FREE eBook

LEARNING Xamarin.Forms

Free unaffiliated eBook created from
Stack Overflow contributors.

#xamarin.fo

rms

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: xamarin-forms

It is an unofficial and free Xamarin.Forms ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Xamarin.Forms.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with Xamarin.Forms

Remarks

Xamarin.Forms makes it possible to create iOS, Android, and Windows apps with large amounts of shared code, including UI code or XAML UI markup. App pages and views are mapped to native controls on each platform, but can be customized to provide platform-specific UI or to access platform-specific features.

Versions

Version	Release Date
2.3.1	2016-08-03
2.3.0-hotfix1	2016-06-29
2.3.0	2016-06-16
2.2.0-hotfix1	2016-05-30
2.2.0	2016-04-27
2.1.0	2016-03-13
2.0.1	2016-01-20
2.0.0	2015-11-17
1.5.1	2016-10-20
1.5.0	2016-09-25
1.4.4	2015-07-27
1.4.3	2015-06-30
1.4.2	2015-04-21
1.4.1	2015-03-30
1.4.0	2015-03-09
1.3.5	2015-03-02

Version	Release Date
1.3.4	2015-02-17
1.3.3	2015-02-09
1.3.2	2015-02-03
1.3.1	2015-01-04
1.3.0	2014-12-24
1.2.3	2014-10-02
1.2.2	2014-07-30
1.2.1	2014-07-14
1.2.0	2014-07-11
1.1.1	2014-06-19
1.1.0	2014-06-12
1.0.1	2014-06-04

Examples

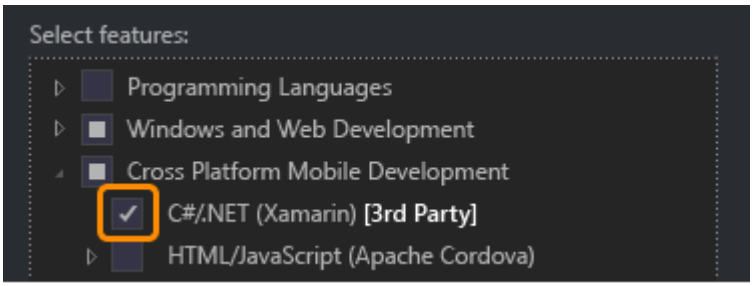
Installation (Visual Studio)

Xamarin.Forms is a cross-platform natively backed UI toolkit abstraction that allows developers to easily create user interfaces that can be shared across Android, iOS, Windows, and Windows Phone. The user interfaces are rendered using the native controls of the target platform, allowing Xamarin.Forms applications to retain the appropriate look and feel for each platform.

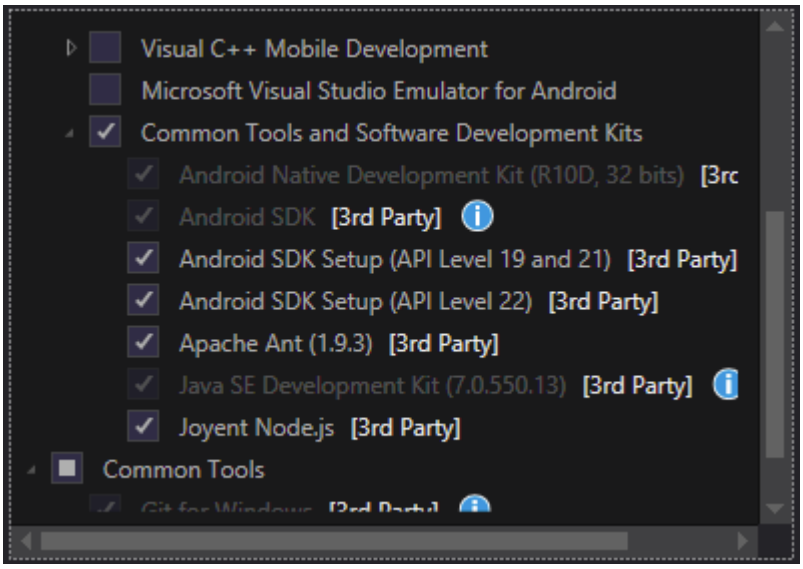
Xamarin Plugin for Visual Studio

To get started with Xamarin.Forms for Visual Studio you need to have the Xamarin plugin itself. The easiest way to have it installed is to download and install the latest Visual Studio.

If you already have the latest Visual Studio installed, go to Control Panel > Programs and Features, right click on Visual Studio, and click Change. When the installer opens, click on Modify, and select the cross-platform mobile development tools:



You can also select to install the Android SDK:



Uncheck it if you already have the SDK installed. You will be able to setup Xamarin to use existing Android SDK later.

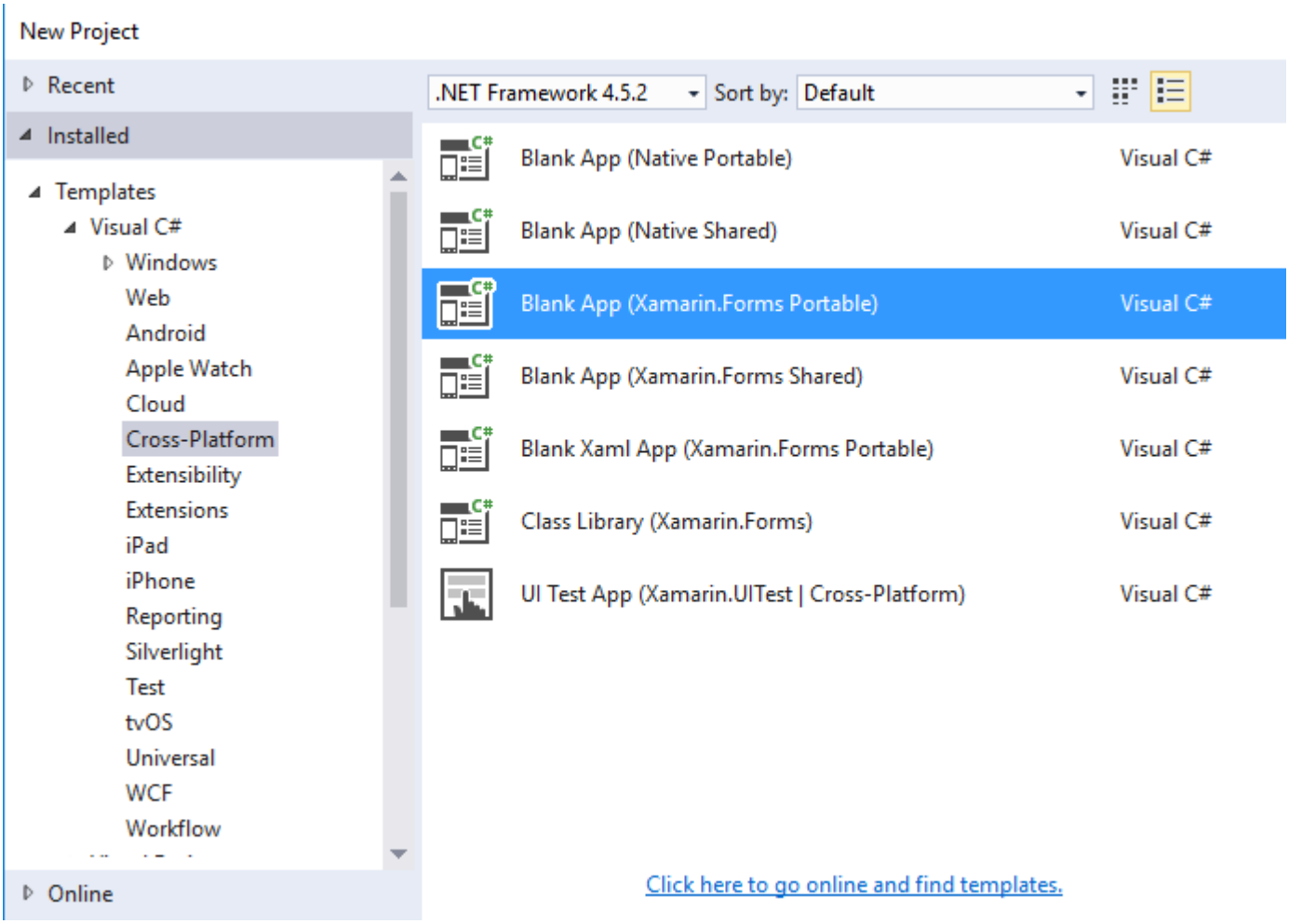
Xamarin.Forms

Xamarin.Forms is a set of libraries for your Portable Class library and native assemblies. The Xamarin.Forms library itself is available as a NuGet package. To add it to your project just use the regular `Install-Package` command of the Package Manager Console:

```
Install-Package Xamarin.Forms
```

for all of your initial assemblies (for example MyProject, MyProject.Droid and MyProject.iOS).

The easiest way to get started with Xamarin.Forms is to create an empty project in Visual Studio:



As you can see there are 2 available options to create the blank app -- Portable and Shared. I recommend you to get started with Portable one because it's the most commonly used in the real world (differences and more explanation to be added).

After creating the project make sure you're using the latest Xamarin.Forms version as your initial template may contain the old one. Use your Package Manager Console or Manage NuGet Packages option to upgrade to the latest Xamarin.Forms (remember it's just a NuGet package).

While the Visual Studio Xamarin.Forms templates will create an iOS platform project for you, you will need to connect Xamarin to a Mac build host to be able to run these projects on the iOS Simulator or physical devices.

Hello World Xamarin Forms: Visual Studio

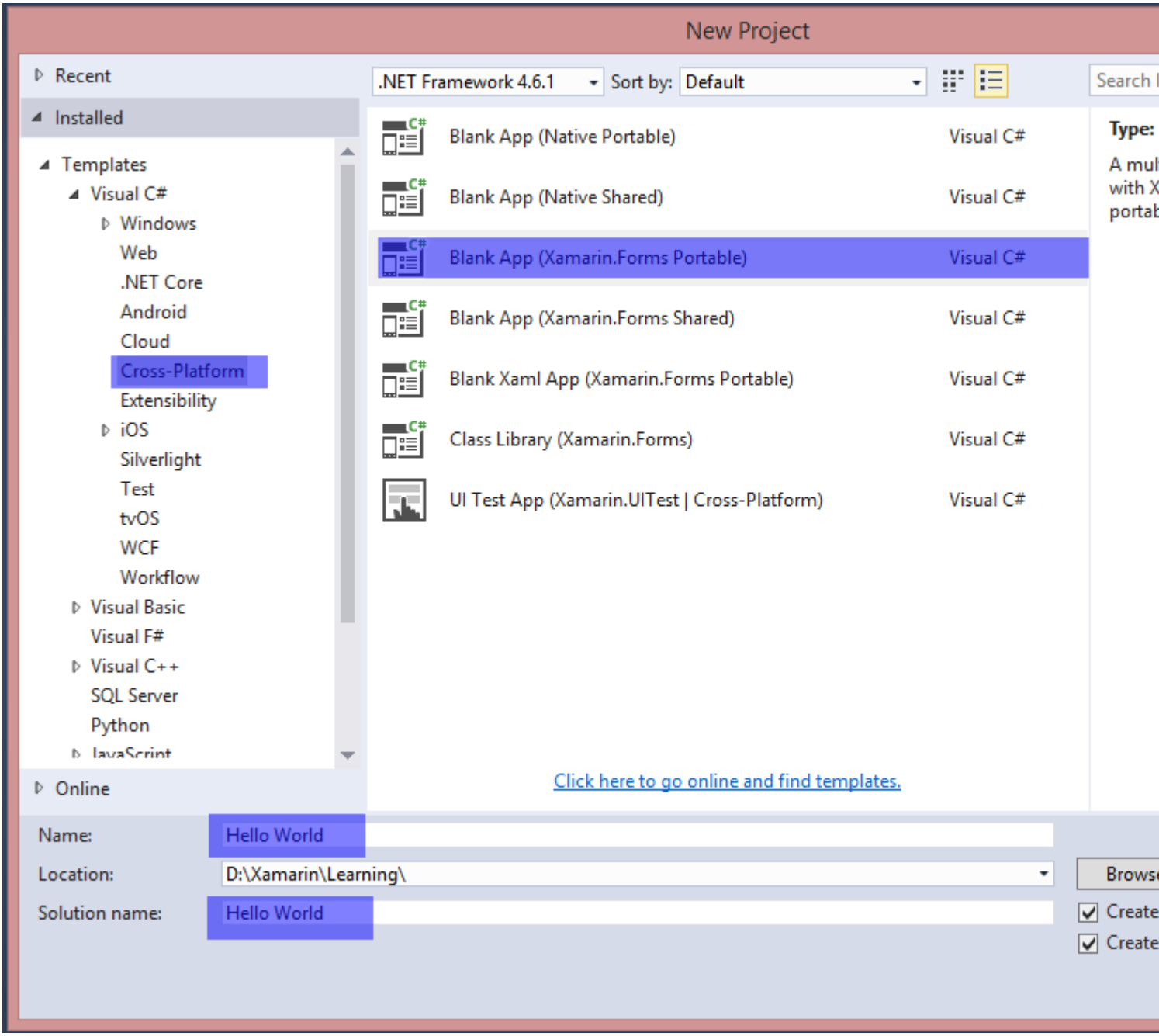
After successfully installing Xamarin as described in the first example, it's time to launch the first sample application.

Step 1: Creating a new Project.

In Visual Studio, choose New -> Project -> Visual C# -> Cross-Platform -> Blank App (Xamarin.Forms Portable)

Name the app "Hello World" and select the location to create the project and click OK. This will create a solution for you which contains three projects:

1. HelloWorld (this is where your logic and views is placed, i.e. the portable project)
2. HelloWorld.Droid (the Android project)
3. HelloWorld.iOS (the iOS project)



Step 2: Investigating the sample

Having created the solution, a sample application will be ready to be deployed. Open the `App.cs` located in the root of the portable project and investigate the code. As seen below, the `Content`s of the sample is a `StackLayout` which contains a `Label`:

```
using Xamarin.Forms;

namespace Hello_World
{
    public class App : Application
    {
        public App()
        {
            // The root page of your application
            MainPage = new ContentPage
            {
                Content = new StackLayout
                {
                    VerticalOptions = LayoutOptions.Center,
                    Children = {
                        new Label {
                            HorizontalTextAlignment = TextAlignment.Center,
                            Text = "Welcome to Xamarin Forms!"
                        }
                    }
                }
            };
        }
        protected override void OnStart()
        {
            // Handle when your app starts
        }
        protected override void OnSleep()
        {
            // Handle when your app sleeps
        }
        protected override void OnResume()
        {
            // Handle when your app resumes
        }
    }
}
```

Step 3: Launching the application

Now simply right-click the project you want to start (`HelloWorld.Droid` or `HelloWorld.iOS`) and click `Set as StartUp Project`. Then, in the Visual Studio toolbar, click the `Start` button (the green triangular button that resembles a Play button) to launch the application on the targeted simulator/emulator.

Read Getting started with Xamarin.Forms online: https://riptutorial.com/xamarin-forms/topic/908/getting-started-with-xamarin-forms

Chapter 2: Accessing native features with DependencyService

Remarks

If you do not want your code to break when no implementation is found, check the `DependencyService` first if it has a implementation available.

You can do this by a simple check if it is not `null`.

```
var speaker = DependencyService.Get<ITextToSpeech>();

if (speaker != null)
{
    speaker.Speak("Ready for action!");
}
```

or, if your IDE supports C# 6, with null-conditional operator:

```
var speaker = DependencyService.Get<ITextToSpeech>();

speaker?.Speak("Ready for action!");
```

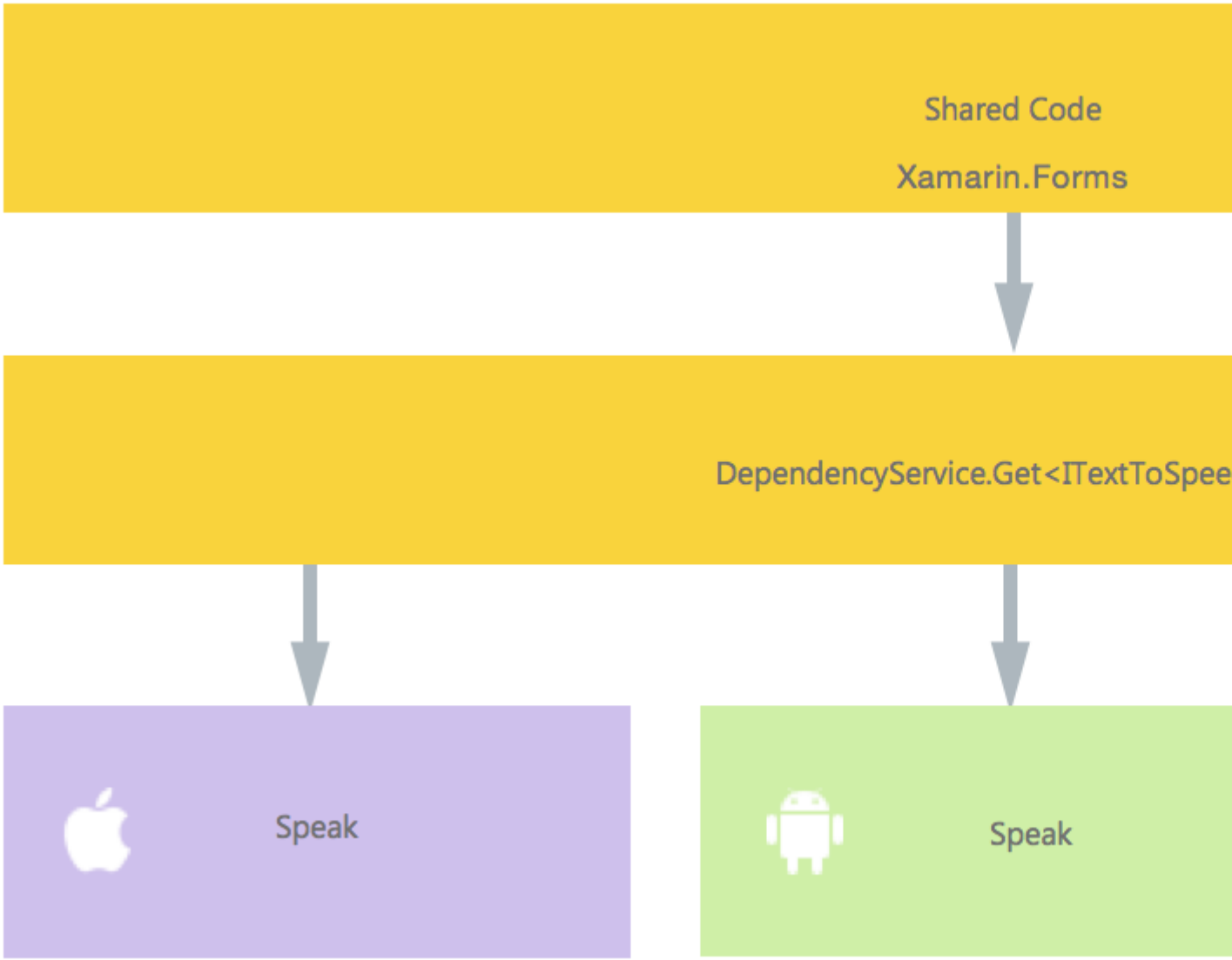
If you don't do this and no implementation is found at runtime, your code will generate an exception.

Examples

Implementing text-to-speech

A good example of a feature that request platform specific code is when you want to implement text-to-speech (tts). This example assumes that you are working with shared code in a PCL library.

A schematic overview of our solution would look like the image underneath.



In our shared code we define an interface which is registered with the `DependencyService`. This is where we will do our calls upon. Define an interface like underneath.

```
public interface ITextToSpeech
{
    void Speak (string text);
}
```

Now in each specific platform, we need to create an implementation of this interface. Let's start with the iOS implementation.

iOS Implementation

```
using AVFoundation;

public class TextToSpeechImplementation : ITextToSpeech
{
```

```
    public TextToSpeechImplementation () {}

    public void Speak (string text)
    {
        var speechSynthesizer = new AVSpeechSynthesizer ();

        var speechUtterance = new AVSpeechUtterance (text) {
            Rate = AVSpeechUtterance.MaximumSpeechRate/4,
            Voice = AVSpeechSynthesisVoice.FromLanguage ("en-US"),
            Volume = 0.5f,
            PitchMultiplier = 1.0f
        };

        speechSynthesizer.SpeakUtterance (speechUtterance);
    }
}
```

In the code example above you notice that there is specific code to iOS. Like types such as `AVSpeechSynthesizer`. These would not work in shared code.

To register this implementation with the Xamarin `DependencyService` add this attribute above the namespace declaration.

```
using AVFoundation;
using DependencyServiceSample.iOS;//enables registration outside of namespace

[assembly: Xamarin.Forms.Dependency (typeof (TextToSpeechImplementation))]
namespace DependencyServiceSample.iOS {
    public class TextToSpeechImplementation : ITextToSpeech
//... Rest of code
```

Now when you do a call like this in your shared code, the right implementation for the platform you are running your app on is injected.

`DependencyService.Get<ITextToSpeech>()`. More on this later on.

Android Implementation

The Android implementation of this code would look like underneath.

```
using Android.Speech.Tts;
using Xamarin.Forms;
using System.Collections.Generic;
using DependencyServiceSample.Droid;

public class TextToSpeechImplementation : Java.Lang.Object, ITextToSpeech,
TextToSpeech.IOnInitListener
{
    TextToSpeech speaker;
    string toSpeak;

    public TextToSpeechImplementation () {}

    public void Speak (string text)
```

```
    {
        var ctx = Forms.Context; // useful for many Android SDK features
        toSpeak = text;
        if (speaker == null) {
            speaker = new TextToSpeech (ctx, this);
        } else {
            var p = new Dictionary<string,string> ();
            speaker.Speak (toSpeak, QueueMode.Flush, p);
        }
    }

    #region IOnInitListener implementation
    public void OnInit (OperationResult status)
    {
        if (status.Equals (OperationResult.Success)) {
            var p = new Dictionary<string,string> ();
            speaker.Speak (toSpeak, QueueMode.Flush, p);
        }
    }
    #endregion
}
```

Again don't forget to register it with the `DependencyService`.

```
using Android.Speech.Tts;
using Xamarin.Forms;
using System.Collections.Generic;
using DependencyServiceSample.Droid;


[assembly: Xamarin.Forms.Dependency (typeof (TextToSpeechImplementation))]
namespace DependencyServiceSample.Droid{
    //... Rest of code
```

Windows Phone Implementation

Finally, for Windows Phone this code can be used.

```
public class TextToSpeechImplementation : ITextToSpeech
{
    public TextToSpeechImplementation() {}

    public async void Speak(string text)
    {
        MediaElement mediaElement = new MediaElement();

        var synth = new Windows.Media.SpeechSynthesis.SpeechSynthesizer();

        SpeechSynthesisStream stream = await synth.SynthesizeTextToStreamAsync("Hello World");

        mediaElement.SetSource(stream, stream.ContentType);
        mediaElement.Play();
        await synth.SynthesizeTextToStreamAsync(text);
    }
}
```

And once more do not forget to register it.

```
using Windows.Media.SpeechSynthesis;
using Windows.UI.Xaml.Controls;
using DependencyServiceSample.WinPhone;//enables registration outside of namespace

[assembly: Xamarin.Forms.Dependency (typeof (TextToSpeechImplementation))]
namespace DependencyServiceSample.WinPhone{
    //... Rest of code
```

Implementing in Shared Code

Now everything is in place to make it work! Finally, in your shared code you can now call this function by using the interface. At runtime, the implementation will be injected which corresponds to the current platform it is running on.

In this code you will see a page that could be in a Xamarin Forms project. It creates a button which invokes the `Speak()` method by using the `DependencyService`.

```
public MainPage ()
{
    var speak = new Button {
        Text = "Hello, Forms !",
        VerticalOptions = LayoutOptions.CenterAndExpand,
        HorizontalOptions = LayoutOptions.CenterAndExpand,
    };
    speak.Clicked += (sender, e) => {
        DependencyService.Get<ITextToSpeech>().Speak("Hello from Xamarin Forms");
    };
    Content = speak;
}
```

The result will be that when the app is ran and the button is clicked, the text provided will be spoken.

All of this without having to do hard stuff like compiler hints and such. You now have one uniform way of accessing platform specific functionality through platform independent code.

Getting Application and Device OS Version Numbers - Android & iOS - PCL

The example below will collect the Device's OS version number and the the version of the application (which is defined in each projects' properties) that is entered into **Version name** on Android and **Version** on iOS.

First make an interface in your PCL project:

```
public interface INativeHelper {
    /// <summary>
    /// On iOS, gets the <c>CFBundleVersion</c> number and on Android, gets the
<c>PackageInfo</c>'s <c>VersionName</c>, both of which are specified in their respective
project properties.
```

```
    /// </summary>
    /// <returns><c>string</c>, containing the build number.</returns>
    string GetAppVersion();

    /// <summary>
    /// On iOS, gets the <c>UIDevice.CurrentDevice.SystemVersion</c> number and on Android,
gets the <c>Build.VERSION.Release</c>.
    /// </summary>
    /// <returns><c>string</c>, containing the OS version number.</returns>
    string GetOsVersion();
}
```

Now we implement the interface in the Android and iOS projects.

Android:

```
[assembly: Dependency(typeof(NativeHelper_Android))]

namespace YourNamespace.Droid{
    public class NativeHelper_Android : INativeHelper {

        /// <summary>
        /// See interface summary.
        /// </summary>
        public string GetAppVersion() {
            Context context = Forms.Context;
            return context.PackageManager.GetPackageInfo(context.PackageName, 0).VersionName;
        }

        /// <summary>
        /// See interface summary.
        /// </summary>
        public string GetOsVersion() { return Build.VERSION.Release; }
    }
}
```

iOS:

```
[assembly: Dependency(typeof(NativeHelper_iOS))]

namespace YourNamespace.iOS {
    public class NativeHelper_iOS : INativeHelper {

        /// <summary>
        /// See interface summary.
        /// </summary>
        public string GetAppVersion() { return
Foundation.NSBundle.MainBundle.InfoDictionary[new
Foundation.NSString("CFBundleVersion")].ToString(); }

        /// <summary>
        /// See interface summary.
        /// </summary>
        public string GetOsVersion() { return UIDevice.CurrentDevice.SystemVersion; }
    }
}
```

Now to use the code in a method:

```
public string GetOsAndAppVersion {
    INativeHelper helper = DependencyService.Get<INativeHelper>();

    if(helper != null) {
        string osVersion  = helper.GetOsVersion();
        string appVersion = helper.GetBuildNumber()
    }
}
```

Read Accessing native features with DependencyService online: https://riptutorial.com/xamarin-forms/topic/2409/accessing-native-features-with-dependencyservice

Chapter 3: AppSettings Reader in Xamarin.Forms

Examples

Reading app.config file in a Xamarin.Forms Xaml project

While each mobile platforms do offer their own settings management api, there are no built in ways to read settings from a good old .net style app.config xml file; This is due to a bunch of good reasons, notably the .net framework configuration management api being on the heavyweight side, and each platform having their own file system api.

So we built a simple PCLAppConfig library, nicely nuget packaged for your immediate consumption.

This library makes use of the lovely PCLStorage library

This example assumes you are developing a Xamarin.Forms Xaml project, where you would need to access settings from your shared viewmodel.

1. Initialize ConfigurationManager.AppSettings on each of your platform project, just after the 'Xamarin.Forms.Forms.Init' statement, as per below:

iOS (AppDelegate.cs)

```
global::Xamarin.Forms.Forms.Init();
ConfigurationManager.Initialise(PCLAppConfig.FileSystemStream.PortableStream.Current);
LoadApplication(new App());
```

Android (MainActivity.cs)

```
global::Xamarin.Forms.Forms.Init(this, bundle);
ConfigurationManager.Initialise(PCLAppConfig.FileSystemStream.PortableStream.Current);
LoadApplication(new App());
```

UWP / Windows 8.1 / WP 8.1 (App.xaml.cs)

```
Xamarin.Forms.Forms.Init(e);
ConfigurationManager.Initialise(PCLAppConfig.FileSystemStream.PortableStream.Current);
```

2. Add an app.config file to your shared PCL project, and add your appSettings entries, as you would do with any app.config file

```
<configuration>
    <appSettings>
        <add key="config.text" value="hello from app.settings!" />
    </appSettings>
```

```
</configuration>
```

3. Add this PCL app.config file **as a linked file** on all your platform projects. For android, make sure to set the build action to **'AndroidAsset'**, for UWP set the build action to **'Content'**

4. Access your setting: `ConfigurationManager.AppSettings["config.text"];`

Read AppSettings Reader in Xamarin.Forms online: https://riptutorial.com/xamarin-forms/topic/5911/appsettings-reader-in-xamarin-forms

Chapter 4: BDD Unit Testing in Xamarin.Forms

Remarks

- The DI Container/resolver we use internally in this library is Autofac.
- The testing framework is NUnit 3x.
- You should be able to use this library with any Xamarin.Forms framework
- Source and example project available here

Examples

Simple Specflow to test commands and navigation with NUnit Test Runner

Why do we need this?

The current way to do unit testing in Xamarin.Forms is via a platform runner, so your test will have to run within an ios, android, windows or mac UI environment : Running Tests in the IDE

Xamarin also provides awesome UI testing with the Xamarin.TestCloud offering, but when wanting to implement BDD dev practices, and have the ability to test ViewModels and Commands, while running cheaply on a unit test runners locally or on a build server, there is not built in way.

I developed a library that allows to use Specflow with Xamarin.Forms to easily implement your features from your Scenarios definitions up to the ViewModel, independently of any MVVM framework used for the App (such as XLabs, MVVMCross, Prism)

If you are new to BDD, check Specflow out.

Usage:

- If you don't have it yet, install the specflow visual studio extension from here (or from you visual studio IDE): https://visualstudiogallery.msdn.microsoft.com/c74211e7-cb6e-4dfa-855d-df0ad4a37dd6
- Add a Class library to your Xamarin.Forms project. That's your test project.
- Add SpecFlow.Xamarin.Forms package from nuget to your test projects.
- Add a class to you test project that inherits 'TestApp', and register your views/viewmodels pairs as well as adding any DI registration, as per below:

```
public class DemoAppTest : TestApp
{
    protected override void SetViewModelMapping()
    {
        TestViewFactory.EnableCache = false;

        // register your views / viewmodels below
        RegisterView<MainPage, MainViewModel>();
    }

    protected override void InitialiseContainer()
    {
        // add any di registration here
        // Resolver.Instance.Register<TInterface, TType>();
        base.InitialiseContainer();
    }
}
```

- Add a SetupHook class to your test project, in order to add you Specflow hooks. You will need to bootstrap the test application as per below, providing the class you created above, and the your app initial viewmodel:

```
[Binding]
public class SetupHooks : TestSetupHooks
{
    /// <summary>
    ///     The before scenario.
    /// </summary>
    [BeforeScenario]
    public void BeforeScenario()
    {
        // bootstrap test app with your test app and your starting viewmodel
        new TestAppBootstrap().RunApplication<DemoAppTest, MainViewModel>();
    }
}
```

- You will need to add a catch block to your xamarin.forms views codebehind in order to ignore xamarin.forms framework forcing you to run the app ui (something we dont want to do):

```
    public YourView()
    {
        try
        {
            InitializeComponent();
        }
        catch (InvalidOperationException soe)
        {
            if (!soe.Message.Contains("MUST"))
                throw;
        }
    }
```

- Add a specflow feature to your project (using the vs specflow templates shipped with the vs specflow extension)

- Create/Generate a step class that inherits TestStepBase, passing the scenarioContext parameter to the base.
- Use the navigation services and helpers to navigate, execute commands, and test your view models:

```
  [Binding]
    public class GeneralSteps : TestStepBase
    {
        public GeneralSteps(ScenarioContext scenarioContext)
            : base(scenarioContext)
        {
            // you need to instantiate your steps by passing the scenarioContext to the base
        }

        [Given(@"I am on the main view")]
        public void GivenIAmOnTheMainView()
        {
            Resolver.Instance.Resolve<INavigationService>().PushAsync<MainViewModel>();


Resolver.Instance.Resolve<INavigationService>().CurrentViewModelType.ShouldEqualType<MainViewModel>();

        }

        [When(@"I click on the button")]
        public void WhenIClickOnTheButton()
        {
            GetCurrentViewModel<MainViewModel>().GetTextCommand.Execute(null);
        }

        [Then(@"I can see a Label with text ""(.*)""")]
        public void ThenICanSeeALabelWithText(string text)
        {
            GetCurrentViewModel<MainViewModel>().Text.ShouldEqual(text);
        }
    }
```

Advanced Usage for MVVM

To add to the first example, in order to test navigation statements that occurs within the application, we need to provide the ViewModel with a hook to the Navigation. To achieve this:

- Add the package SpecFlow.Xamarin.Forms.IViewModel from nuget to your PCL Xamarin.Forms project
- Implement the IViewModel interface in your ViewModel. This will simply expose the Xamarin.Forms INavigation property:
- `public class MainViewModel : INotifyPropertyChanged, IViewModel.IViewModel { public INavigation Navigation { get; set; }`
- The test framework will pick that up and manage internal navigation
- You can use any MVVM frameworks for you application (such as XLabs, MVVMCross, Prism to name a few. As long as the IViewModel interface is implemented in your ViewModel, the framework will pick it up.

Read BDD Unit Testing in Xamarin.Forms online: https://riptutorial.com/xamarin-

forms/topic/6172/bdd-unit-testing-in-xamarin-forms

Chapter 5: Caching

Examples

Caching using Akavache

About Akavache

Akavache is an incredibly useful library providing reach functionality of caching your data. Akavache provides a key-value storage interface and works on the top of SQLite3. You do not need to keep your schema synced as it's actually No-SQL solution which makes it perfect for most of the mobile applications especially if you need your app to be updated often without data loss.

Recommendations for Xamarin

Akavache is definetly the best caching library for Xamarin application if only you do not need to operate with strongly relative data, binary or really big amounts of data. Use Akavache in the following cases:

- You need your app to cache the data for a given period of time (you can configure expiration timeout for each entity being saved;
- You want your app to work offline;
- It's hard to determine and freeze the schema of your data. For example, you have lists containing different typed objects;
- It's enough for you to have simple key-value access to the data and you do not need to make complex queries.

Akavache is not a "silver bullet" for data storage so think twice about using it in the following cases:

- Your data entities have many relations between each other;
- You don't really need your app to work offline;
- You have huge amount of data to be saved locally;
- You need to migrate your data from version to version;
- You need to perform complex queries typical for SQL like grouping, projections etc.

  Actually you can manually migrate your data just by reading and writing it back with updated fields.

Simple example

Interacting with Akavache is primarily done through an object called `BlobCache`.

Most of the Akavache's methods returns reactive observables, but you also can just await them thanks to extension methods.

```
using System.Reactive.Linq;   // IMPORTANT - this makes await work!

// Make sure you set the application name before doing any inserts or gets
BlobCache.ApplicationName = "AkavacheExperiment";

var myToaster = new Toaster();
await BlobCache.UserAccount.InsertObject("toaster", myToaster);

//
// ...later, in another part of town...
//

// Using async/await
var toaster = await BlobCache.UserAccount.GetObject<Toaster>("toaster");

// or without async/await
Toaster toaster;

BlobCache.UserAccount.GetObject<Toaster>("toaster")
    .Subscribe(x => toaster = x, ex => Console.WriteLine("No Key!"));
```

Error handling

```
Toaster toaster;

try {
    toaster = await BlobCache.UserAccount.GetObjectAsync("toaster");
} catch (KeyNotFoundException ex) {
    toaster = new Toaster();
}

// Or without async/await:
toaster = await BlobCache.UserAccount.GetObjectAsync<Toaster>("toaster")
    .Catch(Observable.Return(new Toaster()));
```

Read Caching online: https://riptutorial.com/xamarin-forms/topic/3644/caching

Chapter 6: CarouselView - Pre-release version

Remarks

CarouselView is a Xamarin Control which can contains any kind of View. This pre-release control can only be used in Xamarin Forms projects.

