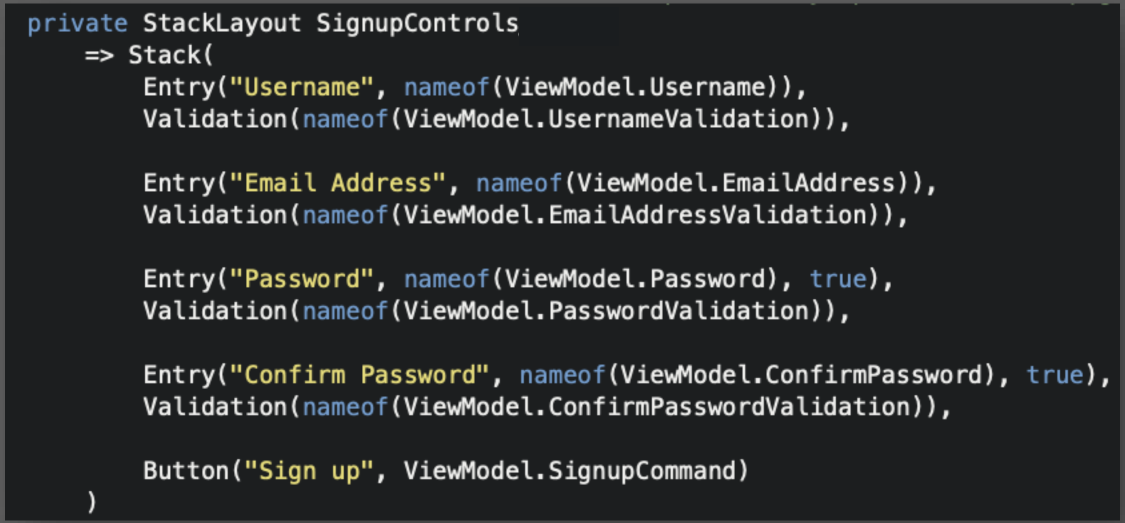


other benefits of code-based ui

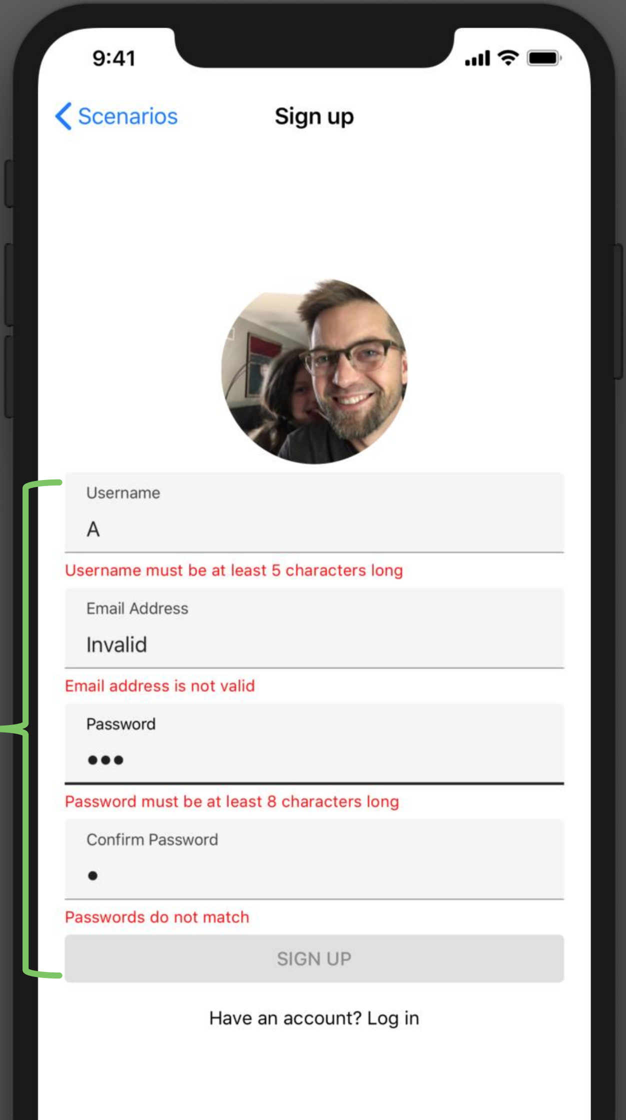
defining mini-DSLs can reduce repetition

CSharpForMarkup balances conciseness, consistency, constraint and maintainability.