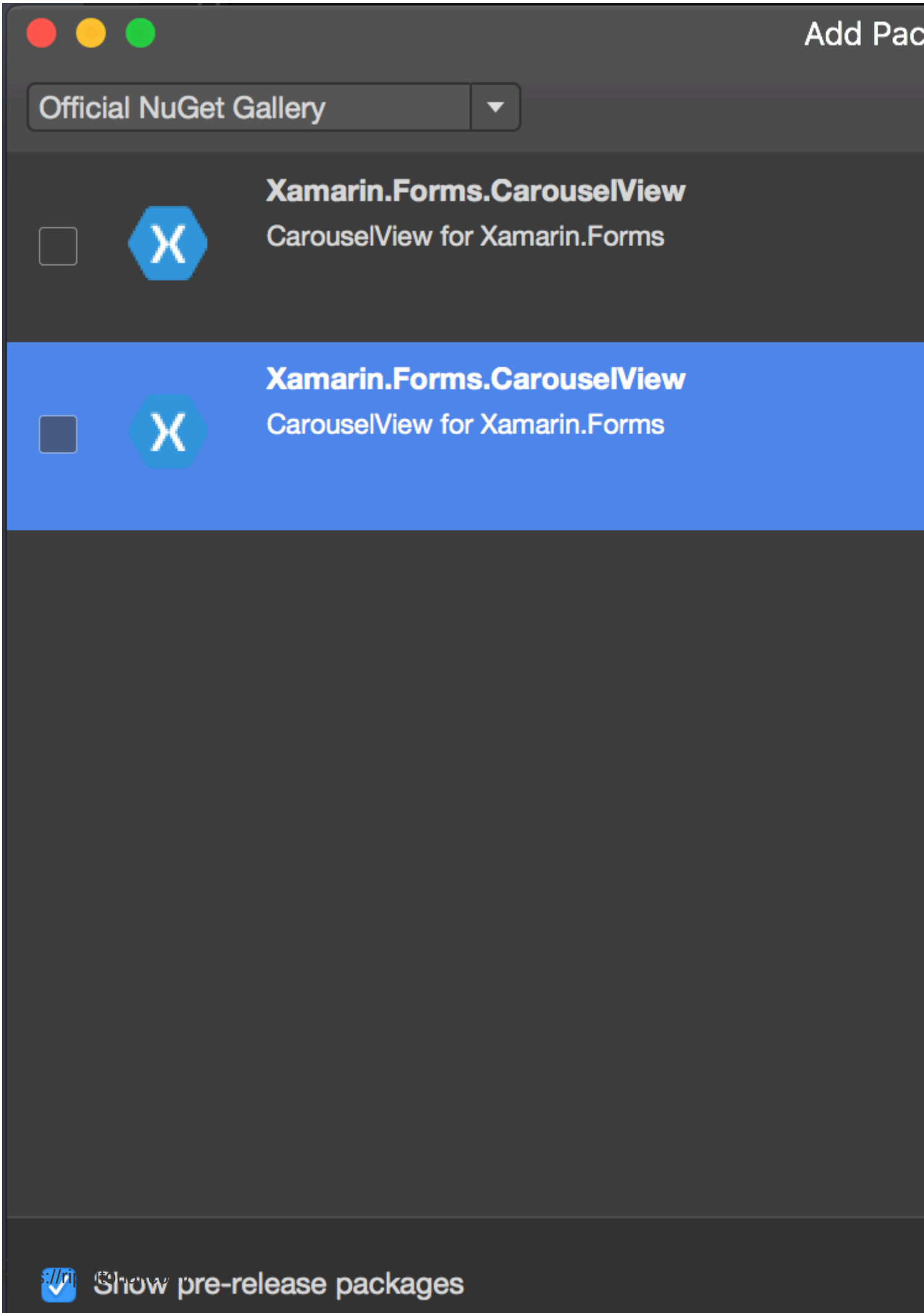
In the example provided by James Montemagno, on the blog of Xamarin, CarouselView is used to display images.

At this moment CarouselView is not integrated in Xamarin.Forms. To use this in your project(s), you will have to add the NuGet-Package (see example above).

Examples

Import CarouselView

The easiest way to import CarouselView is to use the NuGet-Packages Manager in Xamarin / Visual studio:


Add Pac
Official NuGet Gallery
Xamarin.Forms.CarouselView
CarouselView for Xamarin.Forms
Xamarin.Forms.CarouselView
CarouselView for Xamarin.Forms
Show pre-release packages

https://riptutorial.com/xamarin-forms/topic/6094/carouselview---pre-release-version

Chapter 7: Contact Picker - Xamarin Forms (Android and iOS)

Remarks

Contact Picker XF (Android and iOS)

Examples

contact_picker.cs

```
using System;

using Xamarin.Forms;

namespace contact_picker
{
    public class App : Application
    {
        public App ()
        {
            // The root page of your application
            MainPage = new MyPage();
        }

        protected override void OnStart ()
        {
            // Handle when your app starts
        }

        protected override void OnSleep ()
        {
            // Handle when your app sleeps
        }

        protected override void OnResume ()
        {
            // Handle when your app resumes
        }
    }
}
```

MyPage.cs

```
using System;

using Xamarin.Forms;

namespace contact_picker
{
    public class MyPage : ContentPage
```

```
    {
        Button button;
        public MyPage ()
        {
            button = new Button {
                Text = "choose contact"
            };

            button.Clicked += async (object sender, EventArgs e) => {

                    if (Device.OS == TargetPlatform.iOS) {
                        await Navigation.PushModalAsync (new ChooseContactPage ());
                        }
                    else if (Device.OS == TargetPlatform.Android)
                    {
                        MessagingCenter.Send (this, "android_choose_contact", "number1");
                    }


            };

            Content = new StackLayout {
                Children = {
                    new Label { Text = "Hello ContentPage" },
                    button
                }
            };
        }

        protected override void OnSizeAllocated (double width, double height)
        {
            base.OnSizeAllocated (width, height);

            MessagingCenter.Subscribe<MyPage, string> (this, "num_select", (sender, arg) => {
                DisplayAlert ("contact", arg, "OK");
            });

        }
    }
}
```

ChooseContactPicker.cs

```
using System;
using Xamarin.Forms;

namespace contact_picker
{
    public class ChooseContactPage : ContentPage
    {
        public ChooseContactPage ()
        {

        }
    }
}
```

ChooseContactActivity.cs

```
using Android.App;
using Android.OS;
using Android.Content;
using Android.Database;
using Xamarin.Forms;

namespace contact_picker.Droid
{

    [Activity (Label = "ChooseContactActivity")]

    public class ChooseContactActivity : Activity
    {
        public string type_number = "";
        protected override void OnCreate (Bundle savedInstanceState)
        {

            base.OnCreate (savedInstanceState);

            Intent intent = new Intent(Intent.ActionPick,
Android.Provider.ContactsContract.CommonDataKinds.Phone.ContentUri);
            StartActivityForResult(intent, 1);
        }

        protected override void OnActivityResult (int requestCode, Result resultCode, Intent
data)
        {
            // TODO Auto-generated method stub

            base.OnActivityResult (requestCode, resultCode, data);
            if (requestCode == 1) {
                if (resultCode == Result.Ok) {

                    Android.Net.Uri contactData = data.Data;
                    ICursor cursor = ContentResolver.Query(contactData, null, null, null,
null);

                    cursor.MoveToFirst();

                    string number =
cursor.GetString(cursor.GetColumnIndexOrThrow(Android.Provider.ContactsContract.CommonDataKinds.Phone.N



                    var twopage_renderer = new MyPage();
                    MessagingCenter.Send<MyPage, string> (twopage_renderer, "num_select",
number);
                    Finish ();
                    Xamarin.Forms.Application.Current.MainPage.Navigation.PopModalAsync ();


                }
                else if (resultCode == Result.Canceled)
                {
                    Finish ();
                }
            }
        }
    }
}
```

MainActivity.cs

```
using System;

using Android.App;
using Android.Content;
using Android.Content.PM;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;
using Xamarin.Forms;

namespace contact_picker.Droid
{
    [Activity (Label = "contact_picker.Droid", Icon = "@drawable/icon", MainLauncher = true,
ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)]
    public class MainActivity :
global::Xamarin.Forms.Platform.Android.FormsApplicationActivity
    {
        protected override void OnCreate (Bundle bundle)
        {
            base.OnCreate (bundle);

            global::Xamarin.Forms.Forms.Init (this, bundle);

            LoadApplication (new App ());


            MessagingCenter.Subscribe<MyPage, string>(this, "android_choose_contact", (sender,
args) => {
                Intent i = new Intent (Android.App.Application.Context,
typeof(ChooseContactActivity));
                i.PutExtra ("number1", args);
                StartActivity (i);
            });
        }
    }
}
```

ChooseContactRenderer.cs

```
using UIKit;
using AddressBookUI;
using Xamarin.Forms;
using Xamarin.Forms.Platform.iOS;
using contact_picker;
using contact_picker.iOS;


[assembly: ExportRenderer (typeof(ChooseContactPage), typeof(ChooseContactRenderer))]

namespace contact_picker.iOS
{
    public partial class ChooseContactRenderer : PageRenderer
    {
        ABPeoplePickerNavigationController _contactController;
```

```
        public string type_number;

        protected override void OnElementChanged (VisualElementChangedEventArgs e)
        {
            base.OnElementChanged (e);

            var page = e.NewElement as ChooseContactPage;

            if (e.OldElement != null || Element == null) {
                return;

            }

        }

        public override void ViewDidLoad ()
        {
            base.ViewDidLoad ();

            _contactController = new ABPeoplePickerNavigationController ();

            this.PresentModalViewController (_contactController, true); //display contact
chooser


            _contactController.Cancelled += delegate {
                Xamarin.Forms.Application.Current.MainPage.Navigation.PopModalAsync ();

                this.DismissModalViewController (true); };

            _contactController.SelectPerson2 += delegate(object sender,
ABPeoplePickerSelectPerson2EventArgs e) {

                var getphones = e.Person.GetPhones();
                string number = "";

                if (getphones == null)
                {
                    number = "Nothing";
                }
                else if (getphones.Count > 1)
                {
                    //il ya plus de 2 num de telephone
                    foreach(var t in getphones)
                    {
                        number = t.Value + "/" + number;
                    }
                }
                else if (getphones.Count == 1)
                {
                    //il ya 1 num de telephone
                    foreach(var t in getphones)
                    {
                        number = t.Value;
                    }
                }


                Xamarin.Forms.Application.Current.MainPage.Navigation.PopModalAsync ();
```

```
                var twopage_renderer = new MyPage();
                MessagingCenter.Send<MyPage, string> (twopage_renderer, "num_select", number);
                this.DismissModalViewController (true);



            };
        }

        public override void ViewDidUnload ()
        {
            base.ViewDidUnload ();

            // Clear any references to subviews of the main view in order to
            // allow the Garbage Collector to collect them sooner.
            //
            // e.g. myOutlet.Dispose (); myOutlet = null;

            this.DismissModalViewController (true);
        }

        public override bool ShouldAutorotateToInterfaceOrientation (UIInterfaceOrientation
toInterfaceOrientation)
        {
            // Return true for supported orientations
            return (toInterfaceOrientation != UIInterfaceOrientation.PortraitUpsideDown);
        }
    }
}
```

Read Contact Picker - Xamarin Forms (Android and iOS) online: https://riptutorial.com/xamarin-forms/topic/6659/contact-picker---xamarin-forms--android-and-ios-

Chapter 8: Creating custom controls

Examples

Create an Xamarin Forms custom input control (no native required)

Below is an example of a pure Xamarin Forms custom control. No custom rendering is being done for this but could easily be implemented, in fact, in my own code, I use this very same control along with a custom renderer for both the `Label` and `Entry`.

The custom control is a `ContentView` with a `Label`, `Entry`, and a `BoxView` within it, held in place using 2 `StackLayout`s. We also define multiple bindable properties as well as a `TextChanged` event.

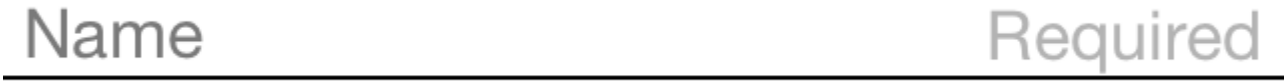
The custom bindable properties work by being defined as they are below and having the elements within the control (in this case a `Label` and an `Entry`) being bound to the custom bindable properties. A few on the bindable properties need to also implement a `BindingPropertyChangedDelegate` in order to make the bounded elements change their values.

```
public class InputFieldContentView : ContentView {

    #region Properties

    /// <summary>
    /// Attached to the <c>InputFieldContentView</c>'s <c>ExtendedEntryOnTextChanged()</c>
event, but returns the <c>sender</c> as <c>InputFieldContentView</c>.
    /// </summary>
    public event System.EventHandler<TextChangedEventArgs> OnContentViewTextChangedEvent; //In
OnContentViewTextChangedEvent() we return our custom InputFieldContentView control as the
sender but we could have returned the Entry itself as the sender if we wanted to do that
instead.

    public static readonly BindableProperty LabelTextProperty =
BindableProperty.Create("LabelText", typeof(string), typeof(InputFieldContentView),
string.Empty);

    public string LabelText {
        get { return (string)GetValue(LabelTextProperty); }
        set { SetValue(LabelTextProperty, value); }
    }

    public static readonly BindableProperty LabelColorProperty =
BindableProperty.Create("LabelColor", typeof(Color), typeof(InputFieldContentView),
Color.Default);

    public Color LabelColor {
        get { return (Color)GetValue(LabelColorProperty); }
        set { SetValue(LabelColorProperty, value); }
    }

    public static readonly BindableProperty EntryTextProperty =
BindableProperty.Create("EntryText", typeof(string), typeof(InputFieldContentView),
string.Empty, BindingMode.TwoWay, null, OnEntryTextChanged);

    public string EntryText {
```

```
        get { return (string)GetValue(EntryTextProperty); }
        set { SetValue(EntryTextProperty, value); }
    }

    public static readonly BindableProperty PlaceholderTextProperty =
BindableProperty.Create("PlaceholderText", typeof(string), typeof(InputFieldContentView),
string.Empty);

    public string PlaceholderText {
        get { return (string)GetValue(PlaceholderTextProperty); }
        set { SetValue(PlaceholderTextProperty, value); }
    }

    public static readonly BindableProperty UnderlineColorProperty =
BindableProperty.Create("UnderlineColor", typeof(Color), typeof(InputFieldContentView),
Color.Black, BindingMode.TwoWay, null, UnderlineColorChanged);

    public Color UnderlineColor {
        get { return (Color)GetValue(UnderlineColorProperty); }
        set { SetValue(UnderlineColorProperty, value); }
    }

    private BoxView _underline;

    #endregion

    public InputFieldContentView() {

        BackgroundColor   = Color.Transparent;
        HorizontalOptions = LayoutOptions.FillAndExpand;

        Label label = new Label {
            BindingContext    = this,
            HorizontalOptions = LayoutOptions.StartAndExpand,
            VerticalOptions   = LayoutOptions.Center,
            TextColor         = Color.Black
        };

        label.SetBinding(Label.TextProperty, (InputFieldContentView view) => view.LabelText,
BindingMode.TwoWay);
        label.SetBinding(Label.TextColorProperty, (InputFieldContentView view) =>
view.LabelColor, BindingMode.TwoWay);

        Entry entry = new Entry {
            BindingContext          = this,
            HorizontalOptions       = LayoutOptions.End,
            TextColor               = Color.Black,
            HorizontalTextAlignment = TextAlignment.End
        };

        entry.SetBinding(Entry.PlaceholderProperty, (InputFieldContentView view) =>
view.PlaceholderText, BindingMode.TwoWay);
        entry.SetBinding(Entry.TextProperty, (InputFieldContentView view) => view.EntryText,
BindingMode.TwoWay);

        entry.TextChanged += OnTextChangedEvent;

        _underline = new BoxView {
            BackgroundColor   = Color.Black,
            HeightRequest     = 1,
            HorizontalOptions = LayoutOptions.FillAndExpand
```

```
        };

        Content = new StackLayout {
            Spacing           = 0,
            HorizontalOptions = LayoutOptions.FillAndExpand,
            Children          = {
                new StackLayout {
                    Padding           = new Thickness(5, 0),
                    Spacing           = 0,
                    HorizontalOptions = LayoutOptions.FillAndExpand,
                    Orientation       = StackOrientation.Horizontal,
                    Children          = { label, entry }
                }, _underline
            }
        };

        SizeChanged += (sender, args) => entry.WidthRequest = Width * 0.5 - 10;
    }

    private static void OnEntryTextChanged(BindableObject bindable, object oldValue, object
newValue) {
        InputFieldContentView contentView = (InputFieldContentView)bindable;
        contentView.EntryText             = (string)newValue;
    }

    private void OnTextChangedEvent(object sender, TextChangedEventArgs args) {
        if(OnContentViewTextChangedEvent != null) { OnContentViewTextChangedEvent(this, new
TextChangedEventArgs(args.OldTextValue, args.NewTextValue)); } //Here is where we pass in
'this' (which is the InputFieldContentView) instead of 'sender' (which is the Entry control)
    }

    private static void UnderlineColorChanged(BindableObject bindable, object oldValue, object
newValue) {
        InputFieldContentView contentView      = (InputFieldContentView)bindable;
        contentView._underline.BackgroundColor = (Color)newValue;
    }
}
```

And here is a picture of the final product on iOS (the image shows what it looks like when using a custom renderer for the `Label` and `Entry` which is being used to remove the border on iOS and to specify a custom font for both elements):



One issue I ran into was getting the `BoxView.BackgroundColor` to change when `UnderlineColor` changed. Even after binding the `BoxView`'s `BackgroundColor` property, it would not change until I added the `UnderlineColorChanged` delegate.

Label with bindable collection of Spans

I created custom label with wrapper around `FormattedText` property:

```
public class MultiComponentLabel : Label
{
    public IList<TextComponent> Components { get; set; }
```

```
    public MultiComponentLabel()
    {
        var components = new ObservableCollection<TextComponent>();
        components.CollectionChanged += OnComponentsChanged;
        Components = components;
    }

    private void OnComponentsChanged(object sender, NotifyCollectionChangedEventArgs e)
    {
        BuildText();
    }

    private void OnComponentPropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
    {
        BuildText();
    }

    private void BuildText()
    {
        var formattedString = new FormattedString();
        foreach (var component in Components)
        {
            formattedString.Spans.Add(new Span { Text = component.Text });
            component.PropertyChanged -= OnComponentPropertyChanged;
            component.PropertyChanged += OnComponentPropertyChanged;
        }

        FormattedText = formattedString;
    }
}
```

I added collection of custom `TextComponent`s:

```
public class TextComponent : BindableObject
{
    public static readonly BindableProperty TextProperty =
        BindableProperty.Create(nameof(Text),
                                typeof(string),
                                typeof(TextComponent),
                                default(string));

    public string Text
    {
        get { return (string)GetValue(TextProperty); }
        set { SetValue(TextProperty, value); }
    }
}
```

And when collection of text components changes or `Text` property of separate component changes I rebuild `FormattedText` property of base `Label`.

And how I used it in `XAML`:

```
<ContentPage x:Name="Page"
         xmlns="http://xamarin.com/schemas/2014/forms"
         xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
```

```
         xmlns:controls="clr-namespace:SuperForms.Controls;assembly=SuperForms.Controls"
         x:Class="SuperForms.Samples.MultiComponentLabelPage">
<controls:MultiComponentLabel Margin="0,20,0,0">
    <controls:MultiComponentLabel.Components>
      <controls:TextComponent Text="Time"/>
      <controls:TextComponent Text=": "/>
      <controls:TextComponent Text="{Binding CurrentTime, Source={x:Reference Page}}"/>
    </controls:MultiComponentLabel.Components>
  </controls:MultiComponentLabel>
</ContentPage>
```

Codebehind of page:

```
public partial class MultiComponentLabelPage : ContentPage
{
    private string _currentTime;

    public string CurrentTime
    {
        get { return _currentTime; }
        set
        {
            _currentTime = value;
            OnPropertyChanged();
        }
    }

    public MultiComponentLabelPage()
    {
        InitializeComponent();
        BindingContext = this;
    }

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

        Device.StartTimer(TimeSpan.FromSeconds(1), () =>
        {
            CurrentTime = DateTime.Now.ToString("hh : mm : ss");
            return true;
        });
    }
}
```

Creating a custom Entry control with a MaxLength property

The Xamarin Forms `Entry` control does not have a `MaxLength` property. To achieve this you can extend `Entry` as below, by adding a Bindable `MaxLength` property. Then you just need to subscribe to the `TextChanged` event on `Entry` and validate the length of the `Text` when this is called:

```
class CustomEntry : Entry
{
    public CustomEntry()
    {
        base.TextChanged += Validate;
    }
```

```
    public static readonly BindableProperty MaxLengthProperty =
BindableProperty.Create(nameof(MaxLength), typeof(int), typeof(CustomEntry), 0);

    public int MaxLength
    {
        get { return (int)GetValue(MaxLengthProperty); }
        set { SetValue(MaxLengthProperty, value); }
    }

    public void Validate(object sender, TextChangedEventArgs args)
    {
        var e = sender as Entry;
        var val = e?.Text;

        if (string.IsNullOrEmpty(val))
            return;

        if (MaxLength > 0 && val.Length > MaxLength)
            val = val.Remove(val.Length - 1);

        e.Text = val;
    }
}
```

Usage in XAML:

```
   <ContentView xmlns="http://xamarin.com/schemas/2014/forms"
         xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
         xmlns:customControls="clr-namespace:CustomControls;assembly=CustomControls"
         x:Class="Views.TestView">
<ContentView.Content>
    <customControls:CustomEntry MaxLength="10" />
</ContentView.Content>
```

Read Creating custom controls online: https://riptutorial.com/xamarin-forms/topic/3913/creating-custom-controls

Chapter 9: Creating custom controls

Introduction

Every `Xamarin.Forms` view has an accompanying renderer for each platform that creates an instance of a native control. When a View is rendered on the specific platform the `ViewRenderer` class is instantiated.

The process for doing this is as follows:

Create a Xamarin.Forms custom control.

Consume the custom control from Xamarin.Forms.

Create the custom renderer for the control on each platform.

Examples

Implementing a CheckBox Control

In this example we will implement a custom Checkbox for Android and iOS.

Creating the Custom Control

```
namespace CheckBoxCustomRendererExample
{
    public class Checkbox : View
    {
        public static readonly BindableProperty IsCheckedProperty =
BindableProperty.Create<Checkbox, bool>(p => p.IsChecked, true, propertyChanged: (s, o, n) =>
{ (s as Checkbox).OnChecked(new EventArgs()); });
        public static readonly BindableProperty ColorProperty =
BindableProperty.Create<Checkbox, Color>(p => p.Color, Color.Default);

        public bool IsChecked
        {
            get
            {
                return (bool)GetValue(IsCheckedProperty);
            }
            set
            {
                SetValue(IsCheckedProperty, value);
            }
        }

        public Color Color
        {
            get
            {
                return (Color)GetValue(ColorProperty);
```

```
            }
            set
            {
                SetValue(ColorProperty, value);
            }
        }

        public event EventHandler Checked;

        protected virtual void OnChecked(EventArgs e)
        {
            if (Checked != null)
                Checked(this, e);
        }
    }
}
```

We'll start off with the Android Custom Renderer by creating a new class (`CheckboxCustomRenderer`) in the `Android` portion of our solution.

A few important details to note:

- We need to mark the top of our class with the ExportRenderer attribute so that the renderer is registered with `Xamarin.Forms`. This way, `Xamarin.Forms` will use this renderer when it's trying to create our `Checkbox` object on `Android`.
- We're doing most of our work in the `OnElementChanged` method, where we instantiate and set up our native control.

Consuming the Custom Control

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" xmlns:local="clr-
namespace:CheckBoxCustomRendererExample"
x:Class="CheckBoxCustomRendererExample.CheckBoxCustomRendererExamplePage">
    <StackLayout Padding="20">
        <local:Checkbox Color="Aqua" />
    </StackLayout>
</ContentPage>
```

Creating the Custom Renderer on each Platform

The process for creating the custom renderer class is as follows:

1. Create a subclass of the `ViewRenderer<T1,T2>` class that renders the custom control. The first type argument should be the custom control the renderer is for, in this case `CheckBox`. The second type argument should be the native control that will implement the custom control.
2. Override the `OnElementChanged` method that renders the custom control and write logic to customize it. This method is called when the corresponding `Xamarin.Forms` control is created.
3. Add an `ExportRenderer` attribute to the custom renderer class to specify that it will be used to render the `Xamarin.Forms` custom control. This attribute is used to register the custom renderer with `Xamarin.Forms`.

Creating the Custom Renderer for Android

```
[assembly: ExportRenderer(typeof(Checkbox), typeof(CheckBoxRenderer))]
namespace CheckBoxCustomRendererExample.Droid
{
    public class CheckBoxRenderer : ViewRenderer<Checkbox, CheckBox>
    {
        private CheckBox checkBox;

        protected override void OnElementChanged(ElementChangedEventArgs<Checkbox> e)
        {
            base.OnElementChanged(e);
            var model = e.NewElement;
            checkBox = new CheckBox(Context);
            checkBox.Tag = this;
            CheckboxPropertyChanged(model, null);
            checkBox.SetOnClickListener(new ClickListener(model));
            SetNativeControl(checkBox);
        }
        private void CheckboxPropertyChanged(Checkbox model, String propertyName)
        {
            if (propertyName == null || Checkbox.IsCheckedProperty.PropertyName ==
propertyName)
            {
                checkBox.Checked = model.IsChecked;
            }

            if (propertyName == null || Checkbox.ColorProperty.PropertyName == propertyName)
            {
                int[][] states = {
                    new int[] { Android.Resource.Attribute.StateEnabled}, // enabled
                    new int[] {Android.Resource.Attribute.StateEnabled}, // disabled
                    new int[] {Android.Resource.Attribute.StateChecked}, // unchecked
                    new int[] { Android.Resource.Attribute.StatePressed}  // pressed
                };
                var checkBoxColor = (int)model.Color.ToAndroid();
                int[] colors = {
                    checkBoxColor,
                    checkBoxColor,
                    checkBoxColor,
                    checkBoxColor
                };
                var myList = new Android.Content.Res.ColorStateList(states, colors);
                checkBox.ButtonTintList = myList;

            }
        }

        protected override void OnElementPropertyChanged(object sender,
PropertyChangedEventArgs e)
        {
            if (checkBox != null)
            {
                base.OnElementPropertyChanged(sender, e);

                CheckboxPropertyChanged((Checkbox)sender, e.PropertyName);
            }
        }

        public class ClickListener : Java.Lang.Object, IOnClickListener
```

```
        {
            private Checkbox _myCheckbox;
            public ClickListener(Checkbox myCheckbox)
            {
                this._myCheckbox = myCheckbox;
            }
            public void OnClick(global::Android.Views.View v)
            {
                _myCheckbox.IsChecked = !_myCheckbox.IsChecked;
            }
        }
    }
}
```

Creating the Custom Renderer for iOS

Since in iOS the is no built in checkbox, we will create a `CheckBoxView` first a then create a renderer for our Xamarin.Forms checkbox.

The `CheckBoxView` is based in two images the checked_checkbox.png and the unchecked_checkbox.png so the property `Color` will be ignored.

The CheckBox view:

```
namespace CheckBoxCustomRendererExample.iOS
{
    [Register("CheckBoxView")]
    public class CheckBoxView : UIButton
    {
        public CheckBoxView()
        {
            Initialize();
        }

        public CheckBoxView(CGRect bounds)
            : base(bounds)
        {
            Initialize();
        }

        public string CheckedTitle
        {
            set
            {
                SetTitle(value, UIControlState.Selected);
            }
        }

        public string UncheckedTitle
        {
            set
            {
                SetTitle(value, UIControlState.Normal);
            }
        }

        public bool Checked
```

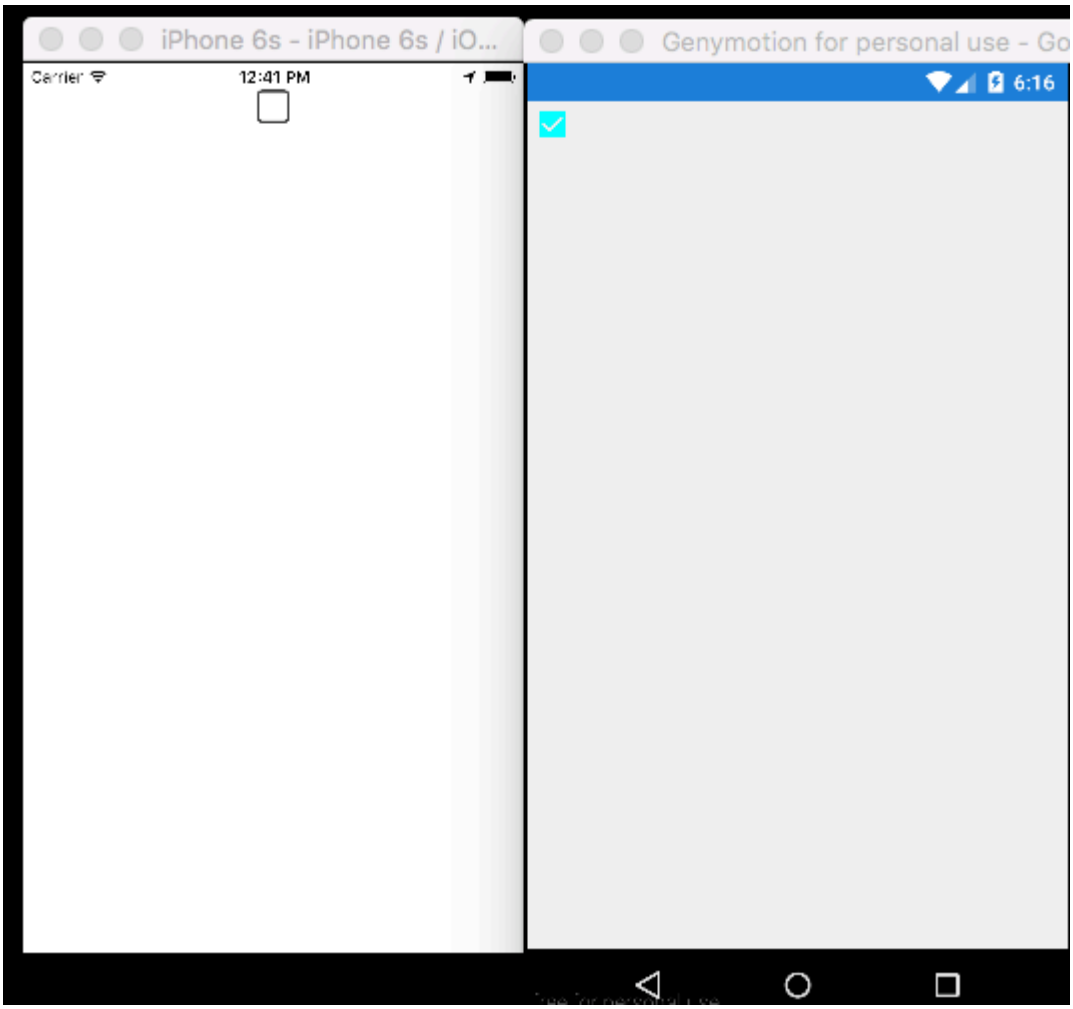
```
        {
            set { Selected = value; }
            get { return Selected; }
        }

        void Initialize()
        {
            ApplyStyle();

            TouchUpInside += (sender, args) => Selected = !Selected;
            // set default color, because type is not UIButtonType.System
            SetTitleColor(UIColor.DarkTextColor, UIControlState.Normal);
            SetTitleColor(UIColor.DarkTextColor, UIControlState.Selected);
        }

        void ApplyStyle()
        {
            SetImage(UIImage.FromBundle("Images/checked_checkbox.png"),
UIControlState.Selected);
            SetImage(UIImage.FromBundle("Images/unchecked_checkbox.png"),
UIControlState.Normal);
        }
    }
}
```

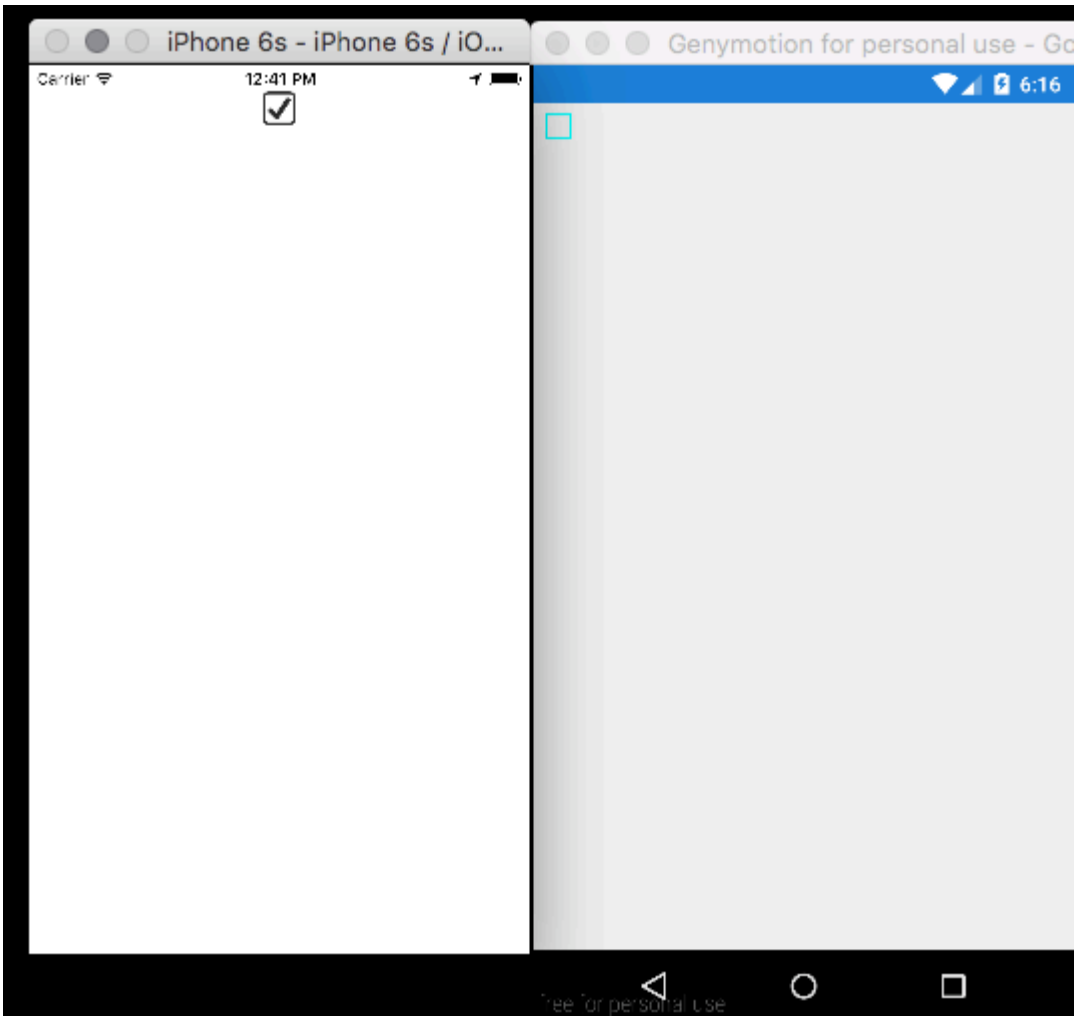
The CheckBox custom renderer:

```
[assembly: ExportRenderer(typeof(Checkbox), typeof(CheckBoxRenderer))]
namespace CheckBoxCustomRendererExample.iOS
{
    public class CheckBoxRenderer : ViewRenderer<Checkbox, CheckBoxView>
    {

        /// <summary>
        /// Handles the Element Changed event
        /// </summary>
        /// <param name="e">The e.</param>
        protected override void OnElementChanged(ElementChangedEventArgs<Checkbox> e)
        {
            base.OnElementChanged(e);

            if (Element == null)
                return;

            BackgroundColor = Element.BackgroundColor.ToUIColor();
            if (e.NewElement != null)
            {
                if (Control == null)
                {
                    var checkBox = new CheckBoxView(Bounds);
                    checkBox.TouchUpInside += (s, args) => Element.IsChecked =
Control.Checked;
                    SetNativeControl(checkBox);
                }
                Control.Checked = e.NewElement.IsChecked;
            }

            Control.Frame = Frame;
            Control.Bounds = Bounds;
```

```
        }

        /// <summary>
        /// Handles the <see cref="E:ElementPropertyChanged" /> event.
        /// </summary>
        /// <param name="sender">The sender.</param>
        /// <param name="e">The <see cref="PropertyChangedEventArgs"/> instance containing the
event data.</param>
        protected override void OnElementPropertyChanged(object sender,
PropertyChangedEventArgs e)
        {
            base.OnElementPropertyChanged(sender, e);

            if (e.PropertyName.Equals("Checked"))
            {
                Control.Checked = Element.IsChecked;
            }
        }
    }
}
```

Result:





Read Creating custom controls online: https://riptutorial.com/xamarin-forms/topic/5975/creating-custom-controls

Chapter 10: Creating custom controls

Examples

Creating custom Button

```
/// <summary>
/// Button with some additional options
/// </summary>
public class TurboButton : Button
{
    public static readonly BindableProperty StringDataProperty = BindableProperty.Create(
      propertyName: "StringData",
      returnType: typeof(string),
      declaringType: typeof(ButtonWithStorage),
      defaultValue: default(string));

    public static readonly BindableProperty IntDataProperty = BindableProperty.Create(
      propertyName: "IntData",
      returnType: typeof(int),
      declaringType: typeof(ButtonWithStorage),
      defaultValue: default(int));

    /// <summary>
    /// You can put here some string data
    /// </summary>
    public string StringData
    {
        get { return (string)GetValue(StringDataProperty); }
        set { SetValue(StringDataProperty, value); }
    }

    /// <summary>
    /// You can put here some int data
    /// </summary>
    public int IntData
    {
        get { return (int)GetValue(IntDataProperty); }
        set { SetValue(IntDataProperty, value); }
    }

    public TurboButton()
    {
        PropertyChanged += CheckIfPropertyLoaded;
    }

    /// <summary>
    /// Called when one of properties is changed
    /// </summary>
    private void CheckIfPropertyLoaded(object sender, PropertyChangedEventArgs e)
    {
        //example of using PropertyChanged
        if(e.PropertyName == "IntData")
        {
            //IntData is now changed, you can operate on updated value
        }
    }
```

```
}
```

Usage in XAML:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="SomeApp.Pages.SomeFolder.Example"
    xmlns:customControls="clr-namespace:SomeApp.CustomControls;assembly=SomeApp">
    <StackLayout>
        <customControls:TurboButton x:Name="exampleControl" IntData="2" StringData="Test" />
    </StackLayout>
</ContentPage>
```

Now, you can use your properties in c#:

```
exampleControl.IntData
```

Note that you need to specify by yourself where your TurboButton class is placed in your project. I've done it in this line:

```
xmlns:customControls="clr-namespace:SomeApp.CustomControls;assembly=SomeApp"
```

You can freely change "customControls" to some other name. It's up to you how you will call it.

Read Creating custom controls online: https://riptutorial.com/xamarin-forms/topic/6592/creating-custom-controls

Chapter 11: Custom Fonts in Styles

Remarks

Resources to look at:

- Xamarin Styles
- Using custom fonts on iOS and Android with Xamarin.Forms
- Custom Renderers
- Resource Dictionaries
- Attached Properties

Examples

Accessing custom Fonts in Syles

Xamarin.Forms provide great mechanism for styling your cross-platforms application with global styles.

In mobile world your application must be pretty and stand out from the other applications. One of this characters is Custom Fonts used in application.

With power support of XAML Styling in Xamarin.Forms just created base style for all labels with yours custom fonts.

To include custom fonts into you iOS and Android project follow the guide in Using custom fonts on iOS and Android with Xamarin.Forms post written by Gerald.

Declare Style in App.xaml file resource section. This make all styles globally visible.

From Gerald post above we need to use StyleId property but it isn't bindable property, so to using it in Style Setter we need to create Attachable Property for it:

```
public static class FontHelper
{
    public static readonly BindableProperty StyleIdProperty =
        BindableProperty.CreateAttached(
            propertyName: nameof(Label.StyleId),
            returnType: typeof(String),
            declaringType: typeof(FontHelper),
            defaultValue: default(String),
            propertyChanged: OnItemTappedChanged);

    public static String GetStyleId(BindableObject bindable) =>
(String)bindable.GetValue(StyleIdProperty);

    public static void SetStyleId(BindableObject bindable, String value) =>
bindable.SetValue(StyleIdProperty, value);
```

```
    public static void OnItemTappedChanged(BindableObject bindable, object oldValue, object
newValue)
    {
        var control = bindable as Element;
        if (control != null)
        {
            control.StyleId = GetStyleId(control);
        }
    }
}
```

Then add style in App.xaml resource:

```
<?xml version="1.0" encoding="utf-8" ?>
<Application xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:h="clr-namespace:My.Helpers"
             x:Class="My.App">

  <Application.Resources>

    <ResourceDictionary>
        <Style x:Key="LabelStyle" TargetType="Label">
            <Setter Property="FontFamily" Value="Metric Bold" />
            <Setter Property="h:FontHelper.StyleId" Value="Metric-Bold" />
        </Style>
    </ResourceDictionary>

  </Application.Resources>

</Application>
```

According to post above we need to create Custom Renderer for Label which inherits from LabelRenderer On Android platform.

```
internal class LabelExRenderer : LabelRenderer
{
    protected override void OnElementChanged(ElementChangedEventArgs<Label> e)
    {
        base.OnElementChanged(e);
        if (!String.IsNullOrEmpty(e.NewElement?.StyleId))
        {
            var font = Typeface.CreateFromAsset(Forms.Context.ApplicationContext.Assets,
e.NewElement.StyleId + ".ttf");
            Control.Typeface = font;
        }
    }
}
```

For iOS platform no custom renderers required.

Now you can obtain style in your`s page markup:

For specific label

```
<Label Text="Some text" Style={StaticResource LabelStyle} />
```

Or apply style to all labels on the page by creating Style Based on LabesStyle

```
<!-- language: xaml -->

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="My.MainPage">

  <ContentPage.Resources>

    <ResourceDictionary>
        <Style TargetType="Label" BasedOn={StaticResource LabelStyle}>
        </Style>
    </ResourceDictionary>

  </ContentPage.Resources>

  <Label Text="Some text" />

</ContentPage>
```

Read Custom Fonts in Styles online: https://riptutorial.com/xamarin-forms/topic/4854/custom-fonts-in-styles

Chapter 12: Custom Renderers

Examples

Custom renderer for ListView

Custom Renderers let developers customize the appearance and behavior of Xamarin.Forms controls on each platform. Developers could use features of native controls.

For example, we need to disable scroll in `ListView`. On iOS `ListView` is scrollable even if all items are placed on the screen and user shouldn't be able to scroll the list. `Xamarin.Forms.ListView` doesn't manage such setting. In this case, a renderer is coming to help.

Firstly, we should create custom control in PCL project, which will declare some required bindable property:

```
public class SuperListView : ListView
{
    public static readonly BindableProperty IsScrollingEnableProperty =
        BindableProperty.Create(nameof(IsScrollingEnable),
                                typeof(bool),
                                typeof(SuperListView),
                                true);

    public bool IsScrollingEnable
    {
        get { return (bool)GetValue(IsScrollingEnableProperty); }
        set { SetValue(IsScrollingEnableProperty, value); }
    }
}
```

Next step will be creating a renderer for each platform.

iOS:

```
[assembly: ExportRenderer(typeof(SuperListView), typeof(SuperListViewRenderer))]
namespace SuperForms.iOS.Renderers
{
    public class SuperListViewRenderer : ListViewRenderer
    {
        protected override void OnElementChanged(ElementChangedEventArgs<ListView> e)
        {
            base.OnElementChanged(e);

            var superListView = Element as SuperListView;
            if (superListView == null)
                return;


            Control.ScrollEnabled = superListView.IsScrollingEnable;
        }
    }
}
```

And Android(Android's list doesn't have scroll if all items are placed on the screen, so we will not disable scrolling, but still we are able to use native properties):

```
[assembly: ExportRenderer(typeof(SuperListView), typeof(SuperListViewRenderer))]
namespace SuperForms.Droid.Renderers
{
    public class SuperListViewRenderer : ListViewRenderer
    {
        protected override void
OnElementChanged(ElementChangedEventArgs<Xamarin.Forms.ListView> e)
        {
            base.OnElementChanged(e);

            var superListView = Element as SuperListView;
            if (superListView == null)
                return;
        }
    }
}
```

`Element` property of renderer is my `SuperListView` control from PCL project.

`Control` property of renderer is native control. `Android.Widget.ListView` for Android and `UIKit.UITableView` for iOS.

And how we will use it in `XAML`:

```
<ContentPage x:Name="Page"
         xmlns="http://xamarin.com/schemas/2014/forms"
         xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
         xmlns:controls="clr-namespace:SuperForms.Controls;assembly=SuperForms.Controls"
         x:Class="SuperForms.Samples.SuperListViewPage">


  <controls:SuperListView ItemsSource="{Binding Items, Source={x:Reference Page}}"
                          IsScrollingEnable="false"
                          Margin="20">
    <controls:SuperListView.ItemTemplate>
      <DataTemplate>
        <ViewCell>
          <Label Text="{Binding .}"/>
        </ViewCell>
      </DataTemplate>
    </controls:SuperListView.ItemTemplate>
  </controls:SuperListView>
</ContentPage>
```

`.cs` file of page:

```
public partial class SuperListViewPage : ContentPage
{
    private ObservableCollection<string> _items;

    public ObservableCollection<string> Items
    {
        get { return _items; }
        set
```

```
        {
            _items = value;
            OnPropertyChanged();
        }
    }

    public SuperListViewPage()
    {
        var list = new SuperListView();

        InitializeComponent();

        var items = new List<string>(10);
        for (int i = 1; i <= 10; i++)
        {
            items.Add($"Item {i}");
        }

        Items = new ObservableCollection<string>(items);
    }
}
```

Custom Renderer for BoxView

Custom Renderer help to allows to add new properties and render them differently in native platform that can not be otherwise does through shared code. In this example we will add radius and shadow to a boxview.

Firstly, we should create custom control in PCL project, which will declare some required bindable property:

```
namespace Mobile.Controls
{
    public class ExtendedBoxView : BoxView
    {
        /// <summary>
        /// Respresents the background color of the button.
        /// </summary>
        public static readonly BindableProperty BorderRadiusProperty =
BindableProperty.Create<ExtendedBoxView, double>(p => p.BorderRadius, 0);

        public double BorderRadius
        {
            get { return (double)GetValue(BorderRadiusProperty); }
            set { SetValue(BorderRadiusProperty, value); }
        }

        public static readonly BindableProperty StrokeProperty =
            BindableProperty.Create<ExtendedBoxView, Color>(p => p.Stroke, Color.Transparent);

        public Color Stroke
        {
            get { return (Color)GetValue(StrokeProperty); }
            set { SetValue(StrokeProperty, value); }
        }

        public static readonly BindableProperty StrokeThicknessProperty =
            BindableProperty.Create<ExtendedBoxView, double>(p => p.StrokeThickness, 0);
```

```
        public double StrokeThickness
        {
            get { return (double)GetValue(StrokeThicknessProperty); }
            set { SetValue(StrokeThicknessProperty, value); }
        }
    }
}
```

Next step will be creating a renderer for each platform.

iOS:

```
[assembly: ExportRenderer(typeof(ExtendedBoxView), typeof(ExtendedBoxViewRenderer))]
namespace Mobile.iOS.Renderers
{
 public class ExtendedBoxViewRenderer : VisualElementRenderer<BoxView>
 {
    public ExtendedBoxViewRenderer()
    {
    }

    protected override void OnElementChanged(ElementChangedEventArgs<BoxView> e)
    {
        base.OnElementChanged(e);
        if (Element == null)
            return;

        Layer.MasksToBounds = true;
        Layer.CornerRadius = (float)((ExtendedBoxView)this.Element).BorderRadius / 2.0f;
    }


    protected override void OnElementPropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
    {
        base.OnElementPropertyChanged(sender, e);
        if (e.PropertyName == ExtendedBoxView.BorderRadiusProperty.PropertyName)
        {
            SetNeedsDisplay();
        }
    }

    public override void Draw(CGRect rect)
    {
        ExtendedBoxView roundedBoxView = (ExtendedBoxView)this.Element;
        using (var context = UIGraphics.GetCurrentContext())
        {
            context.SetFillColor(roundedBoxView.Color.ToCGColor());
            context.SetStrokeColor(roundedBoxView.Stroke.ToCGColor());
            context.SetLineWidth((float)roundedBoxView.StrokeThickness);

            var rCorner = this.Bounds.Inset((int)roundedBoxView.StrokeThickness / 2,
(int)roundedBoxView.StrokeThickness / 2);

            nfloat radius = (nfloat)roundedBoxView.BorderRadius;
            radius = (nfloat)Math.Max(0, Math.Min(radius, Math.Max(rCorner.Height / 2,
rCorner.Width / 2)));

            var path = CGPath.FromRoundedRect(rCorner, radius, radius);
```

```
            context.AddPath(path);
            context.DrawPath(CGPathDrawingMode.FillStroke);
        }
    }

 }
}
```

Again you can customize however you want inside the draw method.

And same for Android:

```
[assembly: ExportRenderer(typeof(ExtendedBoxView), typeof(ExtendedBoxViewRenderer))]
namespace Mobile.Droid
{
    /// <summary>
    ///
    /// </summary>
    public class ExtendedBoxViewRenderer : VisualElementRenderer<BoxView>
    {
        /// <summary>
        ///
        /// </summary>
        public ExtendedBoxViewRenderer()
        {
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="e"></param>
        protected override void OnElementChanged(ElementChangedEventArgs<BoxView> e)
        {
            base.OnElementChanged(e);

            SetWillNotDraw(false);

            Invalidate();
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        protected override void OnElementPropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
        {
            base.OnElementPropertyChanged(sender, e);

            if (e.PropertyName == ExtendedBoxView.BorderRadiusProperty.PropertyName)
            {
                Invalidate();
            }
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="canvas"></param>
```

```
        public override void Draw(Canvas canvas)
        {
            var box = Element as ExtendedBoxView;
            base.Draw(canvas);
            Paint myPaint = new Paint();

            myPaint.SetStyle(Paint.Style.Stroke);
            myPaint.StrokeWidth = (float)box.StrokeThickness;
            myPaint.SetARGB(convertTo255ScaleColor(box.Color.A),
convertTo255ScaleColor(box.Color.R), convertTo255ScaleColor(box.Color.G),
convertTo255ScaleColor(box.Color.B));
            myPaint.SetShadowLayer(20, 0, 5, Android.Graphics.Color.Argb(100, 0, 0, 0));

            SetLayerType(Android.Views.LayerType.Software, myPaint);

            var number = (float)box.StrokeThickness / 2;
            RectF rectF = new RectF(
                        number, // left
                        number, // top
                        canvas.Width - number, // right
                        canvas.Height - number // bottom
                );


            var radius = (float)box.BorderRadius;
            canvas.DrawRoundRect(rectF, radius, radius, myPaint);
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="color"></param>
        /// <returns></returns>
        private int convertTo255ScaleColor(double color)
        {
            return (int) Math.Ceiling(color * 255);
        }
}
```

}

The XAML:

We first reference to our control with the namespace we defined earlier.

```
xmlns:Controls="clr-namespace:Mobile.Controls"
```

We then use the Control as follows and use properties defined at the beginning:

```
<Controls:ExtendedBoxView
    x:Name="search_boxview"
    Color="#444"
    BorderRadius="5"
    HorizontalOptions="CenterAndExpand"
    />
```

Accessing renderer from a native project

```
var renderer = Platform.GetRenderer(visualElement);

if (renderer == null)
{
    renderer = Platform.CreateRenderer(visualElement);
    Platform.SetRenderer(visualElement, renderer);
}

DoSomeThingWithRender(render); // now you can do whatever you want with render
```

Rounded label with a custom renderer for Frame (PCL & iOS parts)

First step : PCL part

```
using Xamarin.Forms;

namespace ProjectNamespace
{
    public class ExtendedFrame : Frame
    {
        /// <summary>
        /// The corner radius property.
        /// </summary>
        public static readonly BindableProperty CornerRadiusProperty =
            BindableProperty.Create("CornerRadius", typeof(double), typeof(ExtendedFrame),
0.0);

        /// <summary>
        /// Gets or sets the corner radius.
        /// </summary>
        public double CornerRadius
        {
            get { return (double)GetValue(CornerRadiusProperty); }
            set { SetValue(CornerRadiusProperty, value); }
        }
    }
}
```

Second step : iOS part

```
using ProjectNamespace;
using ProjectNamespace.iOS;
using Xamarin.Forms;
using Xamarin.Forms.Platform.iOS;

[assembly: ExportRenderer(typeof(ExtendedFrame), typeof(ExtendedFrameRenderer))]
namespace ProjectNamespace.iOS
{
    public class ExtendedFrameRenderer : FrameRenderer
    {
        protected override void OnElementChanged(ElementChangedEventArgs<Frame> e)
        {
            base.OnElementChanged(e);

            if (Element != null)
            {
                Layer.MasksToBounds = true;
                Layer.CornerRadius = (float)(Element as ExtendedFrame).CornerRadius;
```

```
            }
        }

        protected override void OnElementPropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
        {
            base.OnElementPropertyChanged(sender, e);

            if (e.PropertyName == ExtendedFrame.CornerRadiusProperty.PropertyName)
            {
                Layer.CornerRadius = (float)(Element as ExtendedFrame).CornerRadius;
            }
        }
    }
}
```

Third step : XAML code to call an ExtendedFrame

If you want to use it in a XAML part, don't forget to write this :

```
xmlns:controls="clr-namespace:ProjectNamespace;assembly:ProjectNamespace"
```

after

```
xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
```

Now, you can use the ExtendedFrame like this :

```
<controls:ExtendedFrame
    VerticalOptions="FillAndExpand"
    HorizontalOptions="FillAndExpand"
    BackgroundColor="Gray"
    CornerRadius="35.0">
    <Frame.Content>
        <Label
            Text="MyText"
            TextColor="Blue"/>
    </Frame.Content>
</controls:ExtendedFrame>
```

Rounded BoxView with selectable background color

First step : PCL part

```
public class RoundedBoxView : BoxView
{
    public static readonly BindableProperty CornerRadiusProperty =
        BindableProperty.Create("CornerRadius", typeof(double), typeof(RoundedEntry),
default(double));

    public double CornerRadius
    {
        get
        {
```

```
            return (double)GetValue(CornerRadiusProperty);
        }
        set
        {
            SetValue(CornerRadiusProperty, value);
        }
    }

    public static readonly BindableProperty FillColorProperty =
        BindableProperty.Create("FillColor", typeof(string), typeof(RoundedEntry),
default(string));

    public string FillColor
    {
        get
        {
            return (string) GetValue(FillColorProperty);
        }
        set
        {
            SetValue(FillColorProperty, value);
        }
    }
}
```

Second step : Droid part

```
[assembly: ExportRenderer(typeof(RoundedBoxView), typeof(RoundedBoxViewRenderer))]
namespace MyNamespace.Droid
{
    public class RoundedBoxViewRenderer : VisualElementRenderer<BoxView>
    {
        protected override void OnElementChanged(ElementChangedEventArgs<BoxView> e)
        {
            base.OnElementChanged(e);
            SetWillNotDraw(false);
            Invalidate();
        }

        protected override void OnElementPropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
        {
            base.OnElementPropertyChanged(sender, e);
            SetWillNotDraw(false);
            Invalidate();
        }

        public override void Draw(Canvas canvas)
        {
            var box = Element as RoundedBoxView;
            var rect = new Rect();
            var paint = new Paint
            {
                Color = Xamarin.Forms.Color.FromHex(box.FillColor).ToAndroid(),
                AntiAlias = true,
            };

            GetDrawingRect(rect);

            var radius = (float)(rect.Width() / box.Width * box.CornerRadius);
```

```
            canvas.DrawRoundRect(new RectF(rect), radius, radius, paint);
        }
    }
}
```

Third step : iOS part

```
[assembly: ExportRenderer(typeof(RoundedBoxView), typeof(RoundedBoxViewRenderer))]
namespace MyNamespace.iOS
{
    public class RoundedBoxViewRenderer : BoxRenderer
    {
        protected override void OnElementChanged(ElementChangedEventArgs<BoxView> e)
        {
            base.OnElementChanged(e);

            if (Element != null)
            {
                Layer.CornerRadius = (float)(Element as RoundedBoxView).CornerRadius;
                Layer.BackgroundColor = Color.FromHex((Element as
RoundedBoxView).FillColor).ToCGColor();
            }
        }

        protected override void OnElementPropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
        {
            base.OnElementPropertyChanged(sender, e);

            if (Element != null)
            {
                Layer.CornerRadius = (float)(Element as RoundedBoxView).CornerRadius;
                Layer.BackgroundColor = (Element as RoundedBoxView).FillColor.ToCGColor();
            }
        }
    }
}
```

Read Custom Renderers online: https://riptutorial.com/xamarin-forms/topic/2949/custom-renderers

Chapter 13: Data Binding

Remarks

Possible Exceptions

System.ArrayTypeMismatchException: Attempted to access an element as a type incompatible with the array.

This exception can occur when attempting to bind a collection to a non-bindable property when XAML pre-compilation is enabled. A common example is attempting to bind to `Picker.Items`. See below.

System.ArgumentException: Object of type 'Xamarin.Forms.Binding' cannot be converted to type 'System.String'.

This exception can occur when attempting to bind a collection to a non-bindable property when XAML pre-compilation is disabled. A common example is attempting to bind to `Picker.Items`. See below.

The `Picker.Items` Property Is Not Bindable

This code will cause an error:

```
<!-- BAD CODE: will cause an error -->
<Picker Items="{Binding MyViewModelItems}" SelectedIndex="0" />
```

The exception may be one of the following:

> System.ArrayTypeMismatchException: Attempted to access an element as a type incompatible with the array.

or

> System.ArgumentException: Object of type 'Xamarin.Forms.Binding' cannot be converted to type 'System.String'.

Specifically, the `Items` property is non-bindable. Solutions include creating your own custom control or using a third-party control, such as `BindablePicker` from FreshEssentials. After installing the FreshEssentials NuGet package in your project, the package's `BindablePicker` control with a

bindable `ItemsSource` property is available:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:fe="clr-namespace:FreshEssentials;assembly=FreshEssentials"
             xmlns:my="clr-namespace:MyAssembly;assembly=MyAssembly"
             x:Class="MyNamespace.MyPage">
  <ContentPage.BindingContext>
    <my:MyViewModel />
  </ContentPage.BindingContext>
  <ContentPage.Content>
    <fe:BindablePicker ItemsSource="{Binding MyViewModelItems}" SelectedIndex="0" />
  </ContentPage.Content>
</ContentPage>
```

Examples

Basic Binding to ViewModel

EntryPage.xaml:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:vm="clr-namespace:MyAssembly.ViewModel;assembly=MyAssembly"
             x:Class="MyAssembly.EntryPage">
  <ContentPage.BindingContext>
    <vm:MyViewModel />
  </ContentPage.BindingContext>
  <ContentPage.Content>
    <StackLayout VerticalOptions="FillAndExpand"
                 HorizontalOptions="FillAndExpand"
                 Orientation="Vertical"
                 Spacing="15">
      <Label Text="Name:" />
      <Entry Text="{Binding Name}" />
      <Label Text="Phone:" />
      <Entry Text="{Binding Phone}" />
      <Button Text="Save" Command="{Binding SaveCommand}" />
    </StackLayout>
  </ContentPage.Content>
</ContentPage>
```

MyViewModel.cs:

```
using System;
using System.ComponentModel;

namespace MyAssembly.ViewModel
{
    public class MyViewModel : INotifyPropertyChanged
    {
        private string _name = String.Empty;
        private string _phone = String.Empty;
```

```
        public string Name
        {
            get { return _name; }
            set
            {
                if (_name != value)
                {
                    _name = value;
                    OnPropertyChanged(nameof(Name));
                }
            }
        }

        public string Phone
        {
            get { return _phone; }
            set
            {
                if (_phone != value)
                {
                    _phone = value;
                    OnPropertyChanged(nameof(Phone));
                }
            }
        }

        public ICommand SaveCommand { get; private set; }

        public MyViewModel()
        {
            SaveCommand = new Command(SaveCommandExecute);
        }

        private void SaveCommandExecute()
        {

        }

        public event PropertyChangedEventHandler PropertyChanged;

        protected virtual void OnPropertyChanged(string propertyName)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

Read Data Binding online: https://riptutorial.com/xamarin-forms/topic/3915/data-binding

Chapter 14: Dependency Services

Remarks

Access platform specific API from PCL or Shared project.

Examples

Access Camera and Gallery.

(Access Code)[https://github.com/vDoers/vDoersCameraAccess]

Read Dependency Services online: https://riptutorial.com/xamarin-forms/topic/6127/dependency-services

Chapter 15: DependencyService

Remarks

When using `DependencyService` you typically need 3 parts:

- **Interface** - This defines the functions you wish to abstract.
- **Platform implementation** - A class within each platform specific project that implements the previously defined interface.
- **Registration** - Each platform specific implementation class has to be registered with the `DependencyService` through a metadata attribute. This enables the `DependencyService` to find your implementation at run time.

When using `DependencyService` you are required to provide an implementation for each platform you target. When an implementation is not provided the application will fail at run time.

Examples

Interface

The interface defines the behaviour that you want to expose through the `DependencyService`. One example usage of a `DependencyService` is a Text-To-Speech service. There is currently no abstraction for this feature in Xamarin.Forms, so you need to create your own. Start off by defining an interface for the behaviour:

```
public interface ITextToSpeech
{
       void Speak (string whatToSay);
}
```

Because we define our interface we can code against it from our shared code.

Note: Classes that implement the interface need to have a parameterless constructor to work with the `DependencyService`.

iOS implementation

The interface you defined needs to be implemented in every targeted platform. For iOS this is done through the `AVFoundation` framework. The following implementation of the `ITextToSpeech` interface handles speaking a given text in English.

```
using AVFoundation;

public class TextToSpeechiOS : ITextToSpeech
{
    public TextToSpeechiOS () {}
```

```
    public void Speak (string whatToSay)
    {
        var speechSynthesizer = new AVSpeechSynthesizer ();

        var speechUtterance = new AVSpeechUtterance (whatToSay) {
            Rate = AVSpeechUtterance.MaximumSpeechRate/4,
            Voice = AVSpeechSynthesisVoice.FromLanguage ("en-US"),
            Volume = 0.5f,
            PitchMultiplier = 1.0f
        };

        speechSynthesizer.SpeakUtterance (speechUtterance);
    }
}
```

When you've created your class you need to enable the `DependencyService` to discover it at run time. This is done by adding an `[assembly]` attribute above the class definition and outside of any namespace definitions.

```
using AVFoundation;
using DependencyServiceSample.iOS;

[assembly: Xamarin.Forms.Dependency (typeof (TextToSpeechiOS))]
namespace DependencyServiceSample.iOS {
    public class TextToSpeechiOS : ITextToSpeech
...
```

This attribute registers the class with the `DependencyService` so it can be used when an instance of the `ITextToSpeech` interface is needed.

Shared code

After you've created and registered your platform-specific classes you can start hooking them up to your shared code. The following page contains a button that triggers the text-to-speech functionality using a pre-defined sentence. It uses `DependencyService` to retrieve a platform-specific implementation of `ITextToSpeech` at run time using the native SDKs.

```
public MainPage ()
{
    var speakButton = new Button {
        Text = "Talk to me baby!",
        VerticalOptions = LayoutOptions.CenterAndExpand,
        HorizontalOptions = LayoutOptions.CenterAndExpand,
    };

    speakButton.Clicked += (sender, e) => {
        DependencyService.Get<ITextToSpeech>().Speak("Xamarin Forms likes eating cake by the
ocean.");
    };

    Content = speakButton;
}
```

When you run this application on an iOS or Android device and tap the button you will hear the

application speak the given sentence.

Android implementation

The Android specific implementation is a bit more complex because it forces you to inherit from a native `Java.Lang.Object` and forces you to implement the `IOnInitListener` interface. Android requires you to provide a valid Android context for a lot of the SDK methods it exposes. Xamarin.Forms exposes a `Forms.Context` object that provides you with a Android context that you can use in such cases.

```
using Android.Speech.Tts;
using Xamarin.Forms;
using System.Collections.Generic;
using DependencyServiceSample.Droid;

public class TextToSpeechAndroid : Java.Lang.Object, ITextToSpeech,
TextToSpeech.IOnInitListener
{
    TextToSpeech _speaker;

    public TextToSpeechAndroid () {}

    public void Speak (string whatToSay)
    {
        var ctx = Forms.Context;

        if (_speaker == null)
        {
            _speaker = new TextToSpeech (ctx, this);
        }
        else
        {
            var p = new Dictionary<string,string> ();
            _speaker.Speak (whatToSay, QueueMode.Flush, p);
        }
    }

    #region IOnInitListener implementation

    public void OnInit (OperationResult status)
    {
        if (status.Equals (OperationResult.Success))
        {
            var p = new Dictionary<string,string> ();
            _speaker.Speak (toSpeak, QueueMode.Flush, p);
        }
    }

    #endregion
}
```

When you've created your class you need to enable the `DependencyService` to discover it at run time. This is done by adding an `[assembly]` attribute above the class definition and outside of any namespace definitions.

```
using Android.Speech.Tts;
```

```
using Xamarin.Forms;
using System.Collections.Generic;
using DependencyServiceSample.Droid;


[assembly: Xamarin.Forms.Dependency (typeof (TextToSpeechAndroid))]
namespace DependencyServiceSample.Droid {
    ...
```

This attribute registers the class with the `DependencyService` so it can be used when an instance of the `ITextToSpeech` interface is needed.

Read DependencyService online: https://riptutorial.com/xamarin-forms/topic/2508/dependencyservice

Chapter 16: Display Alert

Examples

DisplayAlert

An alert box can be popped-up on a `Xamarin.Forms Page` by the method, `DisplayAlert`. We can provide a Title, Body (Text to be alerted) and one/two Action Buttons. `Page` offers two overrides of `DisplayAlert` method.

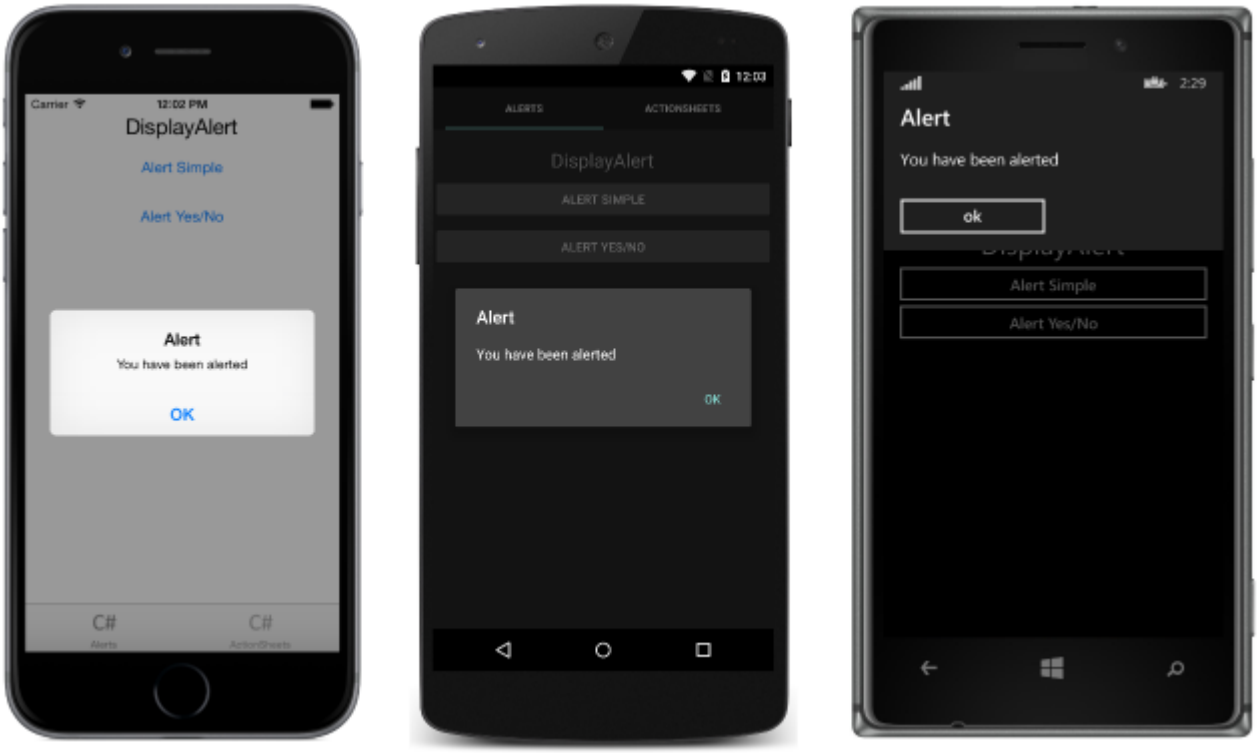
1. `public Task DisplayAlert (String title, String message, String cancel)`

This override presents an alert dialog to the application user with a single cancel button. The alert displays modally and once dismissed the user continues interacting with the application.

Example :

```
DisplayAlert ("Alert", "You have been alerted", "OK");
```

Above snippet will present a native implementation of Alerts in each platform (`AlertDialog` in Android, `UIAlertView` in iOS, `MessageDialog` in Windows) as below.



2. `public System.Threading.Tasks.Task<bool> DisplayAlert (String title, String message, String accept, String cancel)`

This override presents an alert dialog to the application user with an accept and a cancel button. It captures a user's response by presenting two buttons and returning a `boolean`. To get a response from an alert, supply text for both buttons and await the method. After the user selects one of the

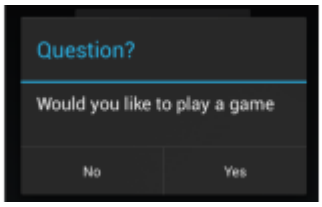
options the answer will be returned to the code.

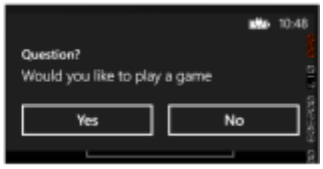
Example :

```
var answer = await DisplayAlert ("Question?", "Would you like to play a game", "Yes", "No");
Debug.WriteLine ("Answer: " + (answer?"Yes":"No"));
```

Example 2:(if Condition true or false check to alert proceed)

```
async void listSelected(object sender, SelectedItemChangedEventArgs e)
    {
        var ans = await DisplayAlert("Question?", "Would you like Delete", "Yes", "No");
        if (ans == true)
        {
            //Success condition
        }
        else
        {
            //false conditon
        }
     }
```





Alert Example with only one button and action

```
var alertResult = await DisplayAlert("Alert Title", Alert Message, null, "OK");
if(!alertResult)
{
   //do your stuff.
}
```

Here we will get Ok click action.

Read Display Alert online: https://riptutorial.com/xamarin-forms/topic/4883/display-alert

Chapter 17: Effects

Introduction

Effects simplifies platform specific customizations. When there is a need to modify a Xamarin Forms Control's properties, Effects can be used. When there is a need to override the Xamarin Forms Control's methods, Custom renderers can be used

Examples

Adding platform specific Effect for an Entry control

1. Create a new Xamarin Forms app using PCL File -> New Solution -> Multiplatform App -> Xamarin Forms -> Forms App; Name the project as `EffectsDemo`
2. Under the iOS project, add a new `Effect` class that inherits from `PlatformEffect` class and overrides the methods `OnAttached`, `OnDetached` and `OnElementPropertyChanged` Notice the two attributes `ResolutionGroupName` and `ExportEffect`, these are required for consuming this effect from the PCL/shared project.

- `OnAttached` is the method where the logic for customization goes in
- `OnDetached` is the method where the clean up and de-registering happens
- `OnElementPropertyChanged` is the method which gets triggered upon property changes of different elements. To identify the right property, check for the exact property change and add your logic. In this example, `OnFocus` will give the `Blue` color and `OutofFocus` will give `Red` Color

```
using System;
using EffectsDemo.iOS;
using UIKit;
using Xamarin.Forms;
using Xamarin.Forms.Platform.iOS;

[assembly: ResolutionGroupName("xhackers")]
[assembly: ExportEffect(typeof(FocusEffect), "FocusEffect")]
namespace EffectsDemo.iOS
{
public class FocusEffect : PlatformEffect
{
public FocusEffect()
{
}
UIColor backgroundColor;
protected override void OnAttached()
{
    try
    {
        Control.BackgroundColor = backgroundColor = UIColor.Red;
    }
```