Defining a DSL via helper methods can result in code that is potentially more concise and readable, at the cost of consistency and maintainability.



```
private StackLayout SignupControls
    => Stack(
        Entry("Username", nameof(ViewModel.Username)),
        Validation(nameof(ViewModel.UsernameValidation)),

        Entry("Email Address", nameof(ViewModel.EmailAddress)),
        Validation(nameof(ViewModel.EmailAddressValidation)),

        Entry("Password", nameof(ViewModel.Password), true),
        Validation(nameof(ViewModel.PasswordValidation)),

        Entry("Confirm Password", nameof(ViewModel.ConfirmPassword), true),
        Validation(nameof(ViewModel.ConfirmPasswordValidation)),

        Button("Sign up", ViewModel.SignupCommand)
    )
```



Username
A
Username must be at least 5 characters long

Email Address
Invalid
Email address is not valid

Password
●●●
Password must be at least 8 characters long

Confirm Password
●
Passwords do not match

SIGN UP

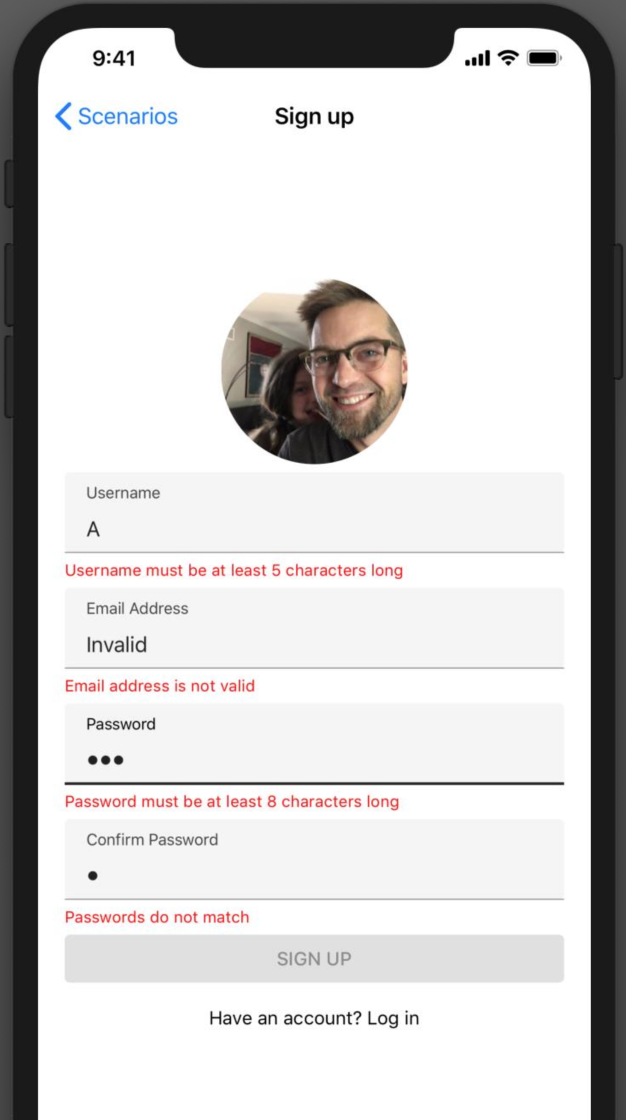
Have an account? Log in

defining mini-DSLs can reduce repetition

CSharpForMarkup balances conciseness, consistency, constraint and maintainability.

Defining a DSL via helper methods can result in code that is potentially more concise and readable, at the expense of consistency and maintainability.