```
     catch (Exception ex)
     {
         Console.WriteLine("Cannot set attacked property" + ex.Message);
     }
 }

 protected override void OnDetached()
 {
     throw new NotImplementedException();
 }

 protected override void
OnElementPropertyChanged(System.ComponentModel.PropertyChangedEventArgs args)
 {
     base.OnElementPropertyChanged(args);

     try
     {
         if (args.PropertyName == "IsFocused")
         {
             if (Control.BackgroundColor == backgroundColor)
             {
                 Control.BackgroundColor = UIColor.Blue;
             }
             else
             {
                 Control.BackgroundColor = backgroundColor;
             }
         }
     }
     catch (Exception ex)
     {
         Console.WriteLine("Cannot set property " + ex.Message);
     }
 }
```

}}

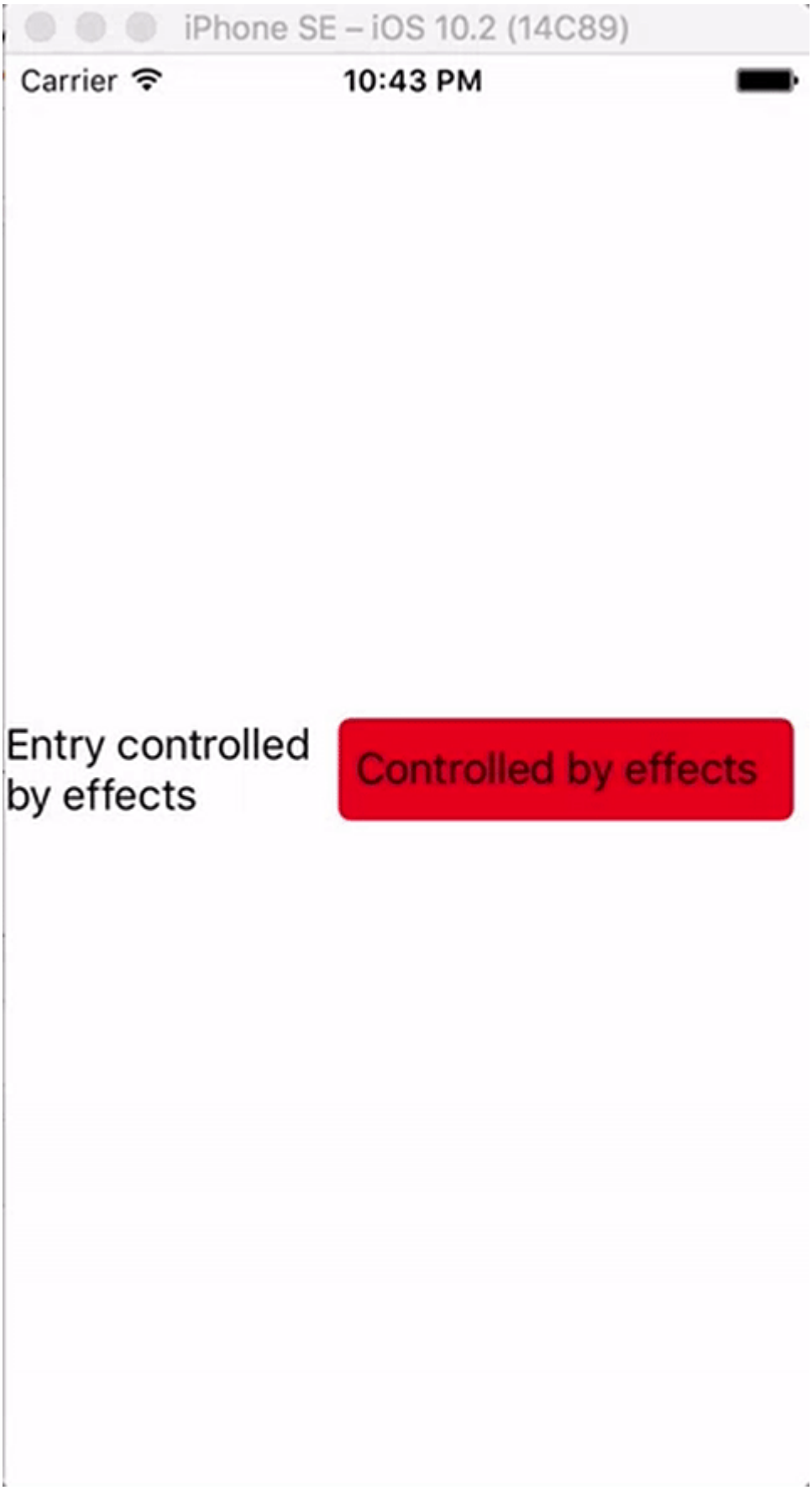
3. To Consume this effect in the application, Under the `PCL` project, create a new class named `FocusEffect` which inherits from `RoutingEffect`. This is essential to make the PCL instantiate the platform specific implementation of the effect. Sample code below:

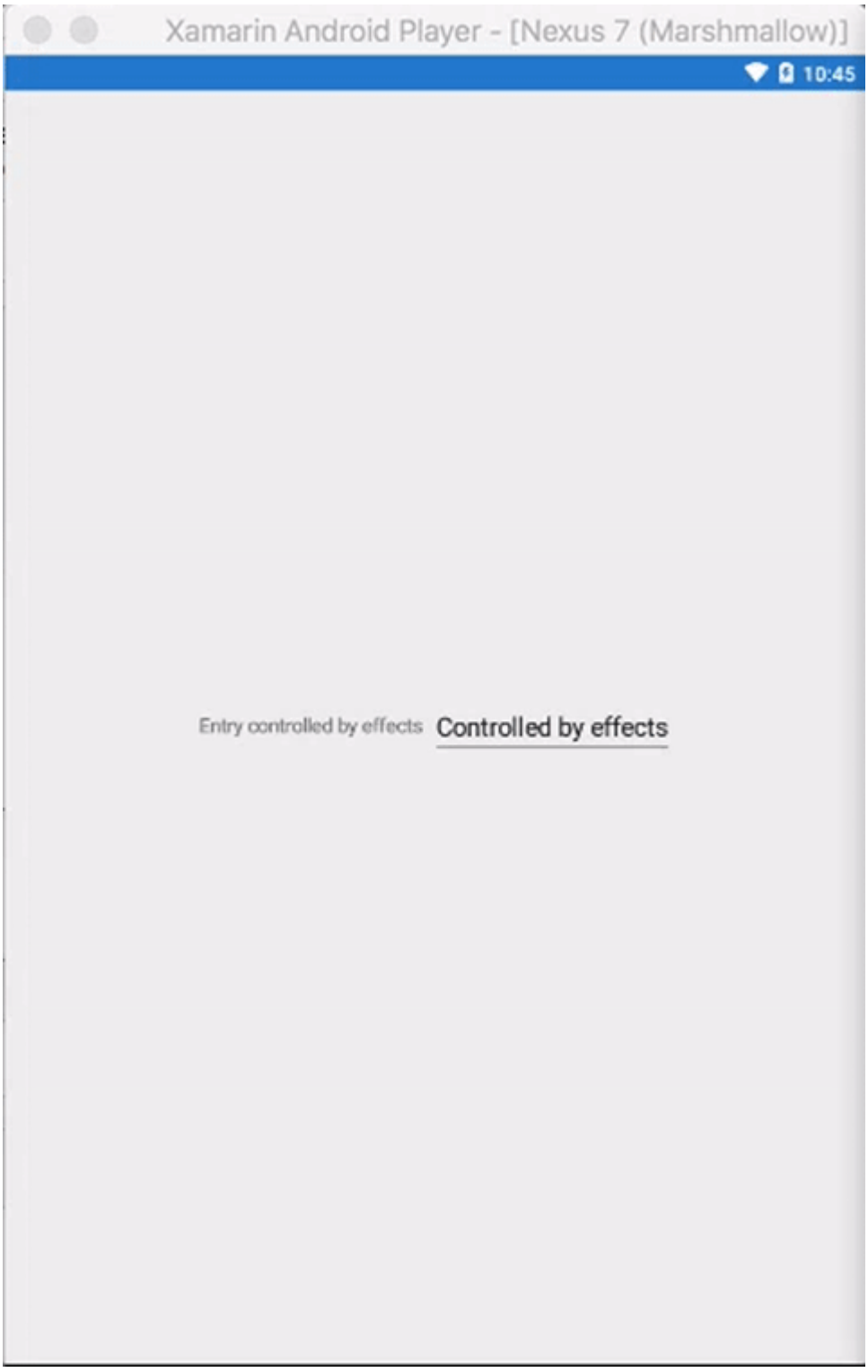
```
using Xamarin.Forms;
namespace EffectsDemo
{
    public class FocusEffect : RoutingEffect
    {
        public FocusEffect() : base("xhackers.FocusEffect")
        {
        }
    }
}
```

4. Add the effect to `Entry` control in the XAML

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
```

```
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" xmlns:local="clr-
namespace:EffectsDemo" x:Class="EffectsDemo.EffectsDemoPage">
<StackLayout Orientation="Horizontal" HorizontalOptions="Center"
VerticalOptions="Center">
<Label Text="Effects Demo" HorizontalOptions="StartAndExpand" VerticalOptions="Center"
></Label>
<Entry Text="Controlled by effects" HorizontalOptions="FillAndExpand"
VerticalOptions="Center">
    <Entry.Effects>
        <local:FocusEffect>
        </local:FocusEffect>
    </Entry.Effects>
</Entry>
</StackLayout>
</ContentPage>
```


iPhone SE – iOS 10.2 (14C89)
Carrier
10:43 PM
Entry controlled by effects
Controlled by effects



Since the Effect was implemented only in iOS version, when the app runs in `iOS Simulator` upon focusing the `Entry` background color changes and nothing happens in `Android Emulator` as the `Effect` wasn't created under `Droid` project

Read Effects online: https://riptutorial.com/xamarin-forms/topic/9252/effects

Chapter 18: Exception handling

Examples

One way to report about exceptions on iOS

Go to `Main.cs` file in **iOS project** and change existed code, like presented below:

```
    static void Main(string[] args)
    {
        try
        {
            UIApplication.Main(args, null, "AppDelegate");
        }
        catch (Exception ex)
        {
            Debug.WriteLine("iOS Main Exception: {0}", ex);

            var watson = new LittleWatson();
            watson.SaveReport(ex);
        }
    }
```

`ILittleWatson` interface, used in portable code, could look like this:

```
public interface ILittleWatson
{
    Task<bool> SendReport();

    void SaveReport(Exception ex);
}
```

Implementation for **iOS project**:

```
[assembly: Xamarin.Forms.Dependency(typeof(LittleWatson))]
namespace SomeNamespace
{
    public class LittleWatson : ILittleWatson
    {
        private const string FileName = "Report.txt";

        private readonly static string DocumentsFolder;
        private readonly static string FilePath;

        private TaskCompletionSource<bool> _sendingTask;

        static LittleWatson()
        {
            DocumentsFolder =
Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
            FilePath = Path.Combine(DocumentsFolder, FileName);
        }

        public async Task<bool> SendReport()
```

```
        {
            _sendingTask = new TaskCompletionSource<bool>();

            try
            {
                var text = File.ReadAllText(FilePath);
                File.Delete(FilePath);
                if (MFMailComposeViewController.CanSendMail)
                {
                    var email = ""; // Put receiver email here.
                    var mailController = new MFMailComposeViewController();
                    mailController.SetToRecipients(new string[] { email });
                    mailController.SetSubject("iPhone error");
                    mailController.SetMessageBody(text, false);
                    mailController.Finished += (object s, MFComposeResultEventArgs args) =>
                    {
                        args.Controller.DismissViewController(true, null);
                        _sendingTask.TrySetResult(true);
                    };

                    ShowViewController(mailController);
                }
            }
            catch (FileNotFoundException)
            {
                // No errors found.
                _sendingTask.TrySetResult(false);
            }

            return await _sendingTask.Task;
        }

        public void SaveReport(Exception ex)
        {
            var exceptionInfo = $"{ex.Message} - {ex.StackTrace}";
            File.WriteAllText(FilePath, exceptionInfo);
        }

        private static void ShowViewController(UIViewController controller)
        {
            var topController = UIApplication.SharedApplication.KeyWindow.RootViewController;
            while (topController.PresentedViewController != null)
            {
                topController = topController.PresentedViewController;
            }

            topController.PresentViewController(controller, true, null);
        }
    }
}
```

And then, somewhere, where app starts, put:

```
        var watson = DependencyService.Get<ILittleWatson>();
        if (watson != null)
        {
            await watson.SendReport();
        }
```

Read Exception handling online: https://riptutorial.com/xamarin-forms/topic/6428/exception-handling

Chapter 19: Generic Xamarin.Forms app lifecycle? Platform-dependant!

Examples

Xamarin.Forms lifecycle is not the actual app lifecycle but a cross-platform representation of it.

Lets have a look at the native app lifecycle methods for different platforms.

Android.

```
//Xamarin.Forms.Platform.Android.FormsApplicationActivity lifecycle methods:
protected override void OnCreate(Bundle savedInstanceState);
protected override void OnDestroy();
protected override void OnPause();
protected override void OnRestart();
protected override void OnResume();
protected override void OnStart();
protected override void OnStop();
```

iOS.

```
//Xamarin.Forms.Platform.iOS.FormsApplicationDelegate lifecycle methods:
public override void DidEnterBackground(UIApplication uiApplication);
public override bool FinishedLaunching(UIApplication uiApplication, NSDictionary
launchOptions);
public override void OnActivated(UIApplication uiApplication);
public override void OnResignActivation(UIApplication uiApplication);
public override void WillEnterForeground(UIApplication uiApplication);
public override bool WillFinishLaunching(UIApplication uiApplication, NSDictionary
launchOptions);
public override void WillTerminate(UIApplication uiApplication);
```

Windows.

```
//Windows.UI.Xaml.Application lifecycle methods:
public event EventHandler<System.Object> Resuming;
public event SuspendingEventHandler Suspending;
protected virtual void OnActivated(IActivatedEventArgs args);
protected virtual void OnFileActivated(FileActivatedEventArgs args);
protected virtual void OnFileOpenPickerActivated(FileOpenPickerActivatedEventArgs args);
protected virtual void OnFileSavePickerActivated(FileSavePickerActivatedEventArgs args);
protected virtual void OnLaunched(LaunchActivatedEventArgs args);
protected virtual void OnSearchActivated(SearchActivatedEventArgs args);
protected virtual void OnShareTargetActivated(ShareTargetActivatedEventArgs args);
protected virtual void OnWindowCreated(WindowCreatedEventArgs args);

//Windows.UI.Xaml.Window lifecycle methods:
public event WindowActivatedEventHandler Activated;
public event WindowClosedEventHandler Closed;
```

```
public event WindowVisibilityChangedEventHandler VisibilityChanged;
```

And now **Xamarin.Forms** app lifecycle methods:

```
//Xamarin.Forms.Application lifecycle methods:
protected virtual void OnResume();
protected virtual void OnSleep();
protected virtual void OnStart();
```

What you can easily tell from merely observing the lists, the Xamarin.Forms cross-platform app lifecycle perspective is greatly simplified. It gives you the generic clue about what state your app is in but in most production cases you will have to build some platform-dependant logic.

Read Generic Xamarin.Forms app lifecycle? Platform-dependant! online: https://riptutorial.com/xamarin-forms/topic/8329/generic-xamarin-forms-app-lifecycle--platform-dependant-

Chapter 20: Gestures

Examples

Make an Image tappable by adding a TapGestureRecognizer

There are a couple of default recognizers available in Xamarin.Forms, one of them is the `TapGestureRecognizer`.

You can add them to virtually any visual element. Have a look at a simple implementation which binds to an `Image`. Here is how to do it in code.

```
var tappedCommand = new Command(() =>
{
    //handle the tap
});

var tapGestureRecognizer = new TapGestureRecognizer { Command = tappedCommand };
image.GestureRecognizers.Add(tapGestureRecognizer);
```

Or in XAML:

```
<Image Source="tapped.jpg">
    <Image.GestureRecognizers>
        <TapGestureRecognizer
                Command="{Binding TappedCommand}"
                NumberOfTapsRequired="2" />
  </Image.GestureRecognizers>
</Image>
```

Here the command is set by using data binding. As you can see you can also set the `NumberOfTapsRequired` to enable it for more taps before it takes action. The default value is 1 tap.

Other gestures are Pinch and Pan.

Zoom an Image with the Pinch gesture

In order to make an `Image` (or any other visual element) zoomable we have to add a `PinchGestureRecognizer` to it. Here is how to do it in code:

```
var pinchGesture = new PinchGestureRecognizer();
pinchGesture.PinchUpdated += (s, e) => {
// Handle the pinch
};

image.GestureRecognizers.Add(pinchGesture);
```

But it can also be done from XAML:

```
<Image Source="waterfront.jpg">
  <Image.GestureRecognizers>
    <PinchGestureRecognizer PinchUpdated="OnPinchUpdated" />
  </Image.GestureRecognizers>
</Image>
```

In the accompanied event handler you should provide the code to zoom your image. Of course other uses can be implement as well.

```
void OnPinchUpdated (object sender, PinchGestureUpdatedEventArgs e)
{
    // ... code here
}
```

Other gestures are Tap and Pan.

Show all of the zoomed Image content with the PanGestureRecognizer

When you have a zoomed `Image` (or other content) you may want to drag around the `Image` to show all of its content in the zoomed in state.

This can be achieved by implementing the PanGestureRecognizer. From code this looks like so:

```
var panGesture = new PanGestureRecognizer();
panGesture.PanUpdated += (s, e) => {
  // Handle the pan
};

image.GestureRecognizers.Add(panGesture);
```

This can also be done from XAML:

```
<Image Source="MonoMonkey.jpg">
  <Image.GestureRecognizers>
    <PanGestureRecognizer PanUpdated="OnPanUpdated" />
  </Image.GestureRecognizers>
</Image>
```

In the code-behind event you can now handle the panning accordingly. Use this method signature to handle it:

```
void OnPanUpdated (object sender, PanUpdatedEventArgs e)
{
  // Handle the pan
}
```

Place a pin where the user touched the screen with MR.Gestures

Xamarins built in gesture recognizers provide only very basic touch handling. E.g. there is no way to get the position of a touching finger. MR.Gestures is a component which adds 14 different touch handling events. The position of the touching fingers is part of the `EventArgs` passed to all

MR.Gestures events.

If you want to place a pin anywhere on the screen, the easiest way is to use an `MR.Gestures.AbsoluteLayout` which handles the `Tapping` event.

```
<mr:AbsoluteLayout x:Name="MainLayout" Tapping="OnTapping">
    ...
</mr:AbsoluteLayout>
```

As you can see the `Tapping="OnTapping"` also feels more like .NET than Xamarins syntax with the nested `GestureRecognizers`. That syntax was copied from iOS and it smells a bit for .NET developers.

In your code behind you could add the `OnTapping` handler like this:

```
private void OnTapping(object sender, MR.Gestures.TapEventArgs e)
{
    if (e.Touches?.Length > 0)
    {
        Point touch = e.Touches[0];
        var image = new Image() { Source = "pin" };
        MainLayout.Children.Add(image, touch);
    }
}
```

Instead of the `Tapping` event, you could also use the `TappingCommand` and bind to your ViewModel, but that would complicate things in this simple example.

More samples for MR.Gestures can be found in the GestureSample app on GitHub and on the MR.Gestures website. These also show how to use all the other touch events with event handlers, commands, MVVM, ...

Read Gestures online: https://riptutorial.com/xamarin-forms/topic/3914/gestures

Chapter 21: MessagingCenter

Introduction

Xamarin.Forms has a built-in messaging mechanism to promote decoupled code. This way, view models and other components do not need to know each other. They can communicate by a simple messaging contract.

There a basically two main ingredients for using the `MessagingCenter`.

Subscribe; listen for messages with a certain signature (the contract) and execute code when a message is received. A message can have multiple subscribers.

Send; sending a message for subscribers to act upon.

Examples

Simple example

Here we will see a simple example of using the MessagingCenter in Xamarin.Forms.

First, let's have a look at subscribing to a message. In the `FooMessaging` model we subscribe to a message coming from the `MainPage`. The message should be "Hi" and when we receive it, we register a handler which sets the property `Greeting`. Lastly `this` means the current `FooMessaging` instance is registering for this message.

```
public class FooMessaging
{
    public string Greeting { get; set; }

    public FooMessaging()
    {
        MessagingCenter.Subscribe<MainPage> (this, "Hi", (sender) => {
            this.Greeting = "Hi there!";
        });
    }
}
```

To send a message triggering this functionality, we need to have a page called `MainPage`, and implement code like underneath.

```
public class MainPage : Page
{
    private void OnButtonClick(object sender, EventArgs args)
    {
        MessagingCenter.Send<MainPage> (this, "Hi");
    }
}
```

In our `MainPage` we have a button with a handler that sends a message. `this` should be an instance of `MainPage`.

Passing arguments

You can also pass arguments with a message to work with.

We will use the classed from our previous example and extend them. In the receiving part, right behind the `Subscribe` method call add the type of the argument you are expecting. Also make sure you also declare the arguments in the handler signature.

```
public class FooMessaging
{
    public string Greeting { get; set; }

    public FooMessaging()
    {
        MessagingCenter.Subscribe<MainPage, string> (this, "Hi", (sender, arg) => {
            this.Greeting = arg;
        });
    }
}
```

When sending a message, make sure to include the argument value. Also, here you add the type right behind the `Send` method and add the argument value.

```
public class MainPage : Page
{
    private void OnButtonClick(object sender, EventArgs args)
    {
        MessagingCenter.Send<MainPage, string> (this, "Hi", "Hi there!");
    }
}
```

In this example a simple string is used, but you can also use any other type of (complex) objects.

Unsubscribing

When you no longer need to receive messages, you can simply unsubscribe. You can do it like this:

```
MessagingCenter.Unsubscribe<MainPage> (this, "Hi");
```

When you are supplying arguments, you have to unsubscribe from the complete signature, like this:

```
MessagingCenter.Unsubscribe<MainPage, string> (this, "Hi");
```

Read MessagingCenter online: https://riptutorial.com/xamarin-forms/topic/9672/messagingcenter

Chapter 22: Navigation in Xamarin.Forms

Examples

NavigationPage flow

```
using System;
using Xamarin.Forms;

namespace NavigationApp
{
    public class App : Application
    {
        public App()
        {
            MainPage = new NavigationPage(new FirstPage());
        }
    }

    public class FirstPage : ContentPage
    {
        Label FirstPageLabel { get; set; } = new Label();

        Button FirstPageButton { get; set; } = new Button();

        public FirstPage()
        {
            Title = "First page";

            FirstPageLabel.Text = "This is the first page";
            FirstPageButton.Text = "Navigate to the second page";
            FirstPageButton.Clicked += OnFirstPageButtonClicked;

            var content = new StackLayout();
            content.Children.Add(FirstPageLabel);
            content.Children.Add(FirstPageButton);

            Content = content;
        }

        async void OnFirstPageButtonClicked(object sender, EventArgs e)
        {
            await Navigation.PushAsync(new SecondPage(), true);
        }
    }

    public class SecondPage : ContentPage
    {
        Label SecondPageLabel { get; set; } = new Label();

        public SecondPage()
        {
            Title = "Second page";

            SecondPageLabel.Text = "This is the second page";

            Content = SecondPageLabel;
```

```
        }
    }
}
```

NavigationPage flow with XAML

App.xaml.cs file (App.xaml file is default, so skipped)

```
using Xamrin.Forms

namespace NavigationApp
{
    public partial class App : Application
    {
        public static INavigation GlobalNavigation { get; private set; }

        public App()
        {
            InitializeComponent();
            var rootPage = new NavigationPage(new FirstPage());

            GlobalNavigation = rootPage.Navigation;

            MainPage = rootPage;
        }
    }
}
```

FirstPage.xaml file

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="NavigationApp.FirstPage"
    Title="First page">
    <ContentPage.Content>
        <StackLayout>
            <Label
                Text="This is the first page" />
            <Button
                Text="Click to navigate to a new page"
                Clicked="GoToSecondPageButtonClicked"/>
            <Button
                Text="Click to open the new page as modal"
                Clicked="OpenGlobalModalPageButtonClicked"/>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

In some cases you need to open the new page not in the current navigation but in the global one. For example, if your current page contains bottom menu, it will be visible when you push the new page in the current navigation. If you need the page to be opened over the whole visible content hiding the bottom menu and other current page's content, you need to push the new page as a modal into the global navigation. See `App.GlobalNavigation` property and the example below.

FirstPage.xaml.cs file

```
using System;
using Xamarin.Forms;

namespace NavigationApp
{
    public partial class FirstPage : ContentPage
    {
        public FirstPage()
        {
            InitializeComponent();
        }

        async void GoToSecondPageButtonClicked(object sender, EventArgs e)
        {
            await Navigation.PushAsync(new SecondPage(), true);
        }

        async void OpenGlobalModalPageButtonClicked(object sender, EventArgs e)
        {
            await App.GlobalNavigation.PushModalAsync(new SecondPage(), true);
        }
    }
}
```

SecondPage.xaml file (xaml.cs file is default, so skipped)

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="NavigationApp.SecondPage"
    Title="Second page">
    <ContentPage.Content>
        <Label
            Text="This is the second page" />
    </ContentPage.Content>
</ContentPage>
```

Hierarchical navigation with XAML

By default, the navigation pattern works like a stack of pages, calling the newest pages over the previous pages. You will need to use the NavigationPage object for this.

Pushing new pages

```
...
public class App : Application
{
    public App()
    {
        MainPage = new NavigationPage(new Page1());
    }
}
```

```
...
```

Page1.xaml

```
...
<ContentPage.Content>
    <StackLayout>
        <Label Text="Page 1" />
        <Button Text="Go to page 2" Clicked="GoToNextPage" />
    </StackLayout>
</ContentPage.Content>
...
```

Page1.xaml.cs

```
...
public partial class Page1 : ContentPage
{
    public Page1()
    {
        InitializeComponent();
    }

    protected async void GoToNextPage(object sender, EventArgs e)
    {
        await Navigation.PushAsync(new Page2());
    }
}
...
```

Page2.xaml

```
...
<ContentPage.Content>
    <StackLayout>
        <Label Text="Page 2" />
        <Button Text="Go to Page 3" Clicked="GoToNextPage" />
    </StackLayout>
</ContentPage.Content>
...
```

Page2.xaml.cs

```
...
public partial class Page2 : ContentPage
{
    public Page2()
    {
        InitializeComponent();
    }

    protected async void GoToNextPage(object sender, EventArgs e)
    {
        await Navigation.PushAsync(new Page3());
```

```
    }
}
...
```

Popping pages

Normally the user uses the back button to return pages, but sometimes you need to control this programatically, so you need to call the method **NavigationPage.PopAsync()** to return to the previous page or **NavigationPage.PopToRootAsync()** to return at the beggining, such like...

Page3.xaml

```
...
<ContentPage.Content>
    <StackLayout>
        <Label Text="Page 3" />
        <Button Text="Go to previous page" Clicked="GoToPreviousPage" />
        <Button Text="Go to beginning" Clicked="GoToStartPage" />
    </StackLayout>
</ContentPage.Content>
...
```

Page3.xaml.cs

```
...
public partial class Page3 : ContentPage
{
    public Page3()
    {
        InitializeComponent();
    }

    protected async void GoToPreviousPage(object sender, EventArgs e)
    {
        await Navigation.PopAsync();
    }

    protected async void GoToStartPage(object sender, EventArgs e)
    {
        await Navigation.PopToRootAsync();
    }
}
...
```

Modal navigation with XAML

Modal pages can created in three ways:

- From **NavigationPage** object for full screen pages
- For Alerts and Notifications
- For ActionSheets that are pop-ups menus

Full screen modals

```
...
// to open
await Navigation.PushModalAsync(new ModalPage());
// to close
await Navigation.PopModalAsync();
...
```

Alerts/Confirmations and Notifications

```
...
// alert
await DisplayAlert("Alert title", "Alert text", "Ok button text");
// confirmation
var booleanAnswer = await DisplayAlert("Confirm?", "Confirmation text", "Yes", "No");
...
```

ActionSheets

```
...
var selectedOption = await DisplayActionSheet("Options", "Cancel", "Destroy", "Option 1",
"Option 2", "Option 3");
...
```

Master Detail Root Page

```
public class App : Application
{
    internal static NavigationPage NavPage;

    public App ()
    {
        // The root page of your application
        MainPage = new RootPage();
    }
}
public class RootPage : MasterDetailPage
{
    public RootPage()
    {
        var menuPage = new MenuPage();
        menuPage.Menu.ItemSelected += (sender, e) => NavigateTo(e.SelectedItem as MenuItem);
        Master = menuPage;
        App.NavPage = new NavigationPage(new HomePage());
        Detail = App.NavPage;
    }
    protected override async void OnAppearing()
    {
        base.OnAppearing();
    }
    void NavigateTo(MenuItem menuItem)
    {
```

```
        Page displayPage = (Page)Activator.CreateInstance(menuItem.TargetType);
        Detail = new NavigationPage(displayPage);
        IsPresented = false;
    }
}
```

Master Detail Navigation

The code below shows how to perform asynchronous navigation when the app is in a MasterDetailPage context.

```
public async Task NavigateMasterDetail(Page page)
{
        if (page == null)
            {
                return;
            }

        var masterDetail = App.Current.MainPage as MasterDetailPage;

        if (masterDetail == null || masterDetail.Detail == null)
            return;

        var navigationPage = masterDetail.Detail as NavigationPage;
        if (navigationPage == null)
        {
            masterDetail.Detail = new NavigationPage(page);
            masterDetail.IsPresented = false;
            return;
        }

        await navigationPage.Navigation.PushAsync(page);


navigationPage.Navigation.RemovePage(navigationPage.Navigation.NavigationStack[navigationPage.Navigatio
- 2]);
        masterDetail.IsPresented = false;
    }
```

Read Navigation in Xamarin.Forms online: https://riptutorial.com/xamarin-forms/topic/1571/navigation-in-xamarin-forms

Chapter 23: Navigation in Xamarin.Forms

Remarks

The navigation on Xamarin.Forms is based on two principal navigation patterns: hierarchical and modal.

The hierarchical pattern allows the user to move down in a stack of pages and return pressing the "back"/"up" button.

The modal pattern is a interruption page that require a specific action from user, but normally can be canceled pressing the cancel button. Some examples are notifications, alerts, dialog boxes and register/edition pages.

Examples

Using INavigation from view model

First step is create navigation interface which we will use on view model:

```
public interface IViewNavigationService
{
    void Initialize(INavigation navigation, SuperMapper navigationMapper);
    Task NavigateToAsync(object navigationSource, object parameter = null);
    Task GoBackAsync();
}
```

In `Initialize` method I use my custom mapper where I keep collection of pages types with associated keys.

```
public class SuperMapper
{
    private readonly ConcurrentDictionary<Type, object> _typeToAssociateDictionary = new
ConcurrentDictionary<Type, object>();

    private readonly ConcurrentDictionary<object, Type> _associateToType = new
ConcurrentDictionary<object, Type>();

    public void AddMapping(Type type, object associatedSource)
    {
        _typeToAssociateDictionary.TryAdd(type, associatedSource);
        _associateToType.TryAdd(associatedSource, type);
    }

    public Type GetTypeSource(object associatedSource)
    {
        Type typeSource;
        _associateToType.TryGetValue(associatedSource, out typeSource);

        return typeSource;
    }
```

```
    public object GetAssociatedSource(Type typeSource)
    {
        object associatedSource;
        _typeToAssociateDictionary.TryGetValue(typeSource, out associatedSource);

        return associatedSource;
    }
}
```

Enum with pages:

```
public enum NavigationPageSource
{
    Page1,
    Page2
}
```

`App.cs` file:

```
public class App : Application
{
    public App()
    {
        var startPage = new Page1();
        InitializeNavigation(startPage);
        MainPage = new NavigationPage(startPage);
    }

    #region Sample of navigation initialization
    private void InitializeNavigation(Page startPage)
    {
        var mapper = new SuperMapper();
        mapper.AddMapping(typeof(Page1), NavigationPageSource.Page1);
        mapper.AddMapping(typeof(Page2), NavigationPageSource.Page2);

        var navigationService = DependencyService.Get<IViewNavigationService>();
        navigationService.Initialize(startPage.Navigation, mapper);
    }
    #endregion
}
```

In mapper I associated type of some page with enum value.

`IViewNavigationService` implementation:

```
[assembly: Dependency(typeof(ViewNavigationService))]
namespace SuperForms.Core.ViewNavigation
{
    public class ViewNavigationService : IViewNavigationService
    {
        private INavigation _navigation;
        private SuperMapper _navigationMapper;

        public void Initialize(INavigation navigation, SuperMapper navigationMapper)
        {
            _navigation = navigation;
```

```
        _navigationMapper = navigationMapper;
    }

    public async Task NavigateToAsync(object navigationSource, object parameter = null)
    {
        CheckIsInitialized();

        var type = _navigationMapper.GetTypeSource(navigationSource);

        if (type == null)
        {
            throw new InvalidOperationException(
                "Can't find associated type for " + navigationSource.ToString());
        }

        ConstructorInfo constructor;
        object[] parameters;

        if (parameter == null)
        {
            constructor = type.GetTypeInfo()
                              .DeclaredConstructors
                              .FirstOrDefault(c => !c.GetParameters().Any());

            parameters = new object[] { };
        }
        else
        {
            constructor = type.GetTypeInfo()
                              .DeclaredConstructors
                              .FirstOrDefault(c =>
                                {
                                    var p = c.GetParameters();
                                    return p.Count() == 1 &&
                                        p[0].ParameterType == parameter.GetType();
                                });

            parameters = new[] { parameter };
        }

        if (constructor == null)
        {
            throw new InvalidOperationException(
                "No suitable constructor found for page " + navigationSource.ToString());
        }

        var page = constructor.Invoke(parameters) as Page;

        await _navigation.PushAsync(page);
    }

    public async Task GoBackAsync()
    {
        CheckIsInitialized();

        await _navigation.PopAsync();
    }

    private void CheckIsInitialized()
    {
        if (_navigation == null || _navigationMapper == null)
```

```
                throw new NullReferenceException("Call Initialize method first.");
            }
        }
    }
```

I get type of page on which user want navigate and create it's instance using reflection.

And then I could use navigation service on view model:

```
var navigationService = DependencyService.Get<IViewNavigationService>();
await navigationService.NavigateToAsync(NavigationPageSource.Page2, "hello from Page1");
```

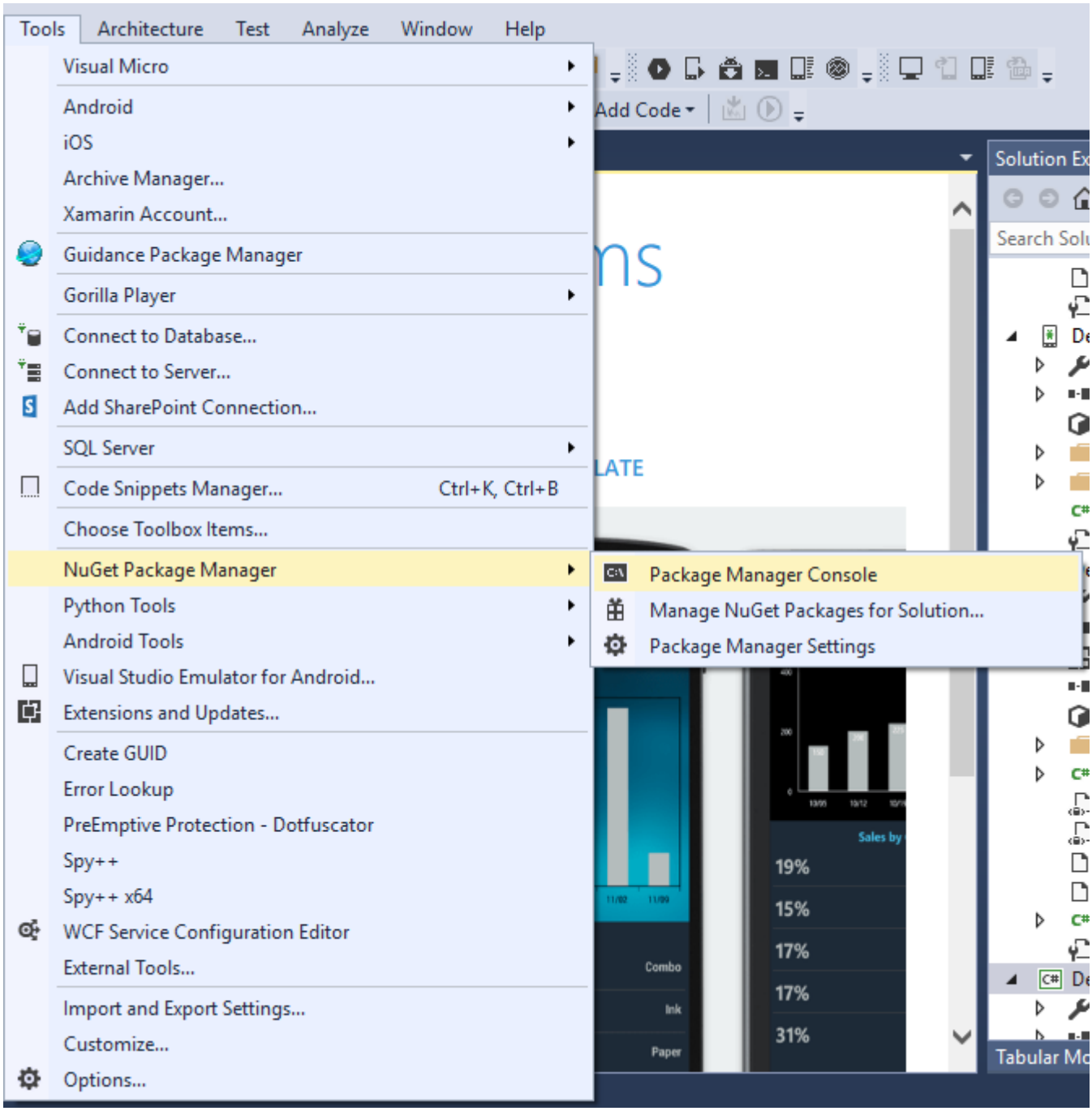
Read Navigation in Xamarin.Forms online: https://riptutorial.com/xamarin-forms/topic/2507/navigation-in-xamarin-forms

Chapter 24: OAuth2

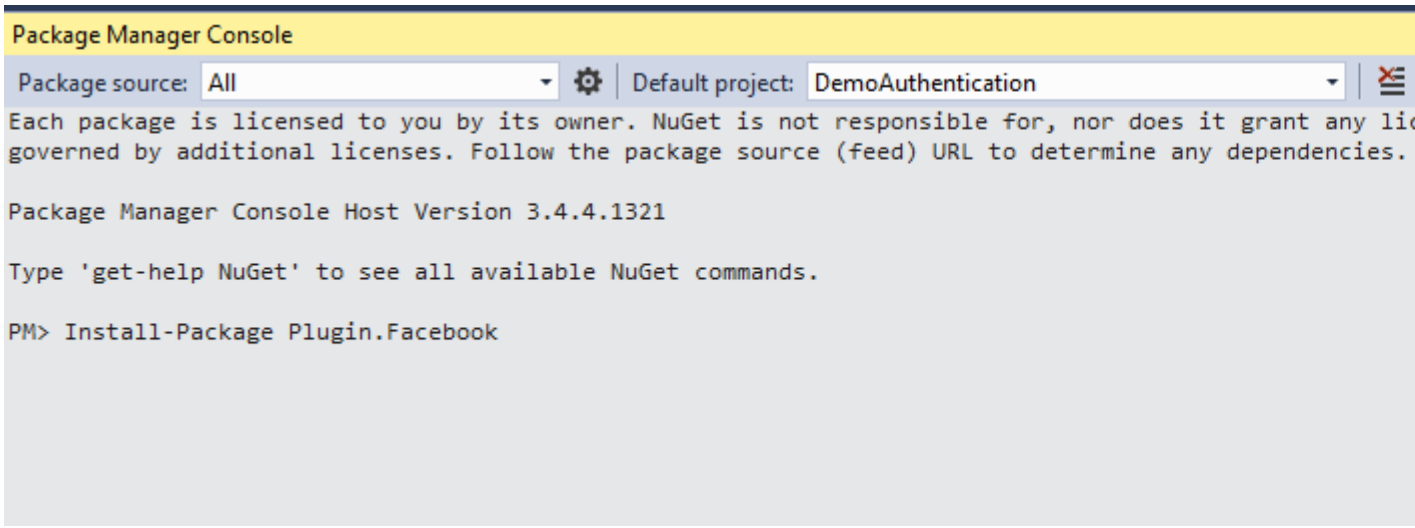
Examples

Authentication by using Plugin

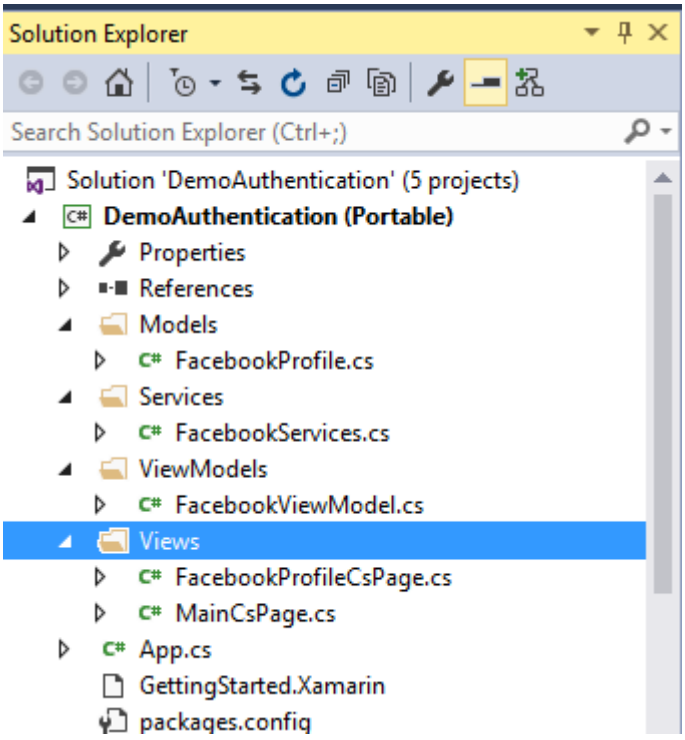
1. First, Go to **Tools** > **NuGet Package Manager** > **Package Manager Console**.



2. Enter this Command "**Install-Package Plugin.Facebook**" in Package Manger Console.



3. Now all the file is automatically created.



Video : Login with Facebook in Xamarin Forms

Other Authentication by using Plugin. Please place the command in Package Manager Console as shown in Step 2.

1. **Youtube** : Install-Package Plugin.Youtube
2. **Twitter** : Install-Package Plugin.Twitter
3. **Foursquare** : Install-Package Plugin.Foursquare
4. **Google** : Install-Package Plugin.Google
5. **Instagram** : Install-Package Plugin.Instagram
6. **Eventbrite** : Install-Package Plugin.Eventbrite

Read OAuth2 online: https://riptutorial.com/xamarin-forms/topic/8828/oauth2

Chapter 25: Platform specific visual adjustments

Examples

Idiom adjustments

Idiom specific adjustments can be done from C# code, for example for changing the layout orientation whether the view is shown or a phone or a tablet.