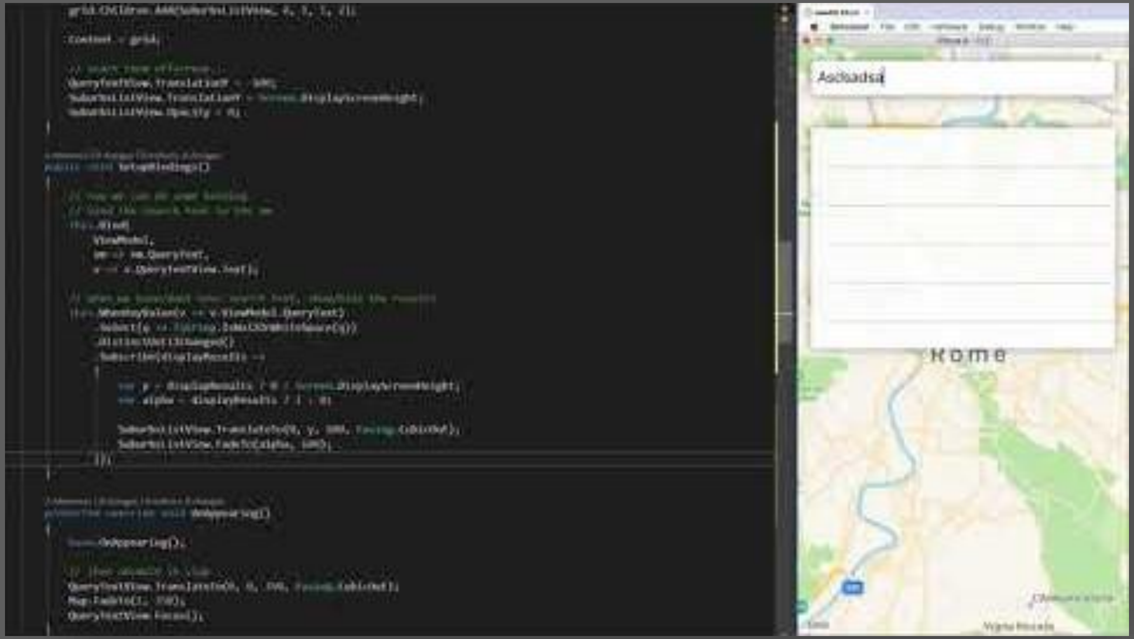
```csharp
// DSL-style helpers for the signup controls
public Entry Entry(string placeholder, string bindTo, bool isPassword = false)
    => new Xamarin.Forms.Entry { Placeholder = placeholder, IsPassword = isPassword }
        .Bind(bindTo);

public Button Button(string text, Command command)
    => new Xamarin.Forms.Button { Text = text, Command = command }
        .Bind(sourcePropertyName: nameof(IsBusy),
                targetProperty: IsEnabledProperty,
                converter: BoolNotConverter.Instance);

public Label Validation(string bindTo)
    => new Label { TextColor = Color.Red }
    .FontSize(12)
    .Bind(bindTo)
    .Bind(sourcePropertyName: bindTo, targetProperty: IsVisibleProperty,
        converter: new FuncConverter<string, bool>(x => !String.IsNullOrWhiteSpace(x)));

public StackLayout Stack(params View[] args)
     => new StackLayout { }
         .Invoke(sl => args.ToList().ForEach(sl.Children.Add));
```



Username
A
Username must be at least 5 characters long
Email Address
Invalid
Email address is not valid
Password
●●●
Password must be at least 8 characters long
Confirm Password
●
Passwords do not match

code-based hot reload is gr8

- Iterate on UI (obviously)
- Iterate on animations and transitions
- Iterate on viewmodels, services etc.