```
if (Device.Idiom == TargetIdiom.Phone)
{
    this.panel.Orientation = StackOrientation.Vertical;
}
else
{
    this.panel.Orientation = StackOrientation.Horizontal;
}
```

Those functionalities are also available directly from XAML code :

```
<StackLayout x:Name="panel">
  <StackLayout.Orientation>
    <OnIdiom x:TypeArguments="StackOrientation">
      <OnIdiom.Phone>Vertical</OnIdiom.Phone>
      <OnIdiom.Tablet>Horizontal</OnIdiom.Tablet>
    </OnIdiom>
  </StackLayout.Orientation>
</StackLayout>
```

Platform adjustments

Adjustments can be done for specific platforms from C# code, for example for changing padding for all the targeted platforms.

```
if (Device.OS == TargetPlatform.iOS)
{
    panel.Padding = new Thickness (10);
}
else
{
    panel.Padding = new Thickness (20);
}
```

An helper method is also available for shortened C# declarations :

```
panel.Padding = new Thickness (Device.OnPlatform(10,20,0));
```

Those functionalities are also available directly from XAML code :

```
<StackLayout x:Name="panel">
  <StackLayout.Padding>
    <OnPlatform x:TypeArguments="Thickness"
      iOS="10"
      Android="20" />
  </StackLayout.Padding>
</StackLayout>
```

Using styles

When working with XAML, using a centralized `Style` allows you to update a set of styled views from one place. All the idiom and platform adjustements can also be integrated to your styles.

```
<Style TargetType="StackLayout">
  <Setter Property="Padding">
    <Setter.Value>
      <OnPlatform x:TypeArguments="Thickness"
                  iOS="10"
                  Android="20"/>
    </Setter.Value>
  </Setter>
</Style>
```

Using custom views

You can create custom views that can be integrated to your page thanks to those adjustment tools.

Select `File > New > File... > Forms > Forms ContentView (Xaml)` and create a view for each specific layout : `TabletHome.xaml`and `PhoneHome.xaml`.

Then select `File > New > File... > Forms > Forms ContentPage` and create a `HomePage.cs` that contains :

```
using Xamarin.Forms;

public class HomePage : ContentPage
{
    public HomePage()
    {
        if (Device.Idiom == TargetIdiom.Phone)
        {
            Content = new PhoneHome();
        }
        else
        {
            Content = new TabletHome();
        }

    }
}
```

You now have a `HomePage` that creates a different view hierarchy for `Phone` and `Tablet` idioms.

Read Platform specific visual adjustments online: https://riptutorial.com/xamarin-forms/topic/5012/platform-specific-visual-adjustments

Chapter 26: Platform-specific behaviour

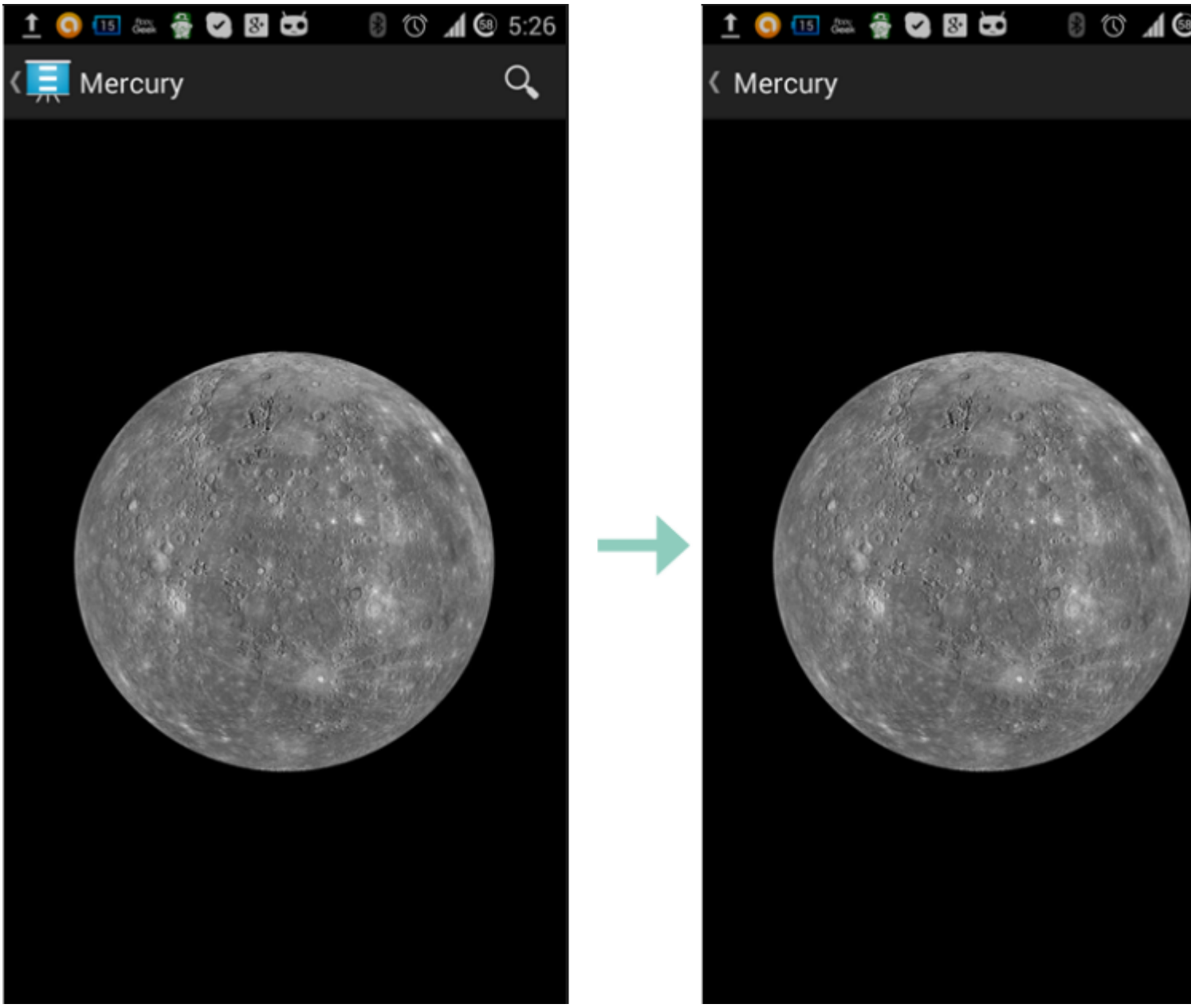
Remarks

Target Platforms

```
if(Device.OS == TargetPlatform.Android)
{

}
else if (Device.OS == TargetPlatform.iOS)
{

}
else if (Device.OS == TargetPlatform.WinPhone)
{

}
else if (Device.OS == TargetPlatform.Windows)
{

}
else if (Device.OS == TargetPlatform.Other)
{

}
```

Examples

Removing icon in navigation header in Anroid



Using a small transparent image called empty.png

```
public class MyPage : ContentPage
{
    public Page()
    {
        if (Device.OS == TargetPlatform.Android)
            NavigationPage.SetTitleIcon(this, "empty.png");
    }
}
```

Make label's font size smaller in iOS

```
Label label = new Label
{
    Text = "text"
};
if(Device.OS == TargetPlatform.iOS)
{
```

```
    label.FontSize = label.FontSize - 2;
}
```

Read Platform-specific behaviour online: https://riptutorial.com/xamarin-forms/topic/6636/platform-specific-behaviour

Chapter 27: Push Notifications

Remarks

There is no uniform way to handle push notifications in Xamarin Forms since the implementation relies heavily on platform specific features and events. Therefor platform specific code will always be necessary.

However, by using the `DependencyService` you can share as much code as possible. Also there is a plugin designed for this by rdelrosario, which can be found on his GitHub.

Code and screenshots are taken from a blog series by Gerald Versluis which explains the process in more detail.

Examples

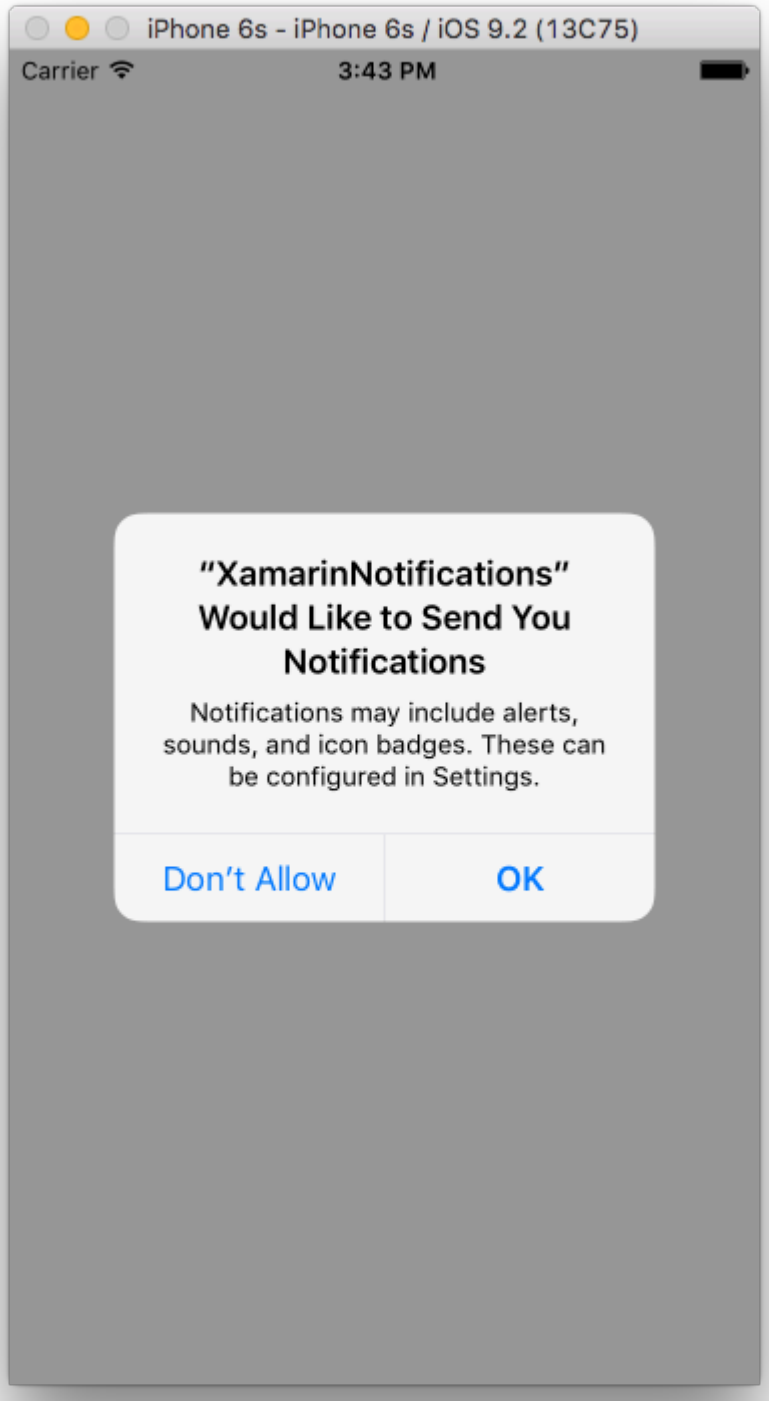
Push notifications for iOS with Azure

To start the registration for push notifications you need to execute the below code.

```
// registers for push
var settings = UIUserNotificationSettings.GetSettingsForTypes(
    UIUserNotificationType.Alert
    | UIUserNotificationType.Badge
    | UIUserNotificationType.Sound,
    new NSSet());

UIApplication.SharedApplication.RegisterUserNotificationSettings(settings);
UIApplication.SharedApplication.RegisterForRemoteNotifications();
```

This code can either be ran directly when the app starts up in the `FinishedLaunching` in the `AppDelegate.cs` file. Or you can do it whenever a user decides that they want to enable push notifications.

Running this code will trigger an alert to prompt the user if they will accept that the app can send them notifications. So also implement a scenario where the user denies that!

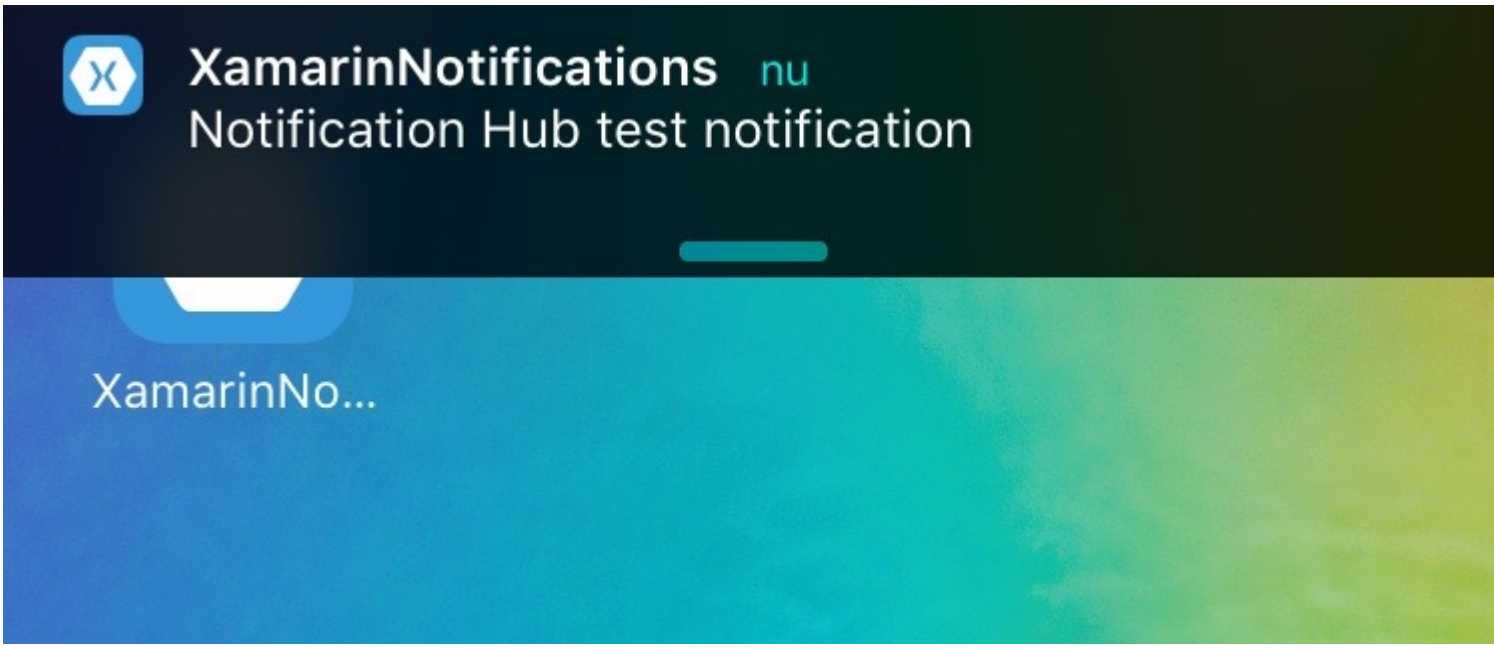


These are the events that need implementation for implementing push notifications on iOS. You can find them in the `AppDelegate.cs` file.

```
// We've successfully registered with the Apple notification service, or in our case Azure
public override void RegisteredForRemoteNotifications(UIApplication application, NSData
deviceToken)
{
   // Modify device token for compatibility Azure
    var token = deviceToken.Description;
```

```
    token = token.Trim('<', '>').Replace(" ", "");

    // You need the Settings plugin for this!
    Settings.DeviceToken = token;

    var hub = new SBNotificationHub("Endpoint=sb://xamarinnotifications-
ns.servicebus.windows.net/;SharedAccessKeyName=DefaultListenSharedAccessSignature;SharedAccessKey=<your
own key>", "xamarinnotifications");

    NSSet tags = null; // create tags if you want, not covered for now
    hub.RegisterNativeAsync(deviceToken, tags, (errorCallback) =>
    {
        if (errorCallback != null)
        {
            var alert = new UIAlertView("ERROR!", errorCallback.ToString(), null, "OK", null);
            alert.Show();
        }
    });
}

// We've received a notification, yay!
public override void ReceivedRemoteNotification(UIApplication application, NSDictionary
userInfo)
{
    NSObject inAppMessage;

    var success = userInfo.TryGetValue(new NSString("inAppMessage"), out inAppMessage);

    if (success)
    {
        var alert = new UIAlertView("Notification!", inAppMessage.ToString(), null, "OK",
null);
        alert.Show();
    }
}

// Something went wrong while registering!
public override void FailedToRegisterForRemoteNotifications(UIApplication application, NSError
error)
{
   var alert = new UIAlertView("Computer says no", "Notification registration failed! Try
again!", null, "OK", null);

   alert.Show();
}
```

When a notification is received this is what it looks like.



Push notifications for Android with Azure

Implementation on Android is a bit more work and requires a specific `Service` to be implemented.

First lets check if our device is capable of receiving push notifications, and if so, register it with Google. This can be done with this code in our `MainActivity.cs` file.

```
protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);

    global::Xamarin.Forms.Forms.Init(this, bundle);

    // Check to ensure everything's setup right for push
    GcmClient.CheckDevice(this);
    GcmClient.CheckManifest(this);
    GcmClient.Register(this, NotificationsBroadcastReceiver.SenderIDs);

    LoadApplication(new App());
}
```

The SenderIDs can be found in the code underneath and is the project number that you get from the Google developer dashboard in order to be able to send push messages.

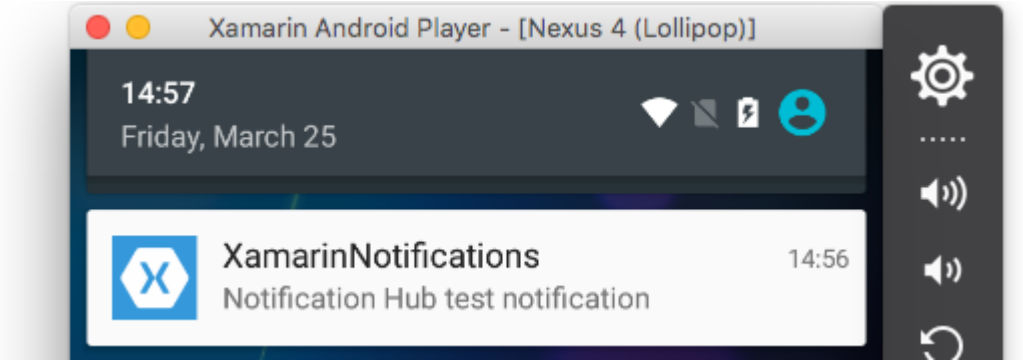
```
using Android.App;
using Android.Content;
using Gcm.Client;
using Java.Lang;
using System;
using WindowsAzure.Messaging;
using XamarinNotifications.Helpers;

// These attributes are to register the right permissions for our app concerning push messages
[assembly: Permission(Name = "com.versluisit.xamarinnotifications.permission.C2D_MESSAGE")]
[assembly: UsesPermission(Name =
"com.versluisit.xamarinnotifications.permission.C2D_MESSAGE")]
[assembly: UsesPermission(Name = "com.google.android.c2dm.permission.RECEIVE")]
```

```
//GET_ACCOUNTS is only needed for android versions 4.0.3 and below
[assembly: UsesPermission(Name = "android.permission.GET_ACCOUNTS")]
[assembly: UsesPermission(Name = "android.permission.INTERNET")]
[assembly: UsesPermission(Name = "android.permission.WAKE_LOCK")]

namespace XamarinNotifications.Droid.PlatformSpecifics
{
    // These attributes belong to the BroadcastReceiver, they register for the right intents
    [BroadcastReceiver(Permission = Constants.PERMISSION_GCM_INTENTS)]
    [IntentFilter(new[] { Constants.INTENT_FROM_GCM_MESSAGE },
    Categories = new[] { "com.versluisit.xamarinnotifications" })]
    [IntentFilter(new[] { Constants.INTENT_FROM_GCM_REGISTRATION_CALLBACK },
    Categories = new[] { "com.versluisit.xamarinnotifications" })]
    [IntentFilter(new[] { Constants.INTENT_FROM_GCM_LIBRARY_RETRY },
    Categories = new[] { "com.versluisit.xamarinnotifications" })]

    // This is the bradcast reciever
    public class NotificationsBroadcastReceiver : GcmBroadcastReceiverBase<PushHandlerService>
    {
        // TODO add your project number here
        public static string[] SenderIDs = { "96688------" };
    }

    [Service] // Don't forget this one! This tells Xamarin that this class is a Android
Service
    public class PushHandlerService : GcmServiceBase
    {
        // TODO add your own access key
        private string _connectionString =
ConnectionString.CreateUsingSharedAccessKeyWithListenAccess(
            new Java.Net.URI("sb://xamarinnotifications-ns.servicebus.windows.net/"), "<your
key here>");

        // TODO add your own hub name
        private string _hubName = "xamarinnotifications";

        public static string RegistrationID { get; private set; }

        public PushHandlerService() : base(NotificationsBroadcastReceiver.SenderIDs)
        {
        }

        // This is the entry point for when a notification is received
        protected override void OnMessage(Context context, Intent intent)
        {
            var title = "XamarinNotifications";

            if (intent.Extras.ContainsKey("title"))
                title = intent.Extras.GetString("title");

            var messageText = intent.Extras.GetString("message");

            if (!string.IsNullOrEmpty(messageText))
                CreateNotification(title, messageText);
        }

        // The method we use to compose our notification
        private void CreateNotification(string title, string desc)
        {
            // First we make sure our app will start when the notification is pressed
```

```
            const int pendingIntentId = 0;
            const int notificationId = 0;

            var startupIntent = new Intent(this, typeof(MainActivity));
            var stackBuilder = TaskStackBuilder.Create(this);

            stackBuilder.AddParentStack(Class.FromType(typeof(MainActivity)));
            stackBuilder.AddNextIntent(startupIntent);

            var pendingIntent =
                stackBuilder.GetPendingIntent(pendingIntentId, PendingIntentFlags.OneShot);

            // Here we start building our actual notification, this has some more
            // interesting customization options!
            var builder = new Notification.Builder(this)
                .SetContentIntent(pendingIntent)
                .SetContentTitle(title)
                .SetContentText(desc)
                .SetSmallIcon(Resource.Drawable.icon);

            // Build the notification
            var notification = builder.Build();
            notification.Flags = NotificationFlags.AutoCancel;

            // Get the notification manager
            var notificationManager =
                GetSystemService(NotificationService) as NotificationManager;

            // Publish the notification to the notification manager
            notificationManager.Notify(notificationId, notification);
        }

        // Whenever an error occurs in regard to push registering, this fires
        protected override void OnError(Context context, string errorId)
        {
            Console.Out.WriteLine(errorId);
        }

        // This handles the successful registration of our device to Google
        // We need to register with Azure here ourselves
        protected override void OnRegistered(Context context, string registrationId)
        {
            var hub = new NotificationHub(_hubName, _connectionString, context);

            Settings.DeviceToken = registrationId;

            // TODO set some tags here if you want and supply them to the Register method
            var tags = new string[] { };

            hub.Register(registrationId, tags);
        }

        // This handles when our device unregisters at Google
        // We need to unregister with Azure
        protected override void OnUnRegistered(Context context, string registrationId)
        {
            var hub = new NotificationHub(_hubName, _connectionString, context);

            hub.UnregisterAll(registrationId);
        }
    }
```

```
}
```

A sample notification on Android looks like this.

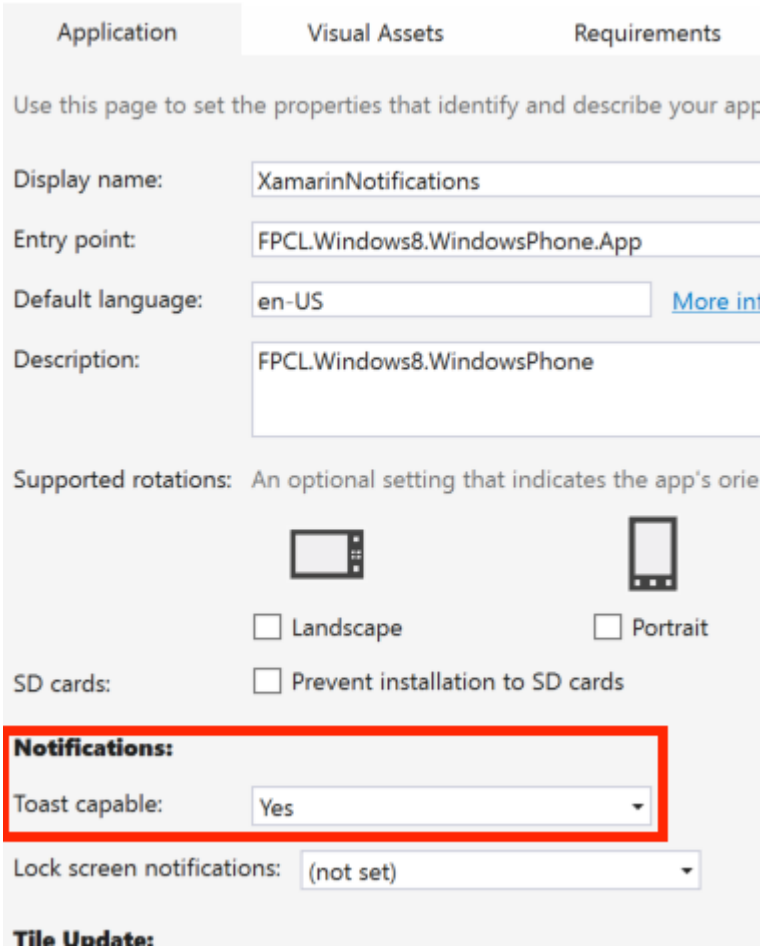


Push notifications for Windows Phone with Azure

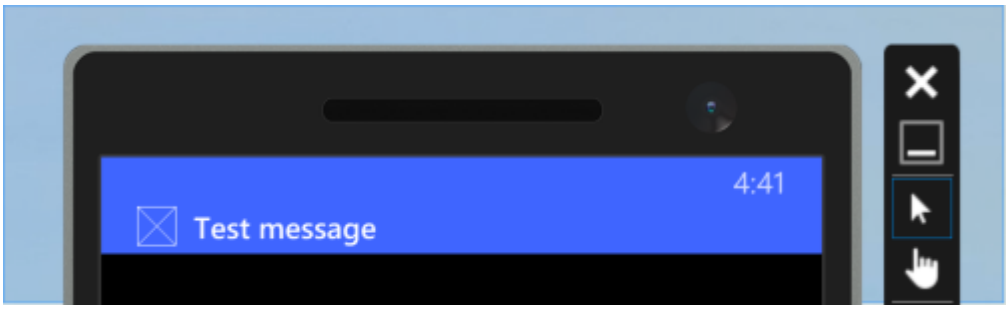
On Windows Phone something like the code underneath needs to be implemented to start working with push notifications. This can be found in the `App.xaml.cs` file.

```
protected async override void OnLaunched(LaunchActivatedEventArgs e)
{
    var channel = await
PushNotificationChannelManager.CreatePushNotificationChannelForApplicationAsync();

    // TODO add connection string here
    var hub = new NotificationHub("XamarinNotifications", "<connection string with listen
access>");
    var result = await hub.RegisterNativeAsync(channel.Uri);

    // Displays the registration ID so you know it was successful
    if (result.RegistrationId != null)
    {
        Settings.DeviceToken = result.RegistrationId;
    }

    // The rest of the default code is here
}
```

Also don't forget to enable the capabilities in the `Package.appxmanifest` file.



A sample push notification may look like this:



Read Push Notifications online: https://riptutorial.com/xamarin-forms/topic/5042/push-notifications

Chapter 28: Push Notifications

Remarks

AWS Simple Notification Service Lingo:

Endpoint - The endpoint can be a phone, email address or whatever, it's what AWS SNS can hit back with a notification

Topic - Essentially a group that contains all of your endpoints

Subscribe - You sign up your phone/client to receive notifcations

Generic Push Notification Lingo:

APNS - Apple Push Notification Service. Apple is the only one who can send push notifications. This is why we provision our app with the proper certificate. We provide AWS SNS the certificate that Apple provides us to authorize SNS to send a notification to APNS on our behalf.

GCM - Google Cloud Messaging is very similar to APNS. Google is the only one who can directly send push notifications. So we first register our App in GCM and hand over our token to AWS SNS. SNS handles all the complex stuff dealing with GCM and sending over the data.

Examples

iOS Example

1. You will need a development device
2. Go to your Apple Developer Account and create a provisioning profile with Push Notifications enabled
3. You will need some sort of way to notify your phone (AWS, Azure..etc) **We will use AWS here**

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();

   //after typical Xamarin.Forms Init Stuff

   //variable to set-up the style of notifications you want, iOS supports 3 types

   var pushSettings = UIUserNotificationSettings.GetSettingsForTypes(
              UIUserNotificationType.Alert |
              UIUserNotificationType.Badge |
              UIUserNotificationType.Sound,
              null );
```

```
        //both of these methods are in iOS, we have to override them and set them up
        //to allow push notifications

        app.RegisterUserNotificationSettings(pushSettings);  //pass the supported push
notifications settings to register app in settings page


}

public override async void RegisteredForRemoteNotifications(UIApplication application, NSData
token)
    {
        AmazonSimpleNotificationServiceClient snsClient = new
AmazonSimpleNotificationServiceClient("your AWS credentials here");

        // This contains the registered push notification token stored on the phone.
        var deviceToken = token.Description.Replace("<", "").Replace(">", "").Replace(" ",
"");

        if (!string.IsNullOrEmpty(deviceToken))
        {
            //register with SNS to create an endpoint ARN, this means AWS can message your
phone
            var response = await snsClient.CreatePlatformEndpointAsync(
            new CreatePlatformEndpointRequest
            {
                Token = deviceToken,
                PlatformApplicationArn = "yourARNwouldgohere" /* insert your platform
application ARN here */
            });

            var endpoint = response.EndpointArn;

            //AWS lets you create topics, so use subscribe your app to a topic, so you can
easily send out one push notification to all of your users
            var subscribeResponse = await snsClient.SubscribeAsync(new SubscribeRequest
            {
                TopicArn = "YourTopicARN here",
                Endpoint = endpoint,
                Protocol = "application"

            });

        }

    }
```

Read Push Notifications online: https://riptutorial.com/xamarin-forms/topic/5998/push-notifications

Chapter 29: SQL Database and API in Xamarin Forms.

Remarks

Create your own api with Microsoft SQL database and implemente them in Xamarin forms application.

Examples

Create API using SQL database and implement in Xamarin forms,

Source Code Blog

Read SQL Database and API in Xamarin Forms. online: https://riptutorial.com/xamarin-forms/topic/6513/sql-database-and-api-in-xamarin-forms-

Chapter 30: Triggers & Behaviours

Examples

Xamarin Forms Trigger Example

`Trigger`s are an easy way to add some UX responsiveness to your application. One easy way to do this is to add a `Trigger` which changes a `Label`'s `TextColor` based on whether its related `Entry` has text entered into it or not.

Using a `Trigger` for this allows the `Label.TextColor` to change from gray (when no text is entered) to black (as soon as the users enters text):

Converter (each converter is given an `Instance` variable which is used in the binding so that a new instance of the class is not created each time it is used):

```
/// <summary>
/// Used in a XAML trigger to return <c>true</c> or <c>false</c> based on the length of
<c>value</c>.
/// </summary>
public class LengthTriggerConverter : Xamarin.Forms.IValueConverter {

    /// <summary>
    /// Used so that a new instance is not created every time this converter is used in the
XAML code.
    /// </summary>
    public static LengthTriggerConverter Instance = new LengthTriggerConverter();

    /// <summary>
    /// If a `ConverterParameter` is passed in, a check to see if <c>value</c> is greater than
<c>parameter</c> is made. Otherwise, a check to see if <c>value</c> is over 0 is made.
    /// </summary>
    /// <param name="value">The length of the text from an Entry/Label/etc.</param>
    /// <param name="targetType">The Type of object/control that the text/value is coming
from.</param>
    /// <param name="parameter">Optional, specify what length to test against (example: for 3
Letter Name, we would choose 2, since the 3 Letter Name Entry needs to be over 2 characters),
if not specified, defaults to 0.</param>
    /// <param name="culture">The current culture set in the device.</param>
    /// <returns><c>object</c>, which is a <c>bool</c> (<c>true</c> if <c>value</c> is greater
than 0 (or is greater than the parameter), <c>false</c> if not).</returns>
    public object Convert(object value, System.Type targetType, object parameter, CultureInfo
culture)     { return DoWork(value, parameter); }
    public object ConvertBack(object value, System.Type targetType, object parameter,
CultureInfo culture) { return DoWork(value, parameter); }

    private static object DoWork(object value, object parameter) {
        int parameterInt = 0;

        if(parameter != null) { //If param was specified, convert and use it, otherwise, 0 is
used

            string parameterString = (string)parameter;
```

```
            if(!string.IsNullOrEmpty(parameterString)) { int.TryParse(parameterString, out
parameterInt); }
        }

        return (int)value > parameterInt;
    }
}
```

XAML (the XAML code uses the `x:Name` of the `Entry` to figure out in the `Entry.Text` property is over 3 characters long.):

```
<StackLayout>
    <Label Text="3 Letter Name">
        <Label.Triggers>
            <DataTrigger TargetType="Label"
                         Binding="{Binding Source={x:Reference NameEntry},
                                           Path=Text.Length,
                                           Converter={x:Static
helpers:LengthTriggerConverter.Instance},
                                           ConverterParameter=2}"
                         Value="False">
                <Setter Property="TextColor"
                        Value="Gray"/>
            </DataTrigger>
        </Label.Triggers>
    </Label>
    <Entry x:Name="NameEntry"
           Text="{Binding MealAmount}"
           HorizontalOptions="StartAndExpand"/>
</StackLayout>
```

Multi Triggers

MultiTrigger is not needed frequently but there are some situations where it is very handy. MultiTrigger behaves similarly to Trigger or DataTrigger but it has multiple conditions. All the conditions must be true for a Setters to fire. Here is a simple example:

```
<!-- Text field needs to be initialized in order for the trigger to work at start -->
<Entry x:Name="email" Placeholder="Email" Text="" />
<Entry x:Name="phone" Placeholder="Phone" Text="" />
<Button Text="Submit">
    <Button.Triggers>
        <MultiTrigger TargetType="Button">
            <MultiTrigger.Conditions>
                <BindingCondition Binding="{Binding Source={x:Reference email},
Path=Text.Length}" Value="0" />
                <BindingCondition Binding="{Binding Source={x:Reference phone},
Path=Text.Length}" Value="0" />
            </MultiTrigger.Conditions>
            <Setter Property="IsEnabled" Value="False" />
        </MultiTrigger>
    </Button.Triggers>
</Button>
```

The example has two different entries, phone and email, and one of them is required to be filled. The MultiTrigger disables the submit button when both fields are empty.

Read Triggers & Behaviours online: https://riptutorial.com/xamarin-forms/topic/6012/triggers---behaviours

Chapter 31: Unit Testing

Examples

Testing the view models

Before we start...

In terms of application layers your ViewModel is a class containing all the business logic and rules making the app do what it should according to the requirements. It's also important to make it as much independent as possible reducing references to UI, data layer, native features and API calls etc. All of these makes your VM be testable.
In short, your ViewModel:

- Should not depend on UI classes (views, pages, styles, events);
- Should not use static data of another classes (as much as you can);
- Should implement the business logic and prepare data to be should on UI;
- Should use other components (database, HTTP, UI-specific) via interfaces being resolved using Dependency Injection.

  Your ViewModel may have properties of another VMs types as well. For example `ContactsPageViewModel` will have propery of collection type like `ObservableCollection<ContactListItemViewModel>`

Business requirements

Let's say we have the following functionality to implement:

```
As an unauthorized user
I want to log into the app
So that I will access the authorized features
```

After clarifying the user story we defined the following scenarios:

```
Scenario: trying to log in with valid non-empty creds
  Given the user is on Login screen
   When the user enters 'user' as username
    And the user enters 'pass' as password
    And the user taps the Login button
   Then the app shows the loading indicator
    And the app makes an API call for authentication

Scenario: trying to log in empty username
  Given the user is on Login screen
   When the user enters '  ' as username
    And the user enters 'pass' as password
```

```
    And the user taps the Login button
   Then the app shows an error message saying 'Please, enter correct username and password'
    And the app doesn't make an API call for authentication
```

We will stay with only these two scenarios. Of course, there should be much more cases and you should define all of them before actual coding, but it's pretty enough for us now to get familiar with unit testing of view models.

Let's follow the classical TDD approach and start with writing an empty class being tested. Then we will write tests and will make them green by implementing the business functionality.

Common classes

```
public abstract class BaseViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

Services

Do you remember our view model must not utilize UI and HTTP classes directly? You should define them as abstractions instead and not to depend on implementation details.

```
/// <summary>
/// Provides authentication functionality.
/// </summary>
public interface IAuthenticationService
{
    /// <summary>
    /// Tries to authenticate the user with the given credentials.
    /// </summary>
    /// <param name="userName">UserName</param>
    /// <param name="password">User's password</param>
    /// <returns>true if the user has been successfully authenticated</returns>
    Task<bool> Login(string userName, string password);
}

/// <summary>
/// UI-specific service providing abilities to show alert messages.
/// </summary>
public interface IAlertService
{
    /// <summary>
    /// Show an alert message to the user.
    /// </summary>
    /// <param name="title">Alert message title</param>
    /// <param name="message">Alert message text</param>
```

```
    Task ShowAlert(string title, string message);
}
```

Building the ViewModel stub

Ok, we're gonna have the page class for Login screen, but let's start with ViewModel first:

```
public class LoginPageViewModel : BaseViewModel
{
    private readonly IAuthenticationService authenticationService;
    private readonly IAlertService alertService;

    private string userName;
    private string password;
    private bool isLoading;

    private ICommand loginCommand;

    public LoginPageViewModel(IAuthenticationService authenticationService, IAlertService
alertService)
    {
        this.authenticationService = authenticationService;
        this.alertService = alertService;
    }

    public string UserName
    {
        get
        {
            return userName;
        }
        set
        {
            if (userName!= value)
            {
                userName= value;
                OnPropertyChanged();
            }
        }
    }

    public string Password
    {
        get
        {
            return password;
        }
        set
        {
            if (password != value)
            {
                password = value;
                OnPropertyChanged();
            }
        }
    }

    public bool IsLoading
```

```
    {
        get
        {
            return isLoading;
        }
        set
        {
            if (isLoading != value)
            {
                isLoading = value;
                OnPropertyChanged();
            }
        }
    }

    public ICommand LoginCommand => loginCommand ?? (loginCommand = new Command(Login));

    private void Login()
    {
        authenticationService.Login(UserName, Password);
    }
}
```

We defined two `string` properties and a command to be bound on UI. We won't describe how to build a page class, XAML markup and bind ViewModel to it in this topic as they have nothing specific.

How to create a LoginPageViewModel instance?

I think you were probably creating the VMs just with constructor. Now as you can see our VM depends on 2 services being injected as constructor parameters so can't just do `var viewModel = new LoginPageViewModel()`. If you're not familiar with Dependency Injection it's the best moment to learn about it. Proper unit-testing is impossible without knowing and following this principle.

Tests

Now let's write some tests according to use cases listed above. First of all you need to create a new assembly (just a class library or select a special testing project if you want to use Microsoft unit testing tools). Name it something like `ProjectName.Tests` and add reference to your original PCL project.

I this example I'm going to use NUnit and Moq but you can go on with any testing libs of your choise. There will be nothing special with them.

Ok, that's the test class:

```
[TestFixture]
public class LoginPageViewModelTest
```

```
{
}
```

Writing tests

Here's the test methods for the first two scenarios. Try keeping 1 test method per 1 expected result and not to check everything in one test. That will help you to receive clearer reports about what has failed in the code.

```
[TestFixture]
public class LoginPageViewModelTest
{
    private readonly Mock<IAuthenticationService> authenticationServiceMock =
        new Mock<IAuthenticationService>();
    private readonly Mock<IAlertService> alertServiceMock =
        new Mock<IAlertService>();

    [TestCase("user", "pass")]
    public void LogInWithValidCreds_LoadingIndicatorShown(string userName, string password)
    {
        LoginPageViewModel model = CreateViewModelAndLogin(userName, password);

        Assert.IsTrue(model.IsLoading);
    }

    [TestCase("user", "pass")]
    public void LogInWithValidCreds_AuthenticationRequested(string userName, string password)
    {
        CreateViewModelAndLogin(userName, password);

        authenticationServiceMock.Verify(x => x.Login(userName, password), Times.Once);
    }

    [TestCase("", "pass")]
    [TestCase("   ", "pass")]
    [TestCase(null, "pass")]
    public void LogInWithEmptyuserName_AuthenticationNotRequested(string userName, string
password)
    {
        CreateViewModelAndLogin(userName, password);

        authenticationServiceMock.Verify(x => x.Login(It.IsAny<string>(), It.IsAny<string>()),
Times.Never);
    }

    [TestCase("", "pass", "Please, enter correct username and password")]
    [TestCase("   ", "pass", "Please, enter correct username and password")]
    [TestCase(null, "pass", "Please, enter correct username and password")]
    public void LogInWithEmptyUserName_AlertMessageShown(string userName, string password,
string message)
    {
        CreateViewModelAndLogin(userName, password);

        alertServiceMock.Verify(x => x.ShowAlert(It.IsAny<string>(), message));
    }

    private LoginPageViewModel CreateViewModelAndLogin(string userName, string password)
```

```
    {
        var model = new LoginPageViewModel(
            authenticationServiceMock.Object,
            alertServiceMock.Object);

        model.UserName = userName;
        model.Password = password;

        model.LoginCommand.Execute(null);

        return model;
    }
}
```

And here we go:

Now the goal is to write correct implementation for ViewModel's `Login` method and that's it.

Business logic implementation

```
private async void Login()
{
    if (String.IsNullOrWhiteSpace(UserName) || String.IsNullOrWhiteSpace(Password))
    {
        await alertService.ShowAlert("Warning", "Please, enter correct username and
password");
    }
    else
    {
        IsLoading = true;
        bool isAuthenticated = await authenticationService.Login(UserName, Password);
    }
}
```

And after running the tests again:

Now you can keep covering your code with new tests making it more stable and regression-safe.

Read Unit Testing online: https://riptutorial.com/xamarin-forms/topic/3529/unit-testing

Chapter 32: Using ListViews

Introduction

This documentation details how to use the different components of the Xamarin Forms ListView

Examples

Pull to Refresh in XAML and Code behind

To enable Pull to Refresh in a `ListView` in Xamarin, you first need to specify that it is `PullToRefresh` enabled and then specify the name of the command you want to invoke upon the `ListView` being pulled:

```
<ListView x:Name="itemListView" IsPullToRefreshEnabled="True" RefreshCommand="Refresh">
```

The same can be achieved in code behind:

```
itemListView.IsPullToRefreshEnabled = true;
itemListView.RefreshCommand = Refresh;
```

Then, you must specify what the `Refresh` Command does in your code behind:

```
public ICommand Refresh
{
    get
    {
        itemListView.IsRefreshing = true; //This turns on the activity
                                          //Indicator for the ListView
        //Then add your code to execute when the ListView is pulled
        itemListView.IsRefreshing = false;
    }
}
```

Read Using ListViews online: https://riptutorial.com/xamarin-forms/topic/9487/using-listviews

Chapter 33: Why Xamarin Forms and When to use Xamarin Forms

Remarks

You can refer to the official Xamarin Forms documentation to explore more:

https://www.xamarin.com/forms

Examples

Why Xamarin Forms and When to use Xamarin Forms

Xamarin is becoming more and more popular - it is hard to decide when to use Xamarin.Forms and when Xamarin.Platform (so Xamarin.iOS and Xamarin.Android).

First of all you should know for what kind of applications you can use Xamarin.Forms:

1. Prototypes - to visualize how your application will look on the different devices.
2. Applications which not require platform specific functionality (like APIs) - but here please note that Xamarin is working busily to provide as many cross-platform compatibility as possible.
3. Applications where code sharing is crucial - more important than UI.
4. Applications where data displayed is more important than advanced functionality

There are also many other factors:

1. Who will be responsible for application development - if your team consists of experienced mobile developers they will be able to handle Xamarin.Forms easily. But if you have one developer per platform (native development) Forms can be bigger challenge.
2. Please also note that with Xamarin.Forms you can still encounter some issues sometimes - Xamarin.Forms platform is still being improved.
3. Fast development is sometimes very important - to reduce costs and time you can decide to use Forms.
4. When developing enterprise applications without any advanced functionality it is better to use Xamarin.Forms - it enables you to share mode code not event in mobile area but in general. Some portions of code can be shared across many platforms.

You should not use Xamarin.Forms when:

1. You have to create custom functionality and and access platform specific APIs
2. You have to create custom UI for the mobile application
3. When some functionality is not ready for Xamarin.Forms (like some specific behaviour on the mobile device)
4. Your team consists of platform specific mobile developers (mobile development in Java and/or Swift/Objective C)

Read Why Xamarin Forms and When to use Xamarin Forms online: https://riptutorial.com/xamarin-forms/topic/6869/why-xamarin-forms-and-when-to-use-xamarin-forms

Chapter 34: Working with local databases

Examples

Using SQLite.NET in a Shared Project

SQLite.NET is an open source library which makes it possible to add local-databases support using `SQLite` version 3 in a `Xamarin.Forms` project.

The steps below demonstrate how to include this component in a `Xamarin.Forms` Shared Project:

1. Download the latest version of the SQLite.cs class and add it to the Shared Project.
2. Every table that will be included in the database needs to be modeled as a class in the Shared Project. A table is defined by adding at least two attributes in the class: `Table` (for the class) and `PrimaryKey` (for a property).

For this example, a new class named `Song` is added to the Shared Project, defined as follows:

```
using System;
using SQLite;

namespace SongsApp
{
    [Table("Song")]
    public class Song
    {
        [PrimaryKey]
        public string ID { get; set; }
        public string SongName { get; set; }
        public string SingerName { get; set; }
    }
}
```

3. Next, add a new class called `Database`, which inherits from the `SQLiteConnection` class (included in SQLite.cs). In this new class, the code for database access, tables creation and CRUD operations for each table is defined. Sample code is shown below:

```
using System;
using System.Linq;
using System.Collections.Generic;
using SQLite;

namespace SongsApp
{
    public class BaseDatos : SQLiteConnection
    {
        public BaseDatos(string path) : base(path)
        {
            Initialize();
        }
```

```
        void Initialize()
        {
            CreateTable<Song>();
        }

        public List<Song> GetSongs()
        {
            return Table<Song>().ToList();
        }

        public Song GetSong(string id)
        {
            return Table<Song>().Where(t => t.ID == id).First();
        }

        public bool AddSong(Song song)
        {
            Insert(song);
        }

        public bool UpdateSong(Song song)
        {
            Update(song);
        }

        public void DeleteSong(Song song)
        {
            Delete(song);
        }
    }
}
```

4. As you could see in the previous step, the constructor of our `Database` class includes a `path` parameter, which represents the location of the file that stores the SQLite database file. A static `Database` object can be declared in `App.cs`. The `path` is platform-specific:

```
public class App : Application
{
    public static Database DB;

    public App ()
    {
        string dbFile = "SongsDB.db3";

#if __ANDROID__
        string docPath = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
        var dbPath = System.IO.Path.Combine(docPath, dbFile);
#else
#if __IOS__
        string docPath = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
        string libPath = System.IO.Path.Combine(docPath, "..", "Library");
        var dbPath = System.IO.Path.Combine(libPath, dbFile);
#else
        var dbPath = System.IO.Path.Combine(ApplicationData.Current.LocalFolder.Path, dbFile);
#endif
#endif

        DB = new Database(dbPath);
```

```
        // The root page of your application
        MainPage = new SongsPage();
    }
}
```

5. Now simply call the `DB` object through the `App` class anytime you need to perform a CRUD operation to the `Songs` table. For example, to insert a new `Song` after the user has clicked a button, you can use the following code:

```
void AddNewSongButton_Click(object sender, EventArgs a)
{
    Song s = new Song();
    s.ID = Guid.NewGuid().ToString();
    s.SongName = songNameEntry.Text;
    s.SingerName = singerNameEntry.Text;

    App.DB.AddSong(song);
}
```

Working with local databases using xamarin.forms in visual studio 2015

SQlite example Step by step Explanation

1. The steps below demonstrate how to include this component in a Xamarin.Forms Shared Project: to add packages in (pcl,Andriod,Windows,Ios) Add References Click on **Manage Nuget packages** ->click on Browse to install **SQLite.Net.Core-PCL** , **SQLite Net Extensions** after installation is completed check it once in references then

2. To add Class **Employee.cs** below code

```
using SQLite.Net.Attributes;

    namespace DatabaseEmployeeCreation.SqlLite
    {
        public   class Employee
        {
            [PrimaryKey,AutoIncrement]
            public int Eid { get; set; }
            public string Ename { get; set; }
            public string Address { get; set; }
            public string phonenumber { get; set; }
            public string email { get; set; }
        }
    }
```

3. To add one interface ISQLite

```
using SQLite.Net;
           //using SQLite.Net;
           namespace DatabaseEmployeeCreation.SqlLite.ViewModel
           {
               public interface ISQLite
               {
                   SQLiteConnection GetConnection();
```

```
            }
        }
```

4. Create a one class for database logics and methods below code is follow .

using SQLite.Net; using System.Collections.Generic; using System.Linq; using Xamarin.Forms; namespace DatabaseEmployeeCreation.SqlLite.ViewModel { public class DatabaseLogic { static object locker = new object(); SQLiteConnection database;

```
    public DatabaseLogic()
    {
        database = DependencyService.Get<ISQLite>().GetConnection();
        // create the tables
        database.CreateTable<Employee>();
    }

    public IEnumerable<Employee> GetItems()
    {
        lock (locker)
        {
            return (from i in database.Table<Employee>() select i).ToList();
        }
    }

    public IEnumerable<Employee> GetItemsNotDone()
    {
        lock (locker)
        {
            return database.Query<Employee>("SELECT * FROM [Employee]");
        }
    }

    public Employee GetItem(int id)
    {
        lock (locker)
        {
            return database.Table<Employee>().FirstOrDefault(x => x.Eid == id);
        }
    }

    public int SaveItem(Employee item)
    {
        lock (locker)
        {
            if (item.Eid != 0)
            {
                database.Update(item);
                return item.Eid;
            }
            else
            {
                return database.Insert(item);
            }
        }
    }

    public int DeleteItem(int Eid)
    {
        lock (locker)
```

```
        {
            return database.Delete<Employee>(Eid);
        }
    }
}
```

}

5. to Create a xaml.forms EmployeeRegistration.xaml

```
    <?xml version="1.0" encoding="utf-8" ?>
    <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
                 xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                 x:Class="DatabaseEmployeeCreation.SqlLite.EmployeeRegistration"
      Title="{Binding Name}" >
      <StackLayout VerticalOptions="StartAndExpand" Padding="20">

        <Label Text="Ename" />
        <Entry x:Name="nameEntry" Text="{Binding Ename}"/>
        <Label Text="Address" />
        <Editor x:Name="AddressEntry" Text="{Binding Address}"/>
        <Label Text="phonenumber" />
        <Entry x:Name="phonenumberEntry" Text="{Binding phonenumber}"/>
        <Label Text="email" />
        <Entry x:Name="emailEntry" Text="{Binding email}"/>

        <Button Text="Add" Clicked="addClicked"/>

       <!-- <Button Text="Delete" Clicked="deleteClicked"/>-->

        <Button Text="Details" Clicked="DetailsClicked"/>

        <!--  <Button Text="Edit" Clicked="speakClicked"/>-->

      </StackLayout>
    </ContentPage>
```

EmployeeRegistration.cs

```
    using DatabaseEmployeeCreation.SqlLite.ViewModel;
    using DatabaseEmployeeCreation.SqlLite.Views;
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using System.Threading.Tasks;

    using Xamarin.Forms;

    namespace DatabaseEmployeeCreation.SqlLite
    {
        public partial class EmployeeRegistration : ContentPage
        {
            private int empid;
            private Employee obj;

            public EmployeeRegistration()
            {
```

```
        InitializeComponent();

    }

    public EmployeeRegistration(Employee obj)
    {
        this.obj = obj;
        var eid = obj.Eid;
        Navigation.PushModalAsync(new EmployeeRegistration());
        var Address = obj.Address;
        var email = obj.email;
        var Ename = obj.Ename;
        var phonenumber = obj.phonenumber;
        AddressEntry. = Address;
        emailEntry.Text = email;
        nameEntry.Text = Ename;

        //AddressEntry.Text = obj.Address;
        //emailEntry.Text = obj.email;
        //nameEntry.Text = obj.Ename;
        //phonenumberEntry.Text = obj.phonenumber;

        Employee empupdate = new Employee(); //updateing Values
        empupdate.Address = AddressEntry.Text;
        empupdate.Ename = nameEntry.Text;
        empupdate.email = emailEntry.Text;
        empupdate.Eid = obj.Eid;
        App.Database.SaveItem(empupdate);
        Navigation.PushModalAsync(new EmployeeRegistration());

    }

    public EmployeeRegistration(int empid)
    {
        this.empid = empid;
        Employee lst = App.Database.GetItem(empid);
        //var Address = lst.Address;
        //var email = lst.email;
        //var Ename = lst.Ename;
        //var phonenumber = lst.phonenumber;
        //AddressEntry.Text = Address;
        //emailEntry.Text = email;
        //nameEntry.Text = Ename;
        //phonenumberEntry.Text = phonenumber;

        // to retriva values based on id to
        AddressEntry.Text = lst.Address;
        emailEntry.Text = lst.email;
        nameEntry.Text = lst.Ename;
        phonenumberEntry.Text = lst.phonenumber;

        Employee empupdate = new Employee(); //updateing Values
        empupdate.Address = AddressEntry.Text;
        empupdate.email = emailEntry.Text;
        App.Database.SaveItem(empupdate);
        Navigation.PushModalAsync(new EmployeeRegistration());
    }

    void addClicked(object sender, EventArgs e)
    {
        //var createEmp = (Employee)BindingContext;
```

```
                Employee emp = new Employee();
                emp.Address = AddressEntry.Text;
                emp.email = emailEntry.Text;
                emp.Ename = nameEntry.Text;
                emp.phonenumber = phonenumberEntry.Text;
                App.Database.SaveItem(emp);
                this.Navigation.PushAsync(new EmployeeDetails());

            }
            //void deleteClicked(object sender, EventArgs e)
            //{
            //    var emp = (Employee)BindingContext;
            //    App.Database.DeleteItem(emp.Eid);
            //    this.Navigation.PopAsync();
            //}
            void DetailsClicked(object sender, EventArgs e)
            {
                var empcancel = (Employee)BindingContext;
                this.Navigation.PushAsync(new EmployeeDetails());
            }
            //    void speakClicked(object sender, EventArgs e)
            //    {
            //        var empspek = (Employee)BindingContext;
            //        //DependencyService.Get<ITextSpeak>().Speak(empspek.Address + " " +
empspek.Ename);
            //    }
        }
    }
```

6. to display EmployeeDetails below code behind

```
using DatabaseEmployeeCreation;
using DatabaseEmployeeCreation.SqlLite;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;

namespace DatabaseEmployeeCreation.SqlLite.Views
{
    public partial class EmployeeDetails : ContentPage
    {
        ListView lv = new ListView();
        IEnumerable<Employee> lst;
        public EmployeeDetails()
        {
            InitializeComponent();
            displayemployee();
        }

        private void displayemployee()
        {
            Button btn = new Button()
            {

                Text = "Details",
                BackgroundColor = Color.Blue,
```

```
            };
            btn.Clicked += Btn_Clicked;
            //IEnumerable<Employee> lst = App.Database.GetItems();
            //IEnumerable<Employee> lst1 = App.Database.GetItemsNotDone();
            //IEnumerable<Employee> lst2 = App.Database.GetItemsNotDone();
            Content = new StackLayout()
            {
                Children = { btn },
            };
        }

        private void Btn_Clicked(object sender, EventArgs e)
        {
            lst = App.Database.GetItems();

            lv.ItemsSource = lst;
            lv.HasUnevenRows = true;
            lv.ItemTemplate = new DataTemplate(typeof(OptionsViewCell));

            Content = new StackLayout()
            {
                Children = { lv },
            };

        }
    }
```

```
public class OptionsViewCell : ViewCell
{

    int empid;
    Button btnEdit;
    public OptionsViewCell()
    {
    }
    protected override void OnBindingContextChanged()
    {
        base.OnBindingContextChanged();

        if (this.BindingContext == null)
            return;

        dynamic obj = BindingContext;
        empid = Convert.ToInt32(obj.Eid);
        var lblname = new Label
        {
            BackgroundColor = Color.Lime,
            Text = obj.Ename,
        };

        var lblAddress = new Label
        {
            BackgroundColor = Color.Yellow,
            Text = obj.Address,
        };

        var lblphonenumber = new Label
        {
            BackgroundColor = Color.Pink,
            Text = obj.phonenumber,
```

```
        };

        var lblemail = new Label
        {
            BackgroundColor = Color.Purple,
            Text = obj.email,
        };

        var lbleid = new Label
        {
            BackgroundColor = Color.Silver,
            Text = (empid).ToString(),
        };

        //var lblname = new Label
        //{
        //    BackgroundColor = Color.Lime,
        //    // HorizontalOptions = LayoutOptions.Start
        //};
        //lblname.SetBinding(Label.TextProperty, "Ename");

        //var lblAddress = new Label
        //{
        //    BackgroundColor = Color.Yellow,
        //    //HorizontalOptions = LayoutOptions.Center,
        //};
        //lblAddress.SetBinding(Label.TextProperty, "Address");

        //var lblphonenumber = new Label
        //{
        //    BackgroundColor = Color.Pink,
        //    //HorizontalOptions = LayoutOptions.CenterAndExpand,
        //};
        //lblphonenumber.SetBinding(Label.TextProperty, "phonenumber");

        //var lblemail = new Label
        //{
        //    BackgroundColor = Color.Purple,
        //    // HorizontalOptions = LayoutOptions.CenterAndExpand
        //};
        //lblemail.SetBinding(Label.TextProperty, "email");
        //var lbleid = new Label
        //{
        //    BackgroundColor = Color.Silver,
        //    // HorizontalOptions = LayoutOptions.CenterAndExpand
        //};
        //lbleid.SetBinding(Label.TextProperty, "Eid");
        Button btnDelete = new Button
        {
            BackgroundColor = Color.Gray,

            Text = "Delete",
            //WidthRequest = 15,
            //HeightRequest = 20,
            TextColor = Color.Red,
            HorizontalOptions = LayoutOptions.EndAndExpand,
        };
        btnDelete.Clicked += BtnDelete_Clicked;
        //btnDelete.PropertyChanged += BtnDelete_PropertyChanged;

        btnEdit = new Button
```

```
                {
                    BackgroundColor = Color.Gray,
                    Text = "Edit",
                    TextColor = Color.Green,
                };
                // lbleid.SetBinding(Label.TextProperty, "Eid");
                btnEdit.Clicked += BtnEdit_Clicked1; ;
                //btnEdit.Clicked += async (s, e) =>{
                //    await App.Current.MainPage.Navigation.PushModalAsync(new
EmployeeRegistration());
                //};

                View = new StackLayout()
                {
                    Orientation = StackOrientation.Horizontal,
                    BackgroundColor = Color.White,
                    Children = { lbleid, lblname, lblAddress, lblemail, lblphonenumber,
btnDelete, btnEdit },
                };

                //View = new StackLayout()
                //{ HorizontalOptions = LayoutOptions.Center, WidthRequest = 10,
BackgroundColor = Color.Yellow, Children = { lblAddress } };

                //View = new StackLayout()
                //{ HorizontalOptions = LayoutOptions.End, WidthRequest = 30, BackgroundColor
= Color.Yellow, Children = { lblemail } };

                //View = new StackLayout()
                //{ HorizontalOptions = LayoutOptions.End, BackgroundColor = Color.Green,
Children = { lblphonenumber } };




                //string Empid =c.eid ;

            }

            private async void BtnEdit_Clicked1(object sender, EventArgs e)
            {
               Employee obj= App.Database.GetItem(empid);
                if (empid > 0)
                {
                    await App.Current.MainPage.Navigation.PushModalAsync(new
EmployeeRegistration(obj));
                }
                else {
                await App.Current.MainPage.Navigation.PushModalAsync(new
EmployeeRegistration(empid));
                }
            }



            private void BtnDelete_Clicked(object sender, EventArgs e)
            {
                // var eid = Convert.ToInt32(empid);
                // var item = (Xamarin.Forms.Button)sender;
                int eid = empid;
                App.Database.DeleteItem(eid);
```

```
            }
            //private void BtnDelete_PropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
            //{
            // var ename=  e.PropertyName;
            //}
        }

        //private void BtnDelete_Clicked(object sender, EventArgs e)
        //{
        //    var eid = 8;
        //    var item = (Xamarin.Forms.Button)sender;

        //    App.Database.DeleteItem(eid);
        //}
    }
```

7. To implement method in Android and ios GetConnection() method

```
    using System;
    using Xamarin.Forms;
    using System.IO;
    using DatabaseEmployeeCreation.Droid;
    using DatabaseEmployeeCreation.SqlLite.ViewModel;
    using SQLite;
    using SQLite.Net;

    [assembly: Dependency(typeof(SQLiteEmployee_Andriod))]
    namespace DatabaseEmployeeCreation.Droid
    {
        public class SQLiteEmployee_Andriod : ISQLite
        {
            public SQLiteEmployee_Andriod()
            {
            }

            #region ISQLite implementation
            public SQLiteConnection GetConnection()
            {
                //var sqliteFilename = "EmployeeSQLite.db3";
                //string documentsPath =
System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal); // Documents
folder
                //var path = Path.Combine(documentsPath, sqliteFilename);

                //// This is where we copy in the prepopulated database
                //Console.WriteLine(path);
                //if (!File.Exists(path))
                //{
                //    var s =
Forms.Context.Resources.OpenRawResource(Resource.Raw.EmployeeSQLite);  // RESOURCE NAME ###

                //    // create a write stream
                //    FileStream writeStream = new FileStream(path, FileMode.OpenOrCreate,
FileAccess.Write);
                //    // write to the stream
                //    ReadWriteStream(s, writeStream);
                //}

                //var conn = new SQLiteConnection(path);
```

```
                //// Return the database connection
                //return conn;
                var filename = "DatabaseEmployeeCreationSQLite.db3";
                var documentspath =
Environment.GetFolderPath(Environment.SpecialFolder.Personal);
                var path = Path.Combine(documentspath, filename);
                var platform = new SQLite.Net.Platform.XamarinAndroid.SQLitePlatformAndroid();
                var connection = new SQLite.Net.SQLiteConnection(platform, path);
                return connection;
            }

            //public  SQLiteConnection GetConnection()
            //{
            //    var filename = "EmployeeSQLite.db3";
            //    var documentspath =
Environment.GetFolderPath(Environment.SpecialFolder.Personal);
            //    var path = Path.Combine(documentspath, filename);

            //    var platform = new
SQLite.Net.Platform.XamarinAndroid.SQLitePlatformAndroid();
            //    var connection = new SQLite.Net.SQLiteConnection(platform, path);
            //    return connection;
            //}
            #endregion

            /// <summary>
            /// helper method to get the database out of /raw/ and into the user filesystem
            /// </summary>
            void ReadWriteStream(Stream readStream, Stream writeStream)
            {
                int Length = 256;
                Byte[] buffer = new Byte[Length];
                int bytesRead = readStream.Read(buffer, 0, Length);
                // write the required bytes
                while (bytesRead > 0)
                {
                    writeStream.Write(buffer, 0, bytesRead);
                    bytesRead = readStream.Read(buffer, 0, Length);
                }
                readStream.Close();
                writeStream.Close();
            }
        }
    }
```

I hope this above example is very easy way i explained

Read Working with local databases online: https://riptutorial.com/xamarin-forms/topic/5997/working-with-local-databases

Chapter 35: Working with Maps

Remarks

If you're going to run your project on another computer, you'll need to generate a new API key for it, because SHA-1 fingerprints will not match for different build machines.

You can explore the project, described in example *Adding a map in Xamarin.Forms* here

Examples

Adding a map in Xamarin.Forms (Xamarin Studio)

You can simply use the native map APIs on each platform with Xamarin Forms. All you need is to download the *Xamarin.Forms.Maps* package from nuget and install it to each project (including the PCL project).

Maps Initialization

First of all you have to add this code to your platform-specific projects. For doing this you have to add the `Xamarin.FormsMaps.Init` method call, like in the examples below.

---

iOS project

File AppDelegate.cs

```
[Register("AppDelegate")]
public partial class AppDelegate : Xamarin.Forms.Platform.iOS.FormsApplicationDelegate
{
    public override bool FinishedLaunching(UIApplication app, NSDictionary options)
    {
        Xamarin.Forms.Forms.Init();
        Xamarin.FormsMaps.Init();

        LoadApplication(new App());

        return base.FinishedLaunching(app, options);
    }
}
```

Android project

File MainActivity.cs

```
[Activity(Label = "MapExample.Droid", Icon = "@drawable/icon", Theme = "@style/MyTheme",
MainLauncher = true, ConfigurationChanges = ConfigChanges.ScreenSize |
ConfigChanges.Orientation)]
public class MainActivity : Xamarin.Forms.Platform.Android.FormsAppCompatActivity
{
    protected override void OnCreate(Bundle bundle)
    {
        TabLayoutResource = Resource.Layout.Tabbar;
        ToolbarResource = Resource.Layout.Toolbar;

        base.OnCreate(bundle);

        Xamarin.Forms.Forms.Init(this, bundle);
        Xamarin.FormsMaps.Init(this, bundle);

        LoadApplication(new App());
    }
}
```

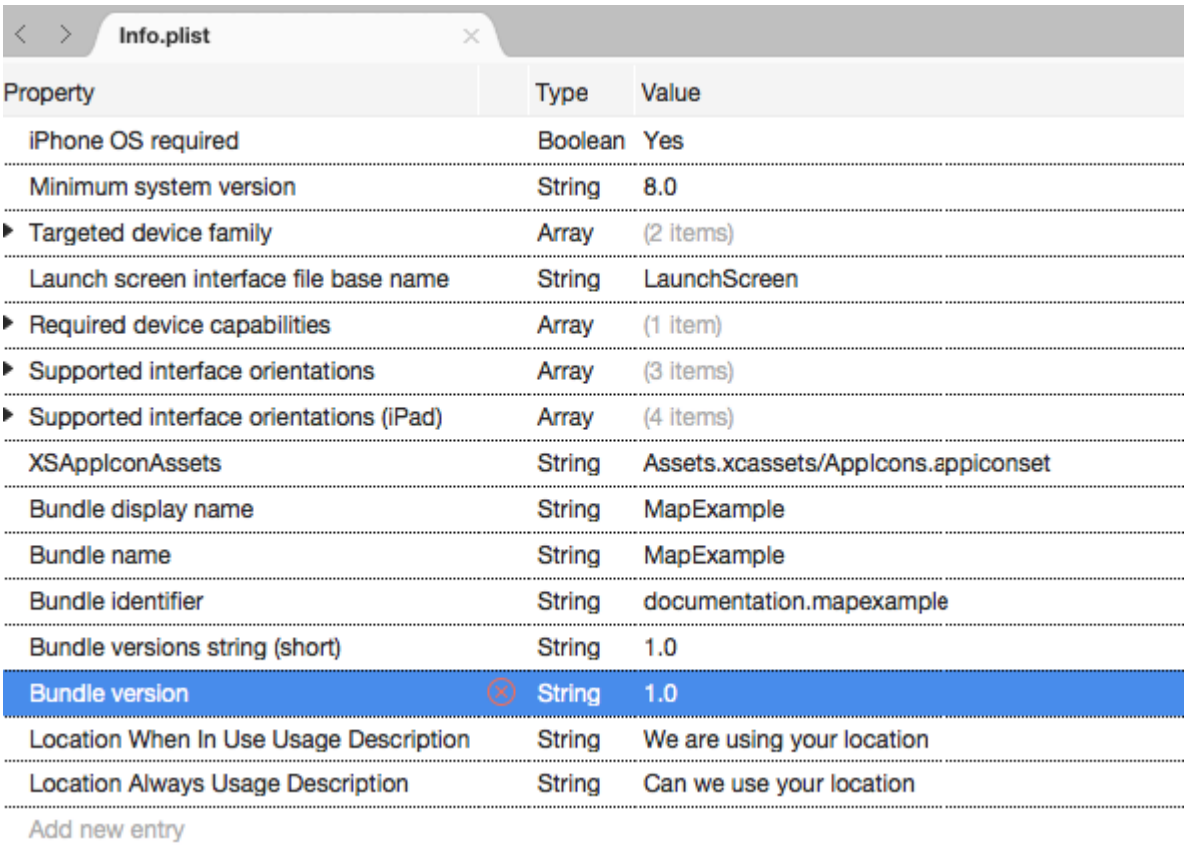
Platform Configuration

Additional configuration steps are required on some platforms before the map will display.

iOS project

In iOS project you just have to add 2 entries to your *Info.plist* file:

- `NSLocationWhenInUseUsageDescription` *string* with value `We are using your location`
- `NSLocationAlwaysUsageDescription` *string* with value `Can we use your location`



Android project

To use Google Maps you have to generate an API key and add it to your project. Follow the instruction below to get this key:

1. (Optional) Find where your keytool tool location (default is `/System/Library/Frameworks/JavaVM.framework/Versions/Current/Commands`)

2. (Optional) Open terminal and go to your keytool location:

   ```
   cd /System/Library/Frameworks/JavaVM.framework/Versions/Current/Commands
   ```

3. Run the following keytool command:

   ```
   keytool -list -v -keystore "/Users/[USERNAME]/.local/share/Xamarin/Mono for
   Android/debug.keystore" -alias androiddebugkey -storepass android -keypass android
   ```

   Where [USERNAME] is, obviously, your current user folder. You should get something similar to this in the output:

   ```
   Alias name: androiddebugkey
   Creation date: Jun 30, 2016
   Entry type: PrivateKeyEntry
   Certificate chain length: 1
   Certificate[1]:
   Owner: CN=Android Debug, O=Android, C=US
   Issuer: CN=Android Debug, O=Android, C=US
   Serial number: 4b5ac934
   ```

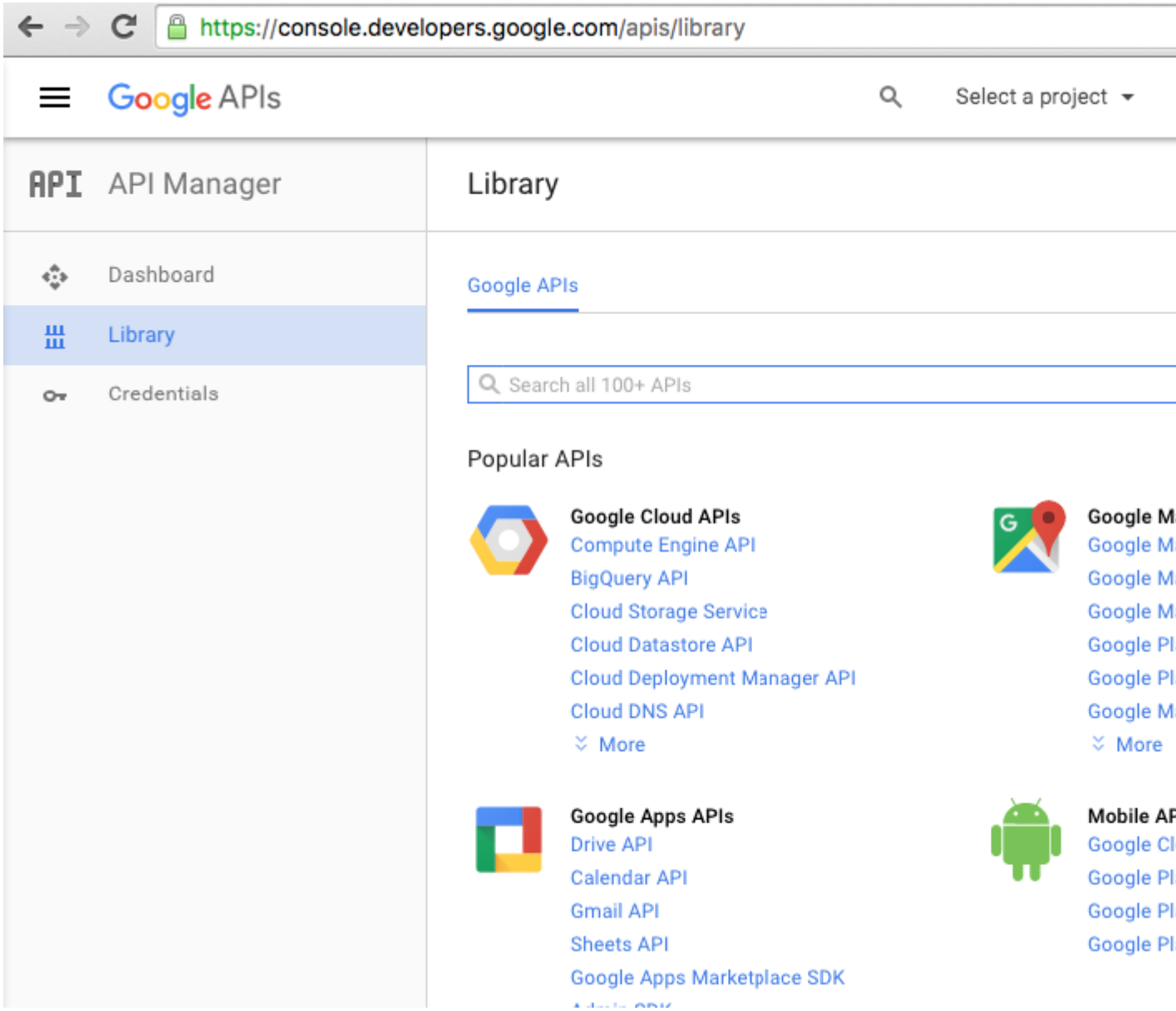
```
Valid from: Thu Jun 30 10:22:00 EEST 2016 until: Sat Jun 23 10:22:00 EEST 2046
Certificate fingerprints:
         MD5:  4E:49:A7:14:99:D6:AB:9F:AA:C7:07:E2:6A:1A:1D:CA
         SHA1: 57:A1:E5:23:CE:49:2F:17:8D:8A:EA:87:65:44:C1:DD:1C:DA:51:95
         SHA256:
70:E1:F3:5B:95:69:36:4A:82:A9:62:F3:67:B6:73:A4:DD:92:95:51:44:E3:4C:3D:9E:ED:99:03:09:9F:90:3F