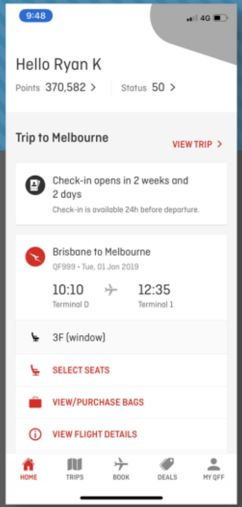
```
grid.Children.Add(SuburbsListView, 0, 1, 1, 2);

Content = grid;

// [illegible]
QueryTextView.TranslationY = -100;
SuburbsListView.TranslationY = Screen.DisplayScreenHeight;
SuburbsListView.Opacity = 0;
}

public void SetupBindings()
{
    // [illegible]
    // [illegible]
    this.Bind(
        ViewModel,
        vm => vm.QueryText,
        v => v.QueryTextView.Text);

    // [illegible]
    this.WhenAnyValue(v => v.ViewModel.QueryText)
        .Select(q => !string.IsNullOrWhiteSpace(q))
        .DistinctUntilChanged()
        .Subscribe(displayResults =>
        {
            var y = displayResults ? 0 : Screen.DisplayScreenHeight;
            var alpha = displayResults ? 1 : 0;

            SuburbsListView.TranslateTo(0, y, 300, Easing.CubicOut);
            SuburbsListView.FadeTo(alpha, 300);
        });
}

protected override void OnAppearing()
{
    base.OnAppearing();

    // [illegible]
    QueryTextView.TranslateTo(0, 0, 250, Easing.CubicOut);
    Map.FadeTo(1, 250);
    QueryTextView.Focus();
```

Live Reload with Continuous
https://www.youtube.com/watch?v=RMMccK_OI9w

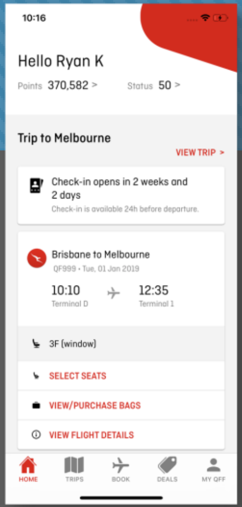
wrapping up

useful resources

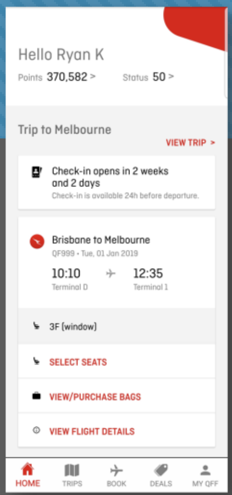
- **CSharpForMarkup**
  https://github.com/VincentH-Net/CSharpForMarkup
- **VincentH on twitter**
  https://twitter.com/vincenth_net
- **Shipping app using CSharpForMarkup**
  http://www.youtube.com/watch?v=50N1LL_Txe8
- **Twitch – Xamarin.Forms C# Markup with Ryan Davis**
  https://www.twitch.tv/videos/441875218
- **Xappy Login Page Code**
  https://github.com/rdavisau/Xappy/
- **Xamarin.Forms 4.0 Challenge Submissions** (code-based)
  https://ryandavis.io/xamarin-forms-4-0-challenge-submissions/



Source (iOS iPhone X)

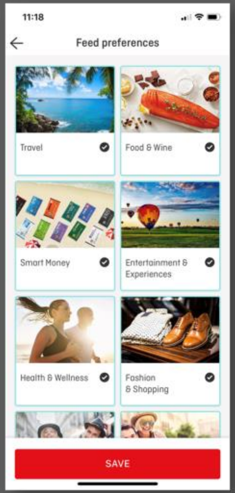


Visual (iOS iPhone X)

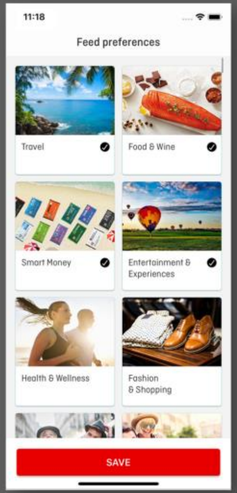


Visual (Android S9)

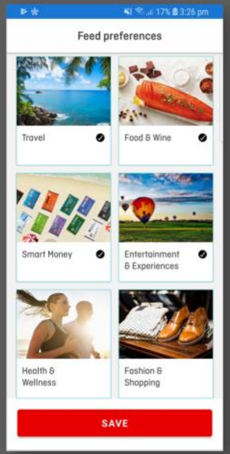
Visual Challenge



Source (iOS iPhone X)



XF4 (iOS iPhone X)



XF4 (Android Samsung S9)

CollectionView Challenge

questions