         Signature algorithm name: SHA256withRSA
         Version: 3
```

4. All we need in this output is the SHA1 certificate fingerprint. In our case it equals to this:

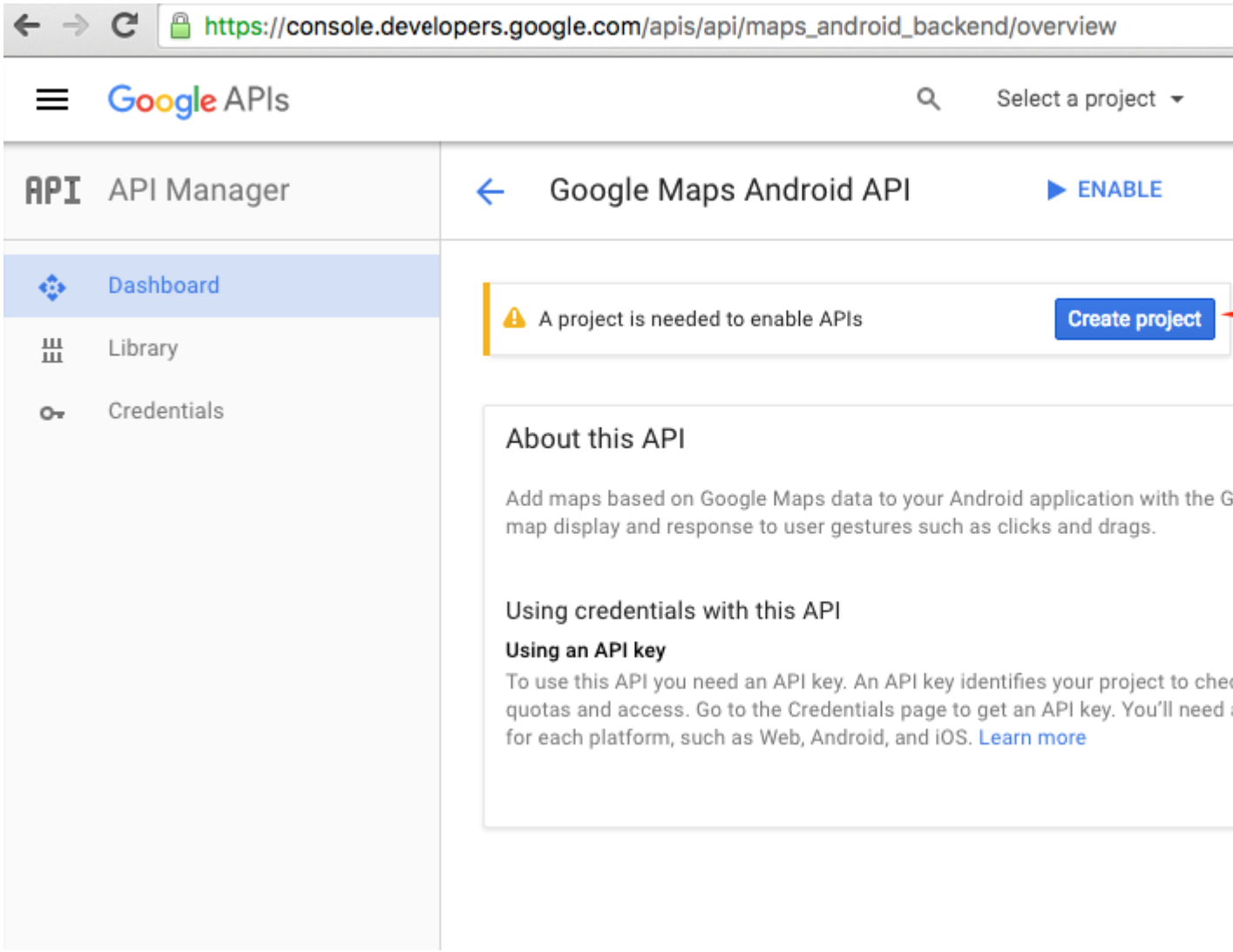
```
57:A1:E5:23:CE:49:2F:17:8D:8A:EA:87:65:44:C1:DD:1C:DA:51:95
```

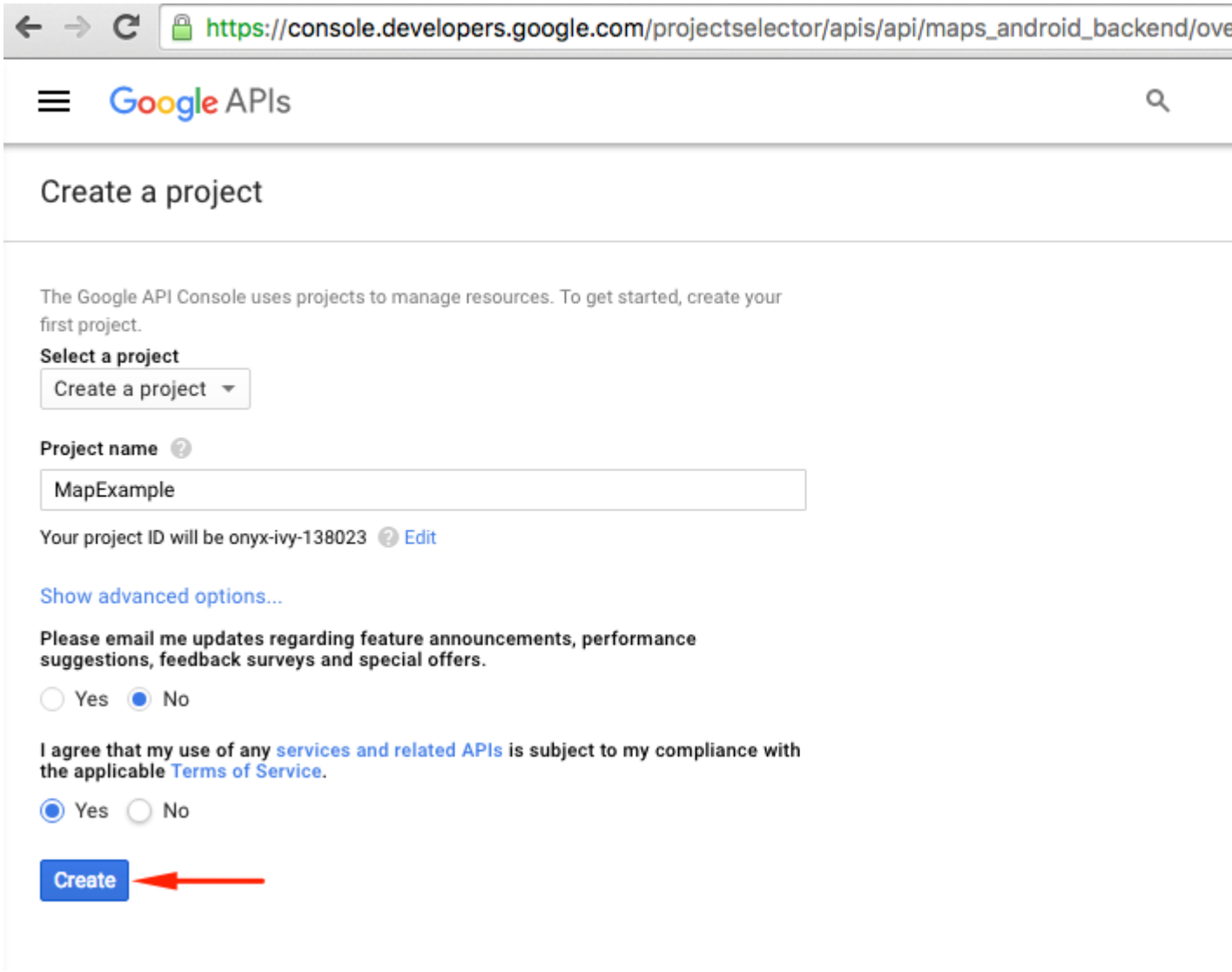
   Copy or save somewhere this key. We will need it later on.

5. Go to Google Developers Console, in our case we have to add Google Maps Android API, so choose it:

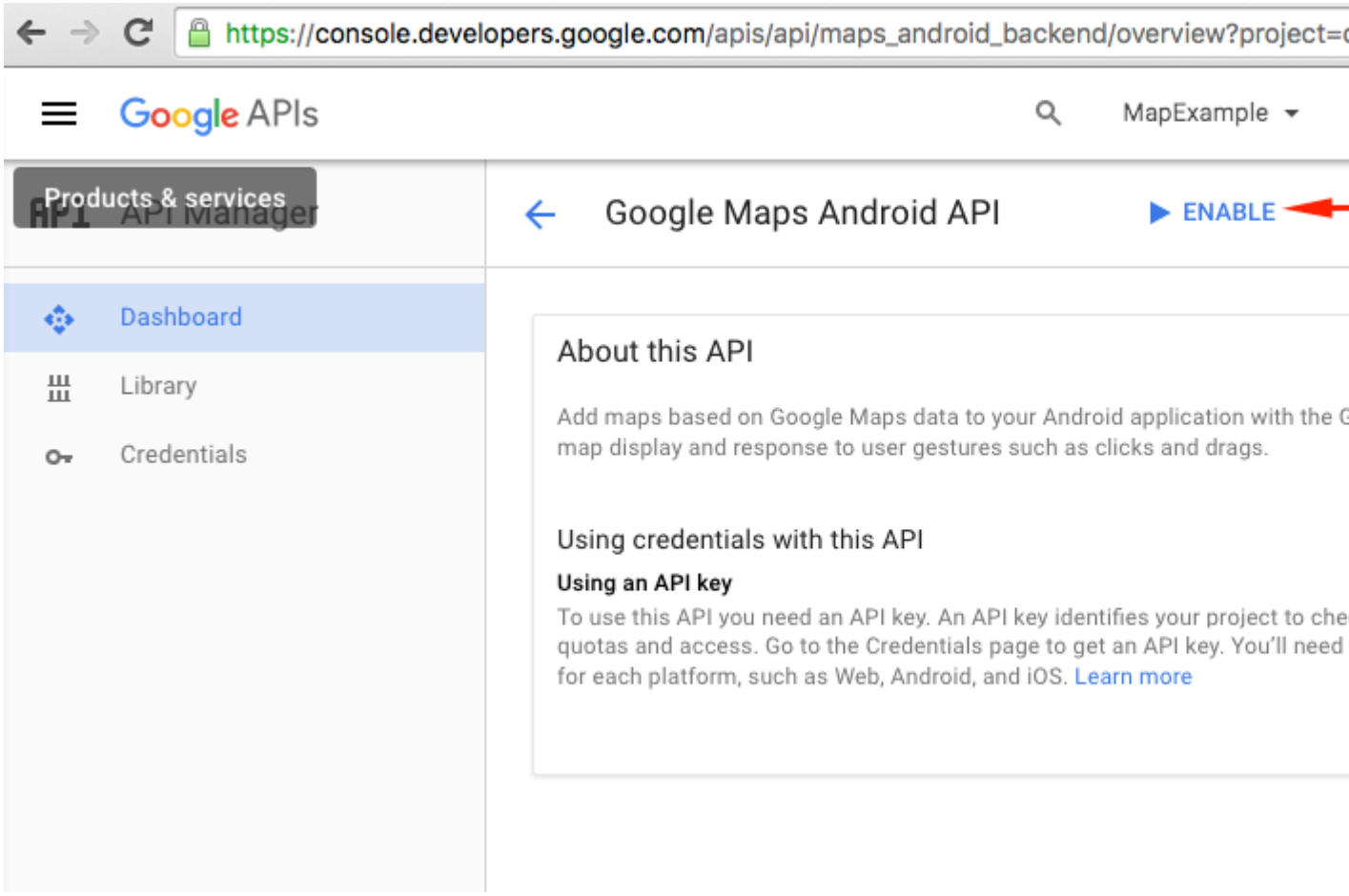


6. Google will ask you to create a project to enable APIs, follow this tip and create the project:

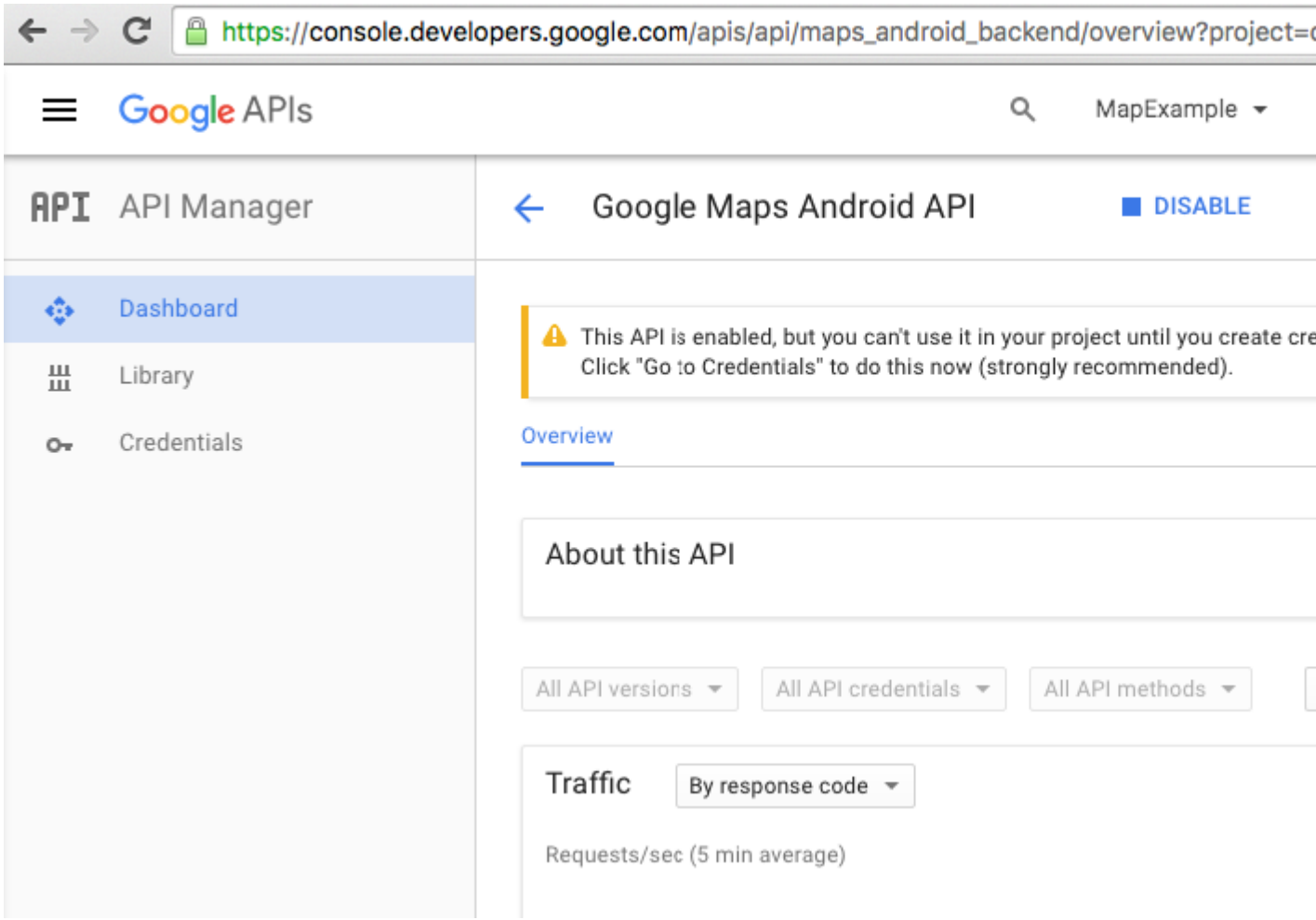

https://console.developers.google.com/apis/api/maps_android_backend/overview
Google APIs
Select a project
API Manager
Dashboard
Library
Credentials
Google Maps Android API
ENABLE
A project is needed to enable APIs
Create project
About this API
Add maps based on Google Maps data to your Android application with the G
map display and response to user gestures such as clicks and drags.
Using credentials with this API
Using an API key
To use this API you need an API key. An API key identifies your project to chec
quotas and access. Go to the Credentials page to get an API key. You'll need
for each platform, such as Web, Android, and iOS. Learn more



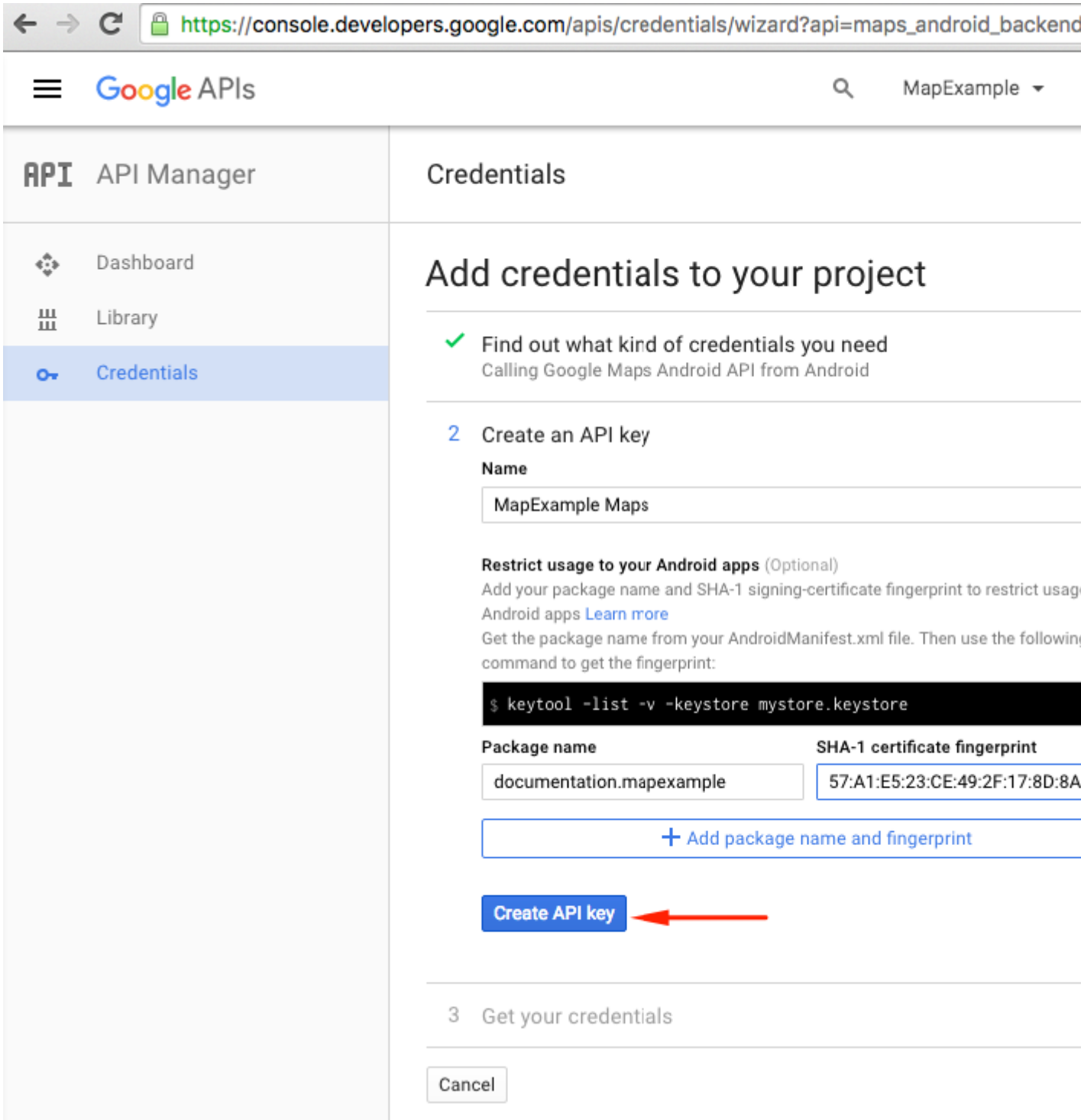
7. Enable Google Maps API for your project:



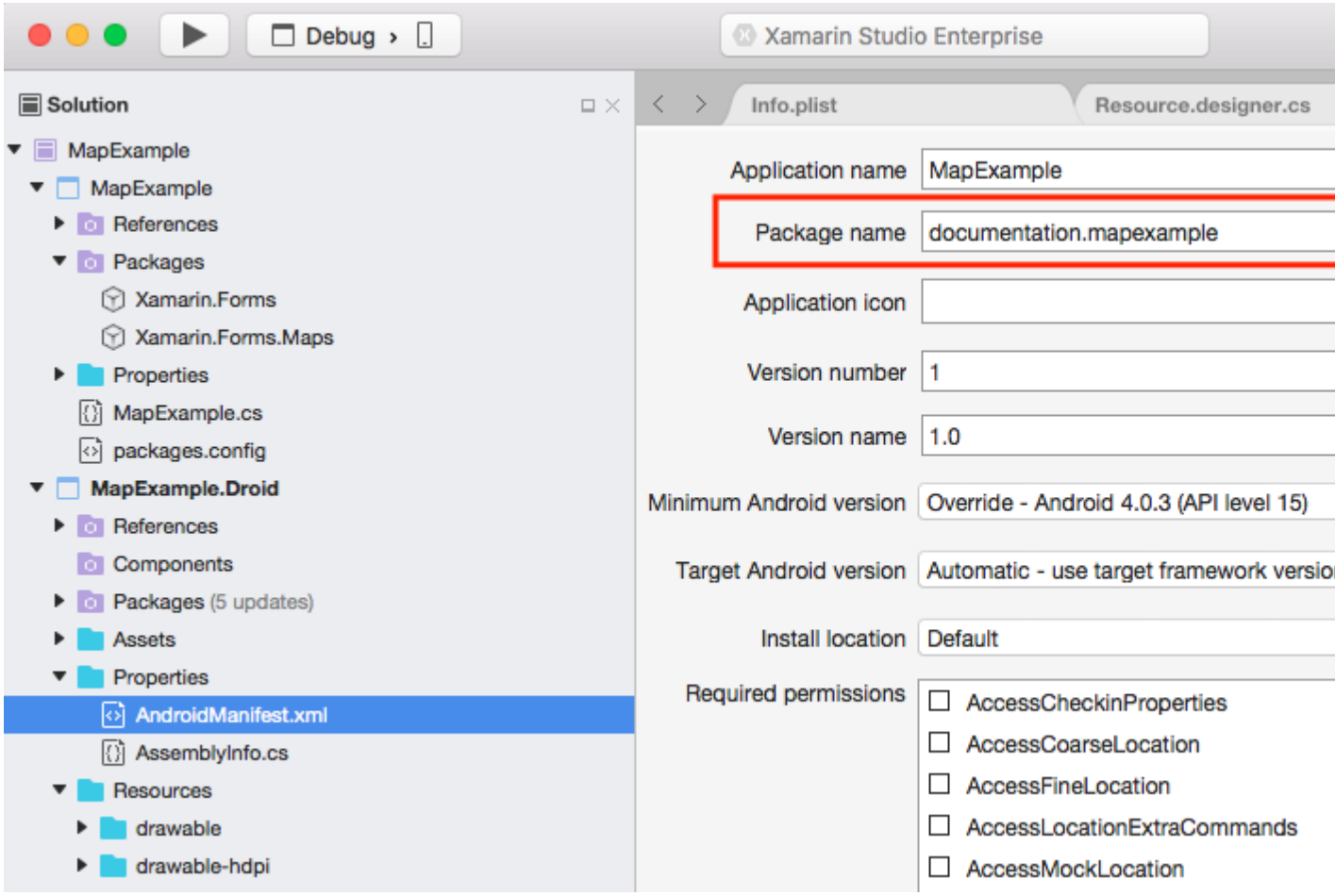
After you have enabled api, you have to create credentials for your app. Follow this tip:



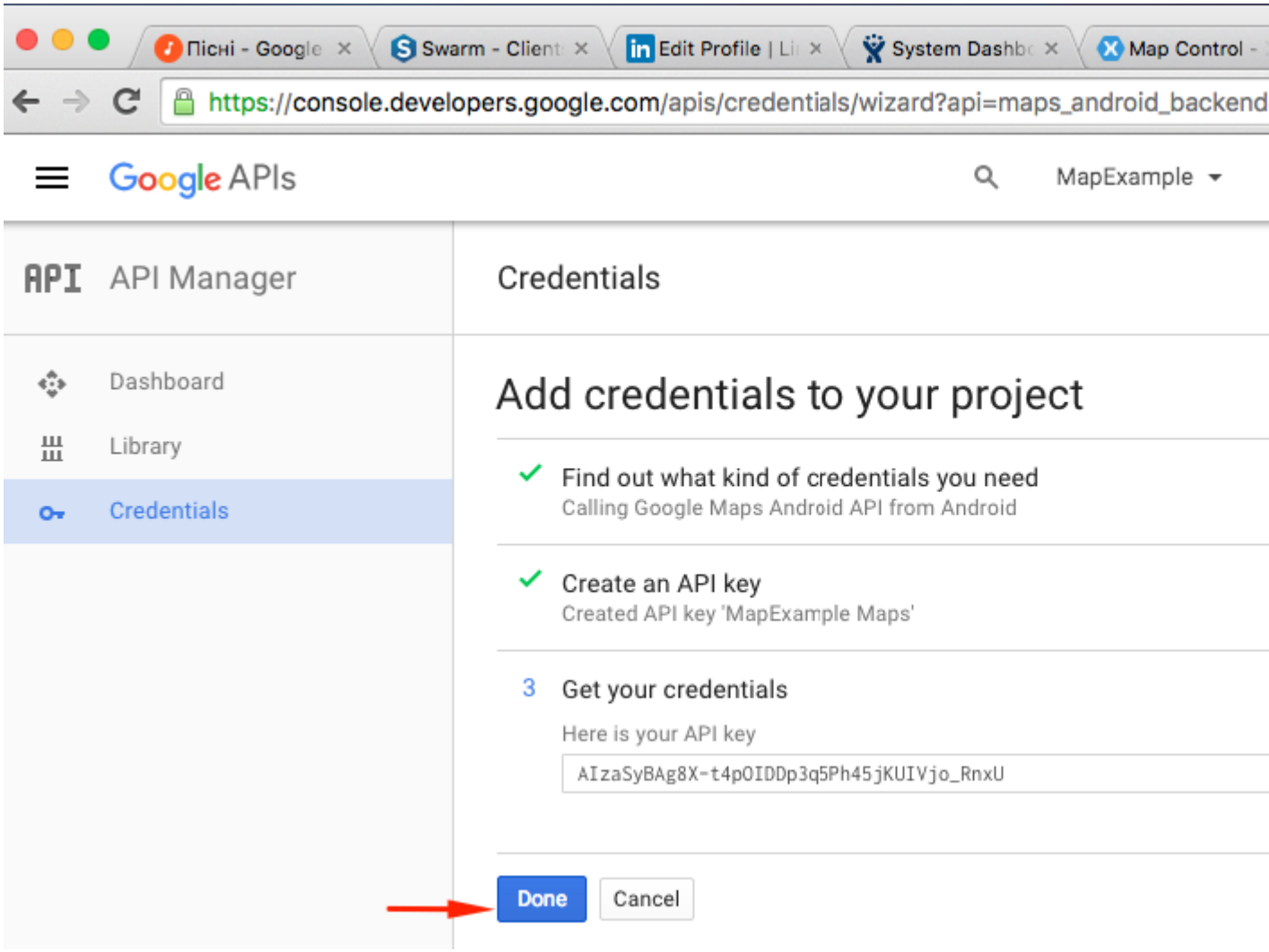
8. On the next page choose the Android platform, tap on "What credentials do I need?" button, create a name for your API key, tap on "Add package name and fingerprint", enter your package name and your SHA1 fingerprint from the step 4 and finally create an API key:



To find your package name in Xamarin Studio go to your .Droid solution -> AndroidManifest.xml:



9. After creation copy the new API key (don't forget to press the "Done" button after) and paste it to your `AndroidManifest.xml` file:



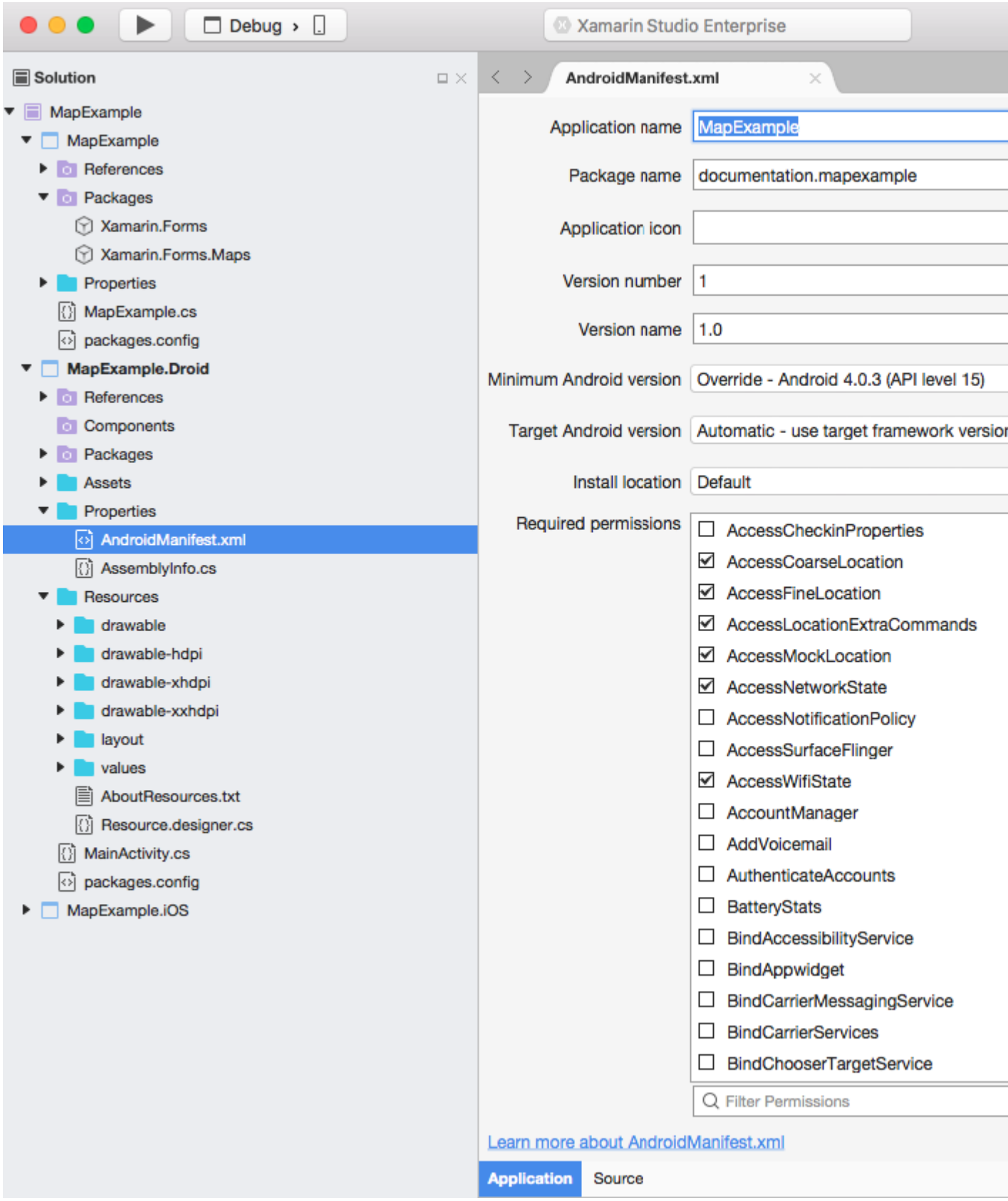
File AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="1"
    android:versionName="1.0"
    package="documentation.mapexample">
    <uses-sdk
        android:minSdkVersion="15" />
    <application
        android:label="MapExample">
        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="AIzaSyBAg8X-t4pOIDDp3q5Ph45jKUIVjo_RnxU" />
        <meta-data
            android:name="com.google.android.gms.version"
            android:value="@integer/google_play_services_version" />
    </application>
</manifest>
```

You'll also need to enable some permissions in your manifest to enable some additional features:

- Access Coarse Location
- Access Fine Location

- Access Location Extra Commands
- Access Mock Location
- Access Network State
- Access Wifi State
- Internet



Although, the last two permissions are required to download Maps data. Read about Android

permissions to learn more. That's all the steps for Android configuration.

> *Note*: if you want to run your app on android simulator, you have to install Google Play Services on it. Follow this tutorial to install Play Services on Xamarin Android Player. If you can't find google play services update after the play store installation, you can update it directly from your app, where you have dependency on maps services

---

Adding a map

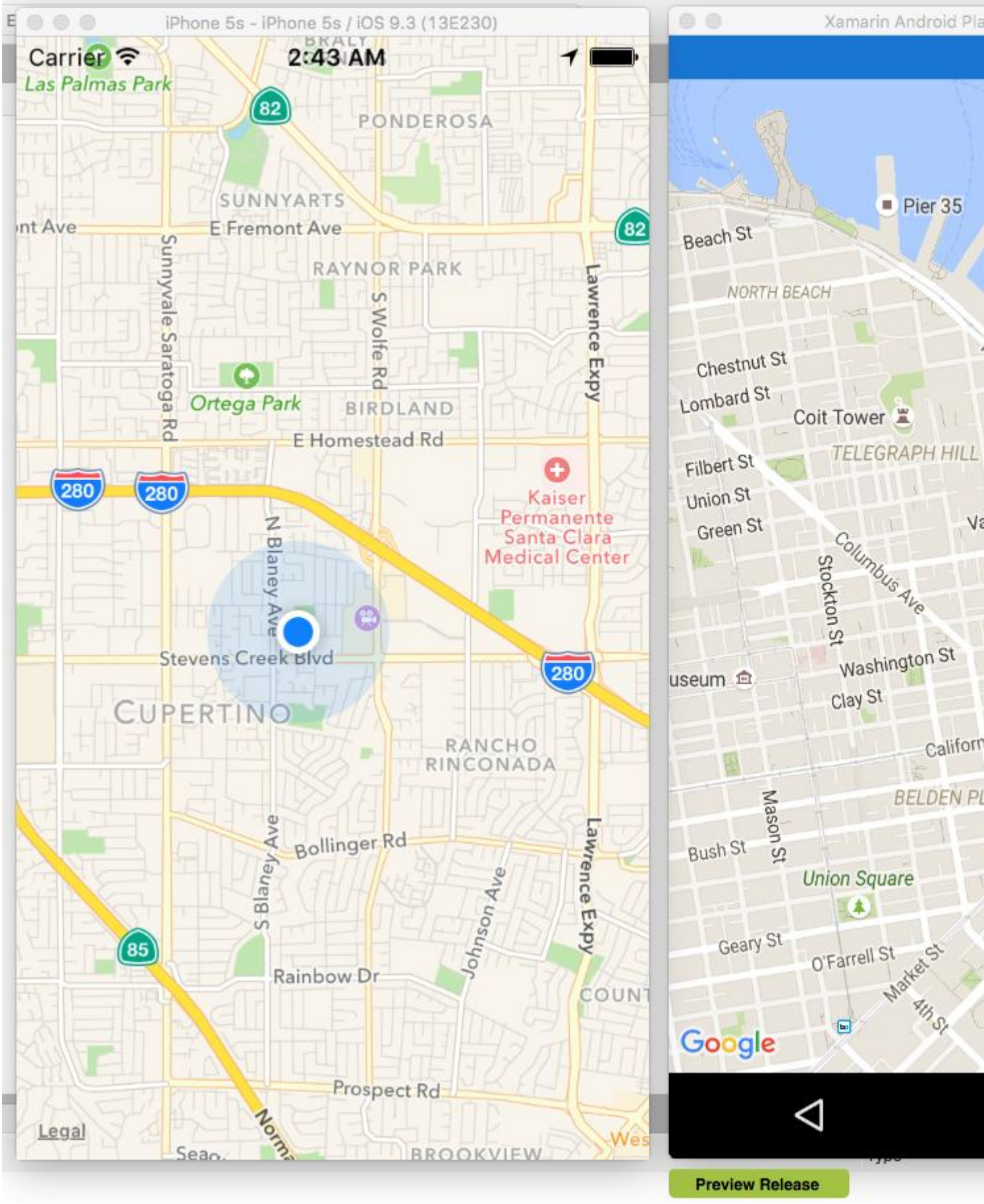
Adding map view to your crossplatform project is quite simple. Here is an example of how you can do it (I'm using PCL project without XAML).

PCL project

File MapExample.cs

```
public class App : Application
{
    public App()
    {
        var map = new Map();
        map.IsShowingUser = true;

        var rootPage = new ContentPage();
        rootPage.Content = map;

        MainPage = rootPage;
    }
}
```

That's all. Now if you'll run your app on iOS or Android, it will show you the map view:



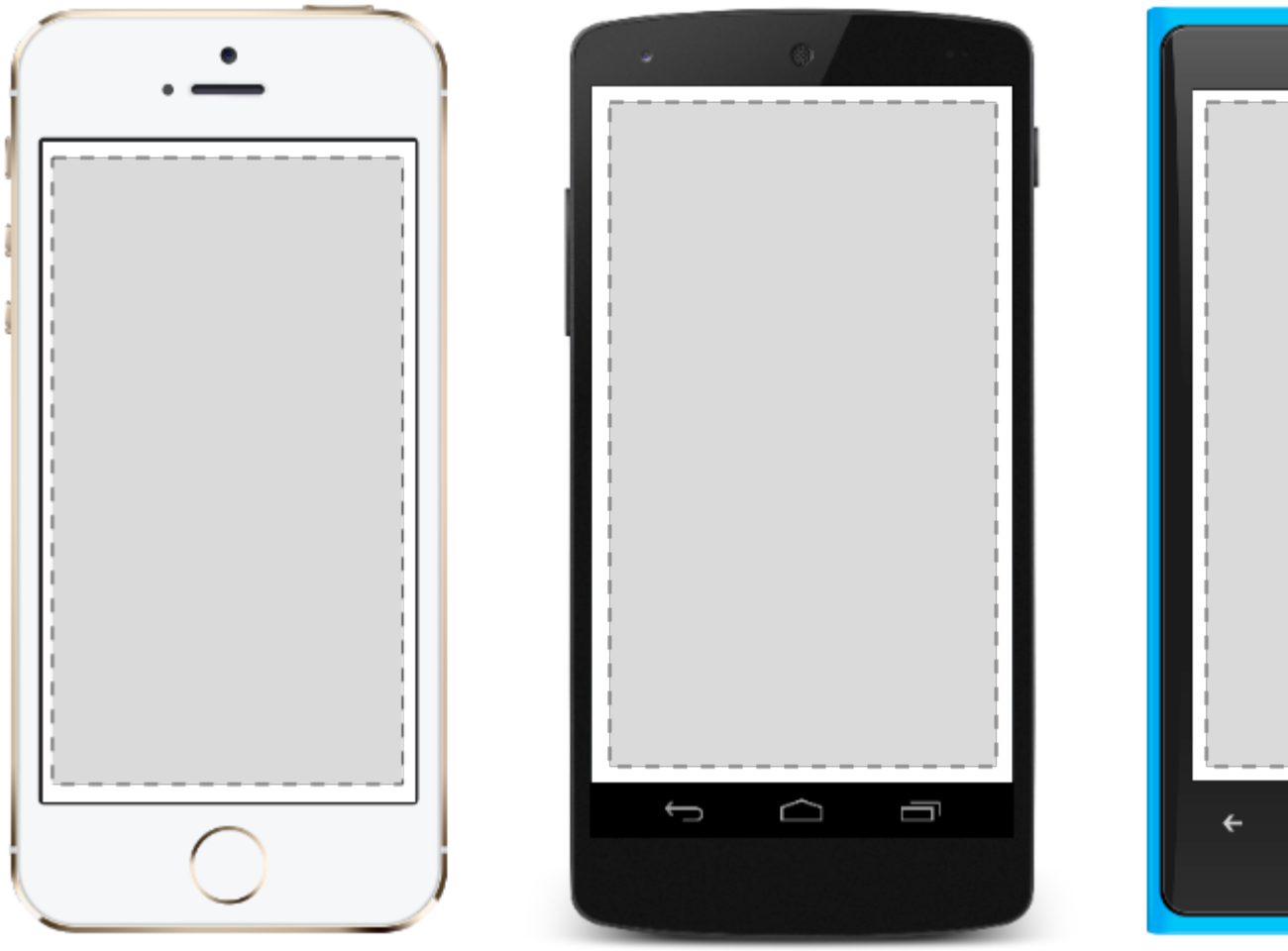
Read Working with Maps online: https://riptutorial.com/xamarin-forms/topic/3917/working-with-maps

Chapter 36: Xamarin Forms Layouts

Examples

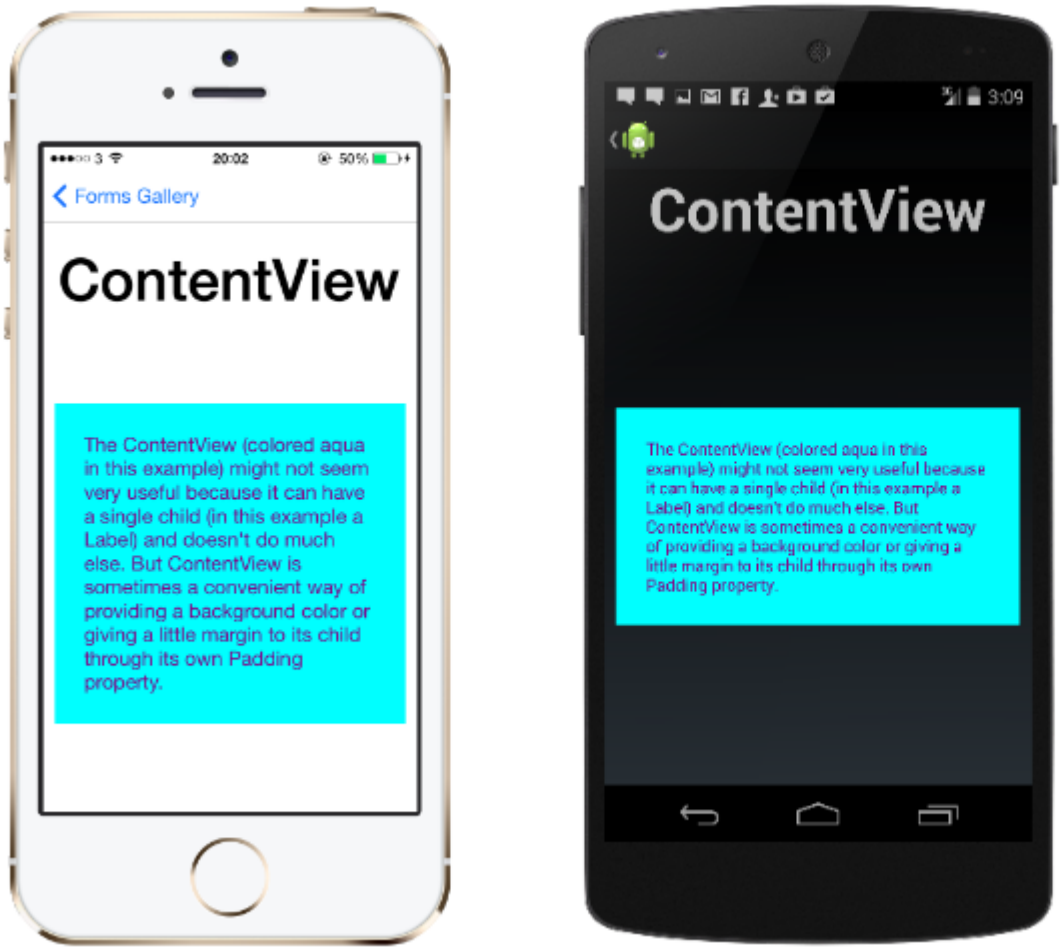
ContentPresenter

A layout manager for templated views. Used within a ControlTemplate to mark where the content to be presented appears.



ContentView

An element with a single content. ContentView has very little use of its own. Its purpose is to serve as a base class for user-defined compound views.



XAML

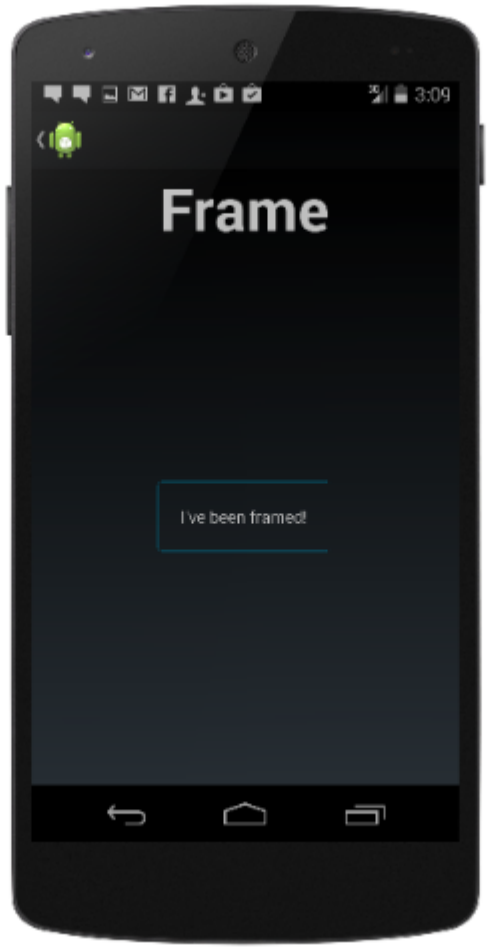
```
<ContentView>
<Label Text="Hi, I'm a simple Label inside of a simple ContentView"
HorizontalOptions="Center"
VerticalOptions="Center"/>
</ContentView>
```

Code

```
var contentView = new ContentView {
Content = new Label {
 Text = "Hi, I'm a simple Label inside of a simple ContentView",
 HorizontalOptions = LayoutOptions.Center,
 VerticalOptions = LayoutOptions.Center
}
};
```

Frame

An element containing a single child, with some framing options. Frame have a default Xamarin.Forms.Layout.Padding of 20.



XAML

```
<Frame>
<Label Text="I've been framed!"
HorizontalOptions="Center"
VerticalOptions="Center" />
</Frame>
```

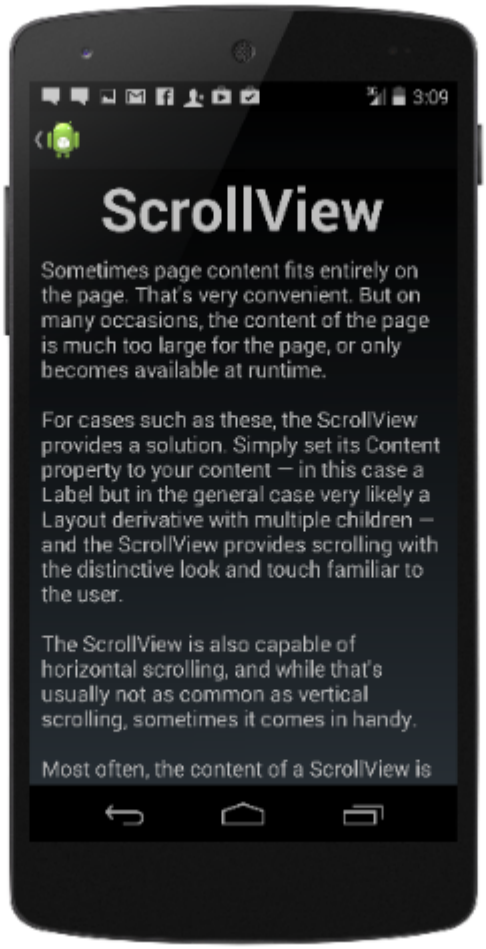
Code

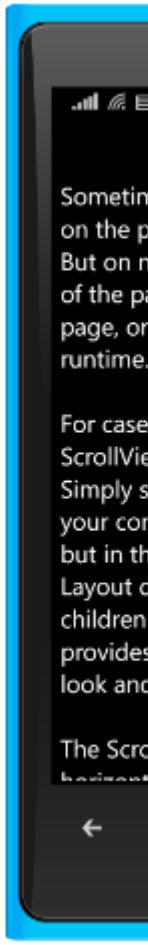
```
var frameView = new Frame {
Content = new Label {
 Text = "I've been framed!",
HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center
},
OutlineColor = Color.Red
};
```

ScrollView

An element capable of scrolling if it's `Content` requires.

`ScrollView` contains layouts and enables them to scroll offscreen. `ScrollView` is also used to allow views to automatically move to the visible portion of the screen when the keyboard is showing.





Note: `ScrollViews` should not be nested. In addition, `ScrollViews` should not be nested with other controls that provide scrolling, like `ListView` and `WebView`.

A `ScrollView` is easy to define. In XAML:

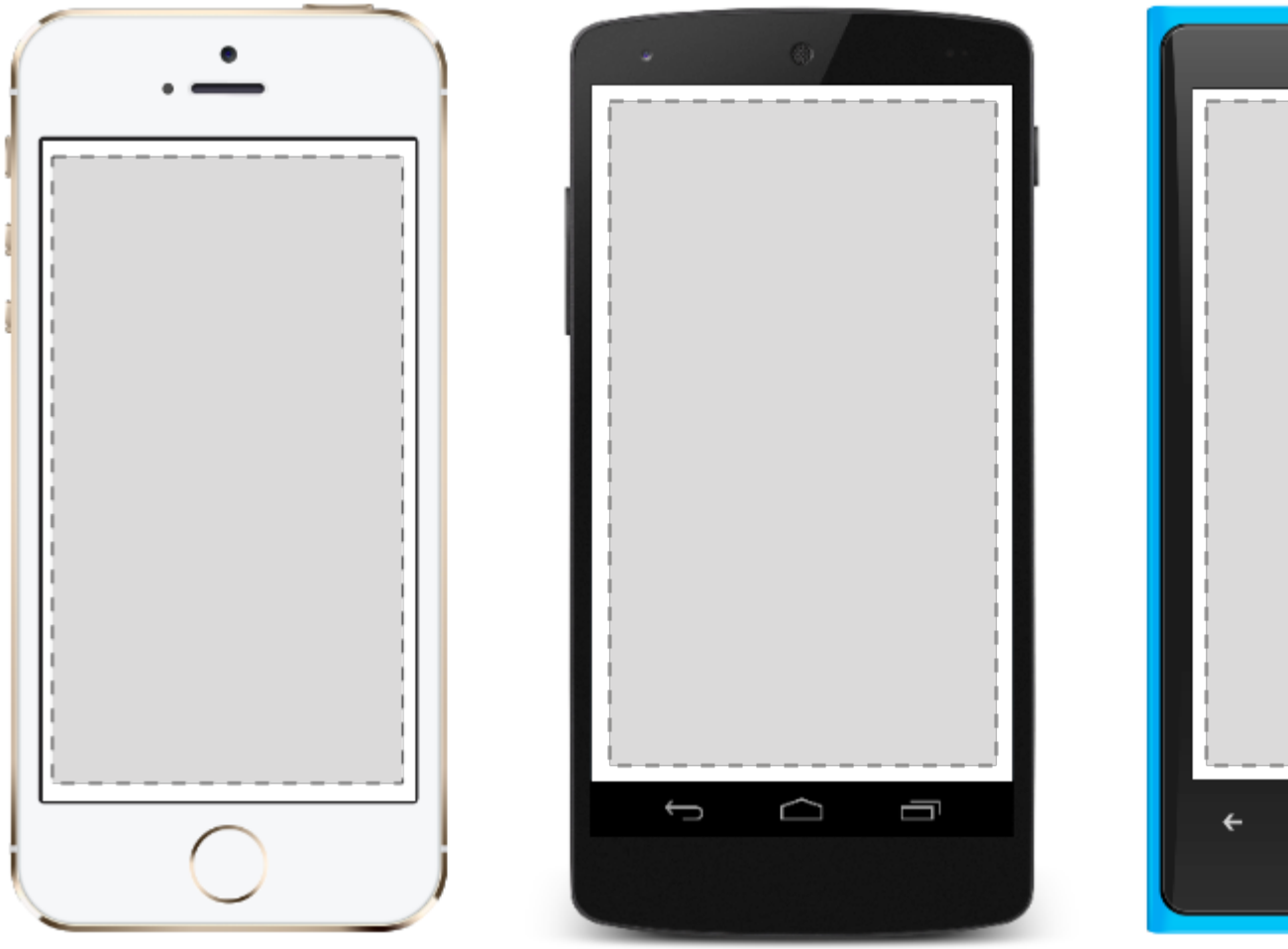
```
<ContentPage.Content>
    <ScrollView>
        <StackLayout>
            <BoxView BackgroundColor="Red" HeightRequest="600" WidthRequest="150" />
            <Entry />
        </StackLayout>
    </ScrollView>
</ContentPage.Content>
```

The same definition in code:

```
var scroll = new ScrollView();
Content = scroll;
var stack = new StackLayout();
stack.Children.Add(new BoxView { BackgroundColor = Color.Red,   HeightRequest = 600,
WidthRequest = 600 });
stack.Children.Add(new Entry());
```

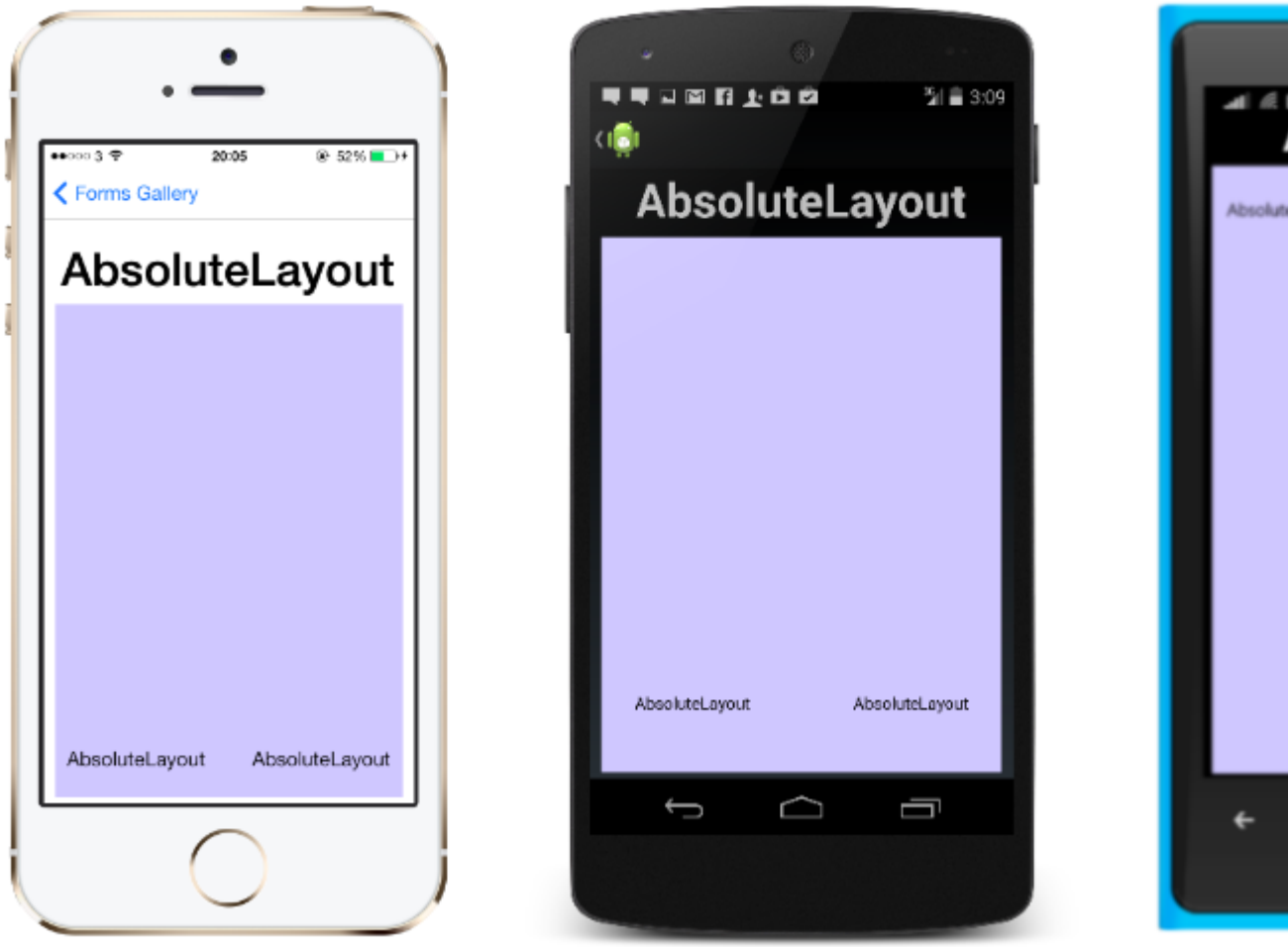
TemplatedView

An element that displays content with a control template, and the base class for `ContentView`.



AbsoluteLayout

`AbsoluteLayout` positions and sizes child elements proportional to its own size and position or by absolute values. Child views may be positioned and sized using proportional values or static values, and proportional and static values can be mixed.



A definition of an `AbsoluteLayout` in XAML looks like this:

```
<AbsoluteLayout>
    <Label Text="I'm centered on iPhone 4 but no other device"
        AbsoluteLayout.LayoutBounds="115,150,100,100" LineBreakMode="WordWrap"  />
    <Label Text="I'm bottom center on every device."
        AbsoluteLayout.LayoutBounds=".5,1,.5,.1" AbsoluteLayout.LayoutFlags="All"
        LineBreakMode="WordWrap"  />
    <BoxView Color="Olive"  AbsoluteLayout.LayoutBounds="1,.5, 25, 100"
        AbsoluteLayout.LayoutFlags="PositionProportional" />
    <BoxView Color="Red" AbsoluteLayout.LayoutBounds="0,.5,25,100"
        AbsoluteLayout.LayoutFlags="PositionProportional" />
    <BoxView Color="Blue" AbsoluteLayout.LayoutBounds=".5,0,100,25"
        AbsoluteLayout.LayoutFlags="PositionProportional" />
    <BoxView Color="Blue" AbsoluteLayout.LayoutBounds=".5,0,1,25"
        AbsoluteLayout.LayoutFlags="PositionProportional, WidthProportional" />
</AbsoluteLayout>
```

The same layout would look like this in code:

```
Title = "Absolute Layout Exploration - Code";
var layout = new AbsoluteLayout();

var centerLabel = new Label {
Text = "I'm centered on iPhone 4 but no other device.",
LineBreakMode = LineBreakMode.WordWrap};
```

```
AbsoluteLayout.SetLayoutBounds (centerLabel, new Rectangle (115, 159, 100, 100));
// No need to set layout flags, absolute positioning is the default

var bottomLabel = new Label { Text = "I'm bottom center on every device.", LineBreakMode =
LineBreakMode.WordWrap };
AbsoluteLayout.SetLayoutBounds (bottomLabel, new Rectangle (.5, 1, .5, .1));
AbsoluteLayout.SetLayoutFlags (bottomLabel, AbsoluteLayoutFlags.All);

var rightBox = new BoxView{ Color = Color.Olive };
AbsoluteLayout.SetLayoutBounds (rightBox, new Rectangle (1, .5, 25, 100));
AbsoluteLayout.SetLayoutFlags (rightBox, AbsoluteLayoutFlags.PositionProportional);

var leftBox = new BoxView{ Color = Color.Red };
AbsoluteLayout.SetLayoutBounds (leftBox, new Rectangle (0, .5, 25, 100));
AbsoluteLayout.SetLayoutFlags (leftBox, AbsoluteLayoutFlags.PositionProportional);

var topBox = new BoxView{ Color = Color.Blue };
AbsoluteLayout.SetLayoutBounds (topBox, new Rectangle (.5, 0, 100, 25));
AbsoluteLayout.SetLayoutFlags (topBox, AbsoluteLayoutFlags.PositionProportional);

var twoFlagsBox = new BoxView{ Color = Color.Blue };
AbsoluteLayout.SetLayoutBounds (topBox, new Rectangle (.5, 0, 1, 25));
AbsoluteLayout.SetLayoutFlags (topBox, AbsoluteLayoutFlags.PositionProportional |
AbsoluteLayout.WidthProportional);

layout.Children.Add (bottomLabel);
layout.Children.Add (centerLabel);
layout.Children.Add (rightBox);
layout.Children.Add (leftBox);
layout.Children.Add (topBox);
```

The AbsoluteLayout control in Xamarin.Forms allows you to specify where exactly on the screen you want the child elements to appear, as well as their size and shape (bounds).

There are a few different ways to set the bounds of the child elements based on the AbsoluteLayoutFlags enumeration that are used during this process. The **AbsoluteLayoutFlags** enumeration contains the following values:

- **All**: All dimensions are proportional.
- **HeightProportional**: Height is proportional to the layout.
- **None**: No interpretation is done.
- **PositionProportional**: Combines XProportional and YProportional.
- **SizeProportional**: Combines WidthProportional and HeightProportional.
- **WidthProportional**: Width is proportional to the layout.
- **XProportional**: X property is proportional to the layout.
- **YProportional**: Y property is proportional to the layout.

The process of working with the layout of the AbsoluteLayout container may seem a little counterintuitive at first, but with a little use it will become familiar. Once you have created your child elements, to set them at an absolute position within the container you will need to follow three steps. You will want to set the flags assigned to the elements using the **AbsoluteLayout.SetLayoutFlags()** method. You will also want to use the **AbsoluteLayout.SetLayoutBounds()** method to give the elements their bounds. Finally, you will want to add the child elements to the Children collection. Since Xamarin.Forms is an abstraction

layer between Xamarin and the device-specific implementations, the positional values can be independent of the device pixels. This is where the layout flags mentioned previously come into play. You can choose how the layout process of the Xamarin.Forms controls should interpret the values you define.

Grid

A layout containing views arranged in rows and columns.

This is a typical `Grid` definition in XAML.

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="2*" />
    <RowDefinition Height="*" />
    <RowDefinition Height="200" />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="*" />
  </Grid.ColumnDefinitions>

  <ContentView Grid.Row="0" Grid.Column="0"/>
  <ContentView Grid.Row="1" Grid.Column="0"/>
  <ContentView Grid.Row="2" Grid.Column="0"/>
```

```
  <ContentView Grid.Row="0" Grid.Column="1"/>
  <ContentView Grid.Row="1" Grid.Column="1"/>
  <ContentView Grid.Row="2" Grid.Column="1"/>

</Grid>
```

The same `Grid` defined in code looks like this:

```
var grid = new Grid();
grid.RowDefinitions.Add (new RowDefinition { Height = new GridLength(2, GridUnitType.Star) });
grid.RowDefinitions.Add (new RowDefinition { Height = new GridLength (1, GridUnitType.Star)
});
grid.RowDefinitions.Add (new RowDefinition { Height = new GridLength(200)});
grid.ColumnDefinitions.Add (new ColumnDefinition{ Width = new GridLength (200) });
```

To add items to the grid: In `XAML`:

```
<Grid>

   <--DEFINITIONS...--!>

   <ContentView Grid.Row="0" Grid.Column="0"/>
   <ContentView Grid.Row="1" Grid.Column="0"/>
   <ContentView Grid.Row="2" Grid.Column="0"/>

   <ContentView Grid.Row="0" Grid.Column="1"/>
   <ContentView Grid.Row="1" Grid.Column="1"/>
   <ContentView Grid.Row="2" Grid.Column="1"/>

</Grid>
```

In C# code:

```
var grid = new Grid();
//DEFINITIONS...
var topLeft = new Label { Text = "Top Left" };
var topRight = new Label { Text = "Top Right" };
var bottomLeft = new Label { Text = "Bottom Left" };
var bottomRight = new Label { Text = "Bottom Right" };
grid.Children.Add(topLeft, 0, 0);
grid.Children.Add(topRight, 0, 1);
grid.Children.Add(bottomLeft, 1, 0);
grid.Children.Add(bottomRight, 1, 1);
```

For `Height` and `Width` a number of units are available.

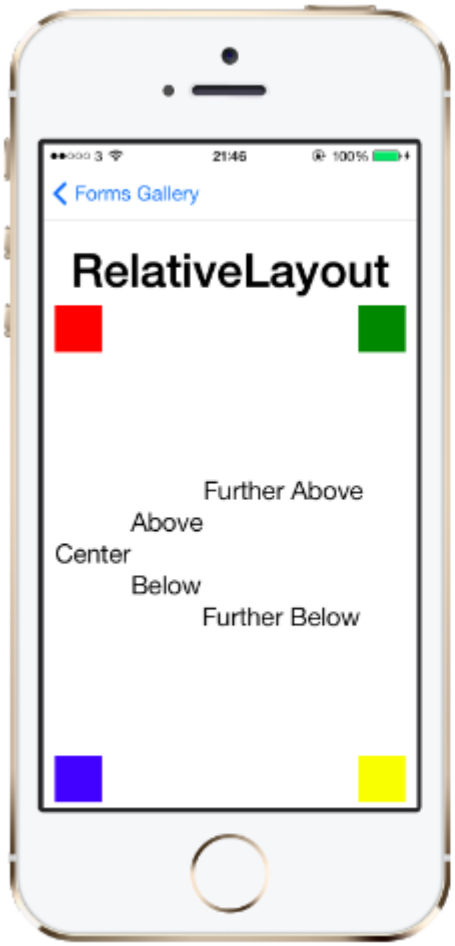
- **Auto** – automatically sizes to fit content in the row or column. Specified as GridUnitType.Auto in C# or as Auto in XAML.
- **Proportional** – sizes columns and rows as a proportion of the remaining space. Specified as a value and GridUnitType.Star in C# and as #* in XAML, with # being your desired value. Specifying one row/column with * will cause it to fill the available space.
- **Absolute** – sizes columns and rows with specific, fixed height and width values. Specified as a value and GridUnitType.Absolute in C# and as # in XAML, with # being your desired value.

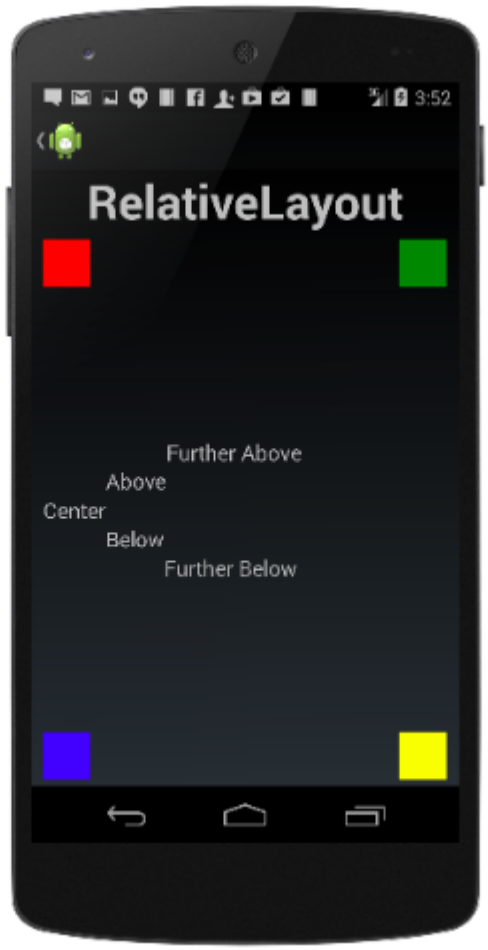
Note: The width values for columns are set as Auto by default in Xamarin.Forms, which means that the width is determined from the size of the children. Note that this differs from the implementation of XAML on Microsoft platforms, where the default width is *, which will fill the available space.

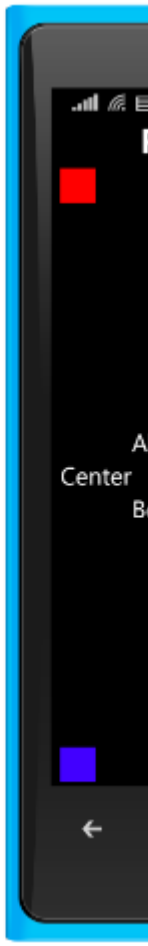
RelativeLayout

A `Layout` that uses `Constraints` to layout its children.

`RelativeLayout` is used to position and size views relative to properties of the layout or sibling views. Unlike `AbsoluteLayout`, `RelativeLayout` does not have the concept of the moving anchor and does not have facilities for positioning elements relative to the bottom or right edges of the layout. `RelativeLayout` does support positioning elements outside of its own bounds.







A RelativeLayout in XAML, is like this:

```
<RelativeLayout>
    <BoxView Color="Red" x:Name="redBox"
        RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToParent,
            Property=Height,Factor=.15,Constant=0}"
        RelativeLayout.WidthConstraint="{ConstraintExpression
            Type=RelativeToParent,Property=Width,Factor=1,Constant=0}"
        RelativeLayout.HeightConstraint="{ConstraintExpression
            Type=RelativeToParent,Property=Height,Factor=.8,Constant=0}" />
    <BoxView Color="Blue"
```

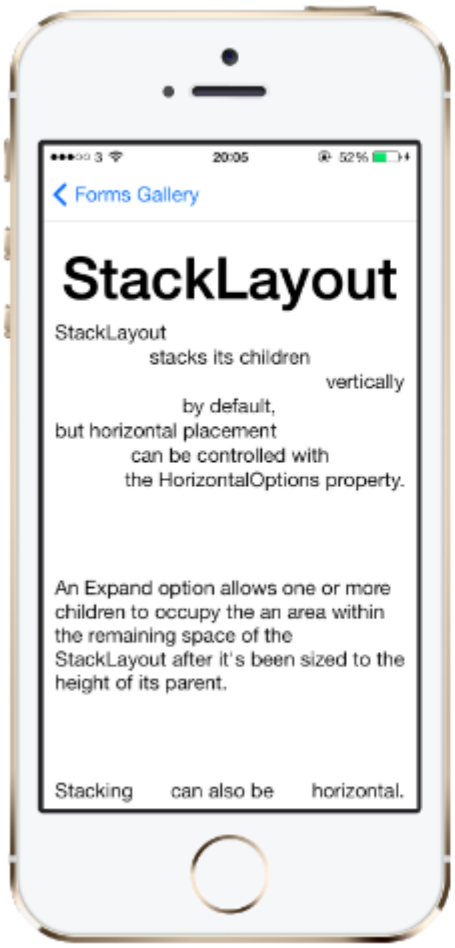
```
        RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToView,
            ElementName=redBox,Property=Y,Factor=1,Constant=20}"
        RelativeLayout.XConstraint="{ConstraintExpression Type=RelativeToView,
            ElementName=redBox,Property=X,Factor=1,Constant=20}"
        RelativeLayout.WidthConstraint="{ConstraintExpression
            Type=RelativeToParent,Property=Width,Factor=.5,Constant=0}"
        RelativeLayout.HeightConstraint="{ConstraintExpression
            Type=RelativeToParent,Property=Height,Factor=.5,Constant=0}" />
</RelativeLayout>
```

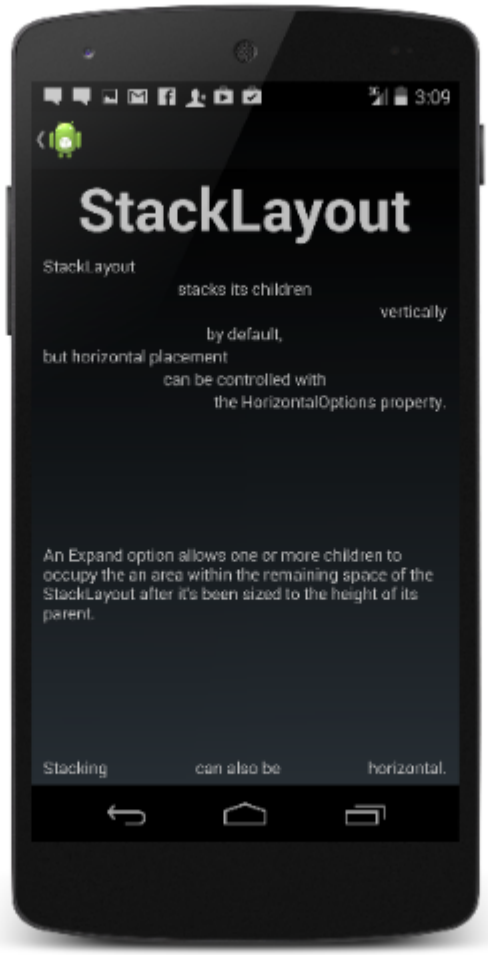
The same layout can be accomplished with this code:

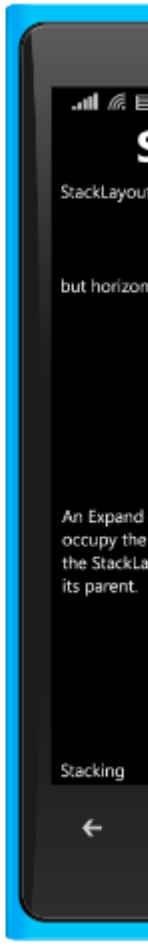
```
layout.Children.Add (redBox, Constraint.RelativeToParent ((parent) => {
        return parent.X;
    }), Constraint.RelativeToParent ((parent) => {
        return parent.Y * .15;
    }), Constraint.RelativeToParent((parent) => {
        return parent.Width;
    }), Constraint.RelativeToParent((parent) => {
        return parent.Height * .8;
    }));
layout.Children.Add (blueBox, Constraint.RelativeToView (redBox, (Parent, sibling) => {
        return sibling.X + 20;
    }), Constraint.RelativeToView (blueBox, (parent, sibling) => {
        return sibling.Y + 20;
    }), Constraint.RelativeToParent((parent) => {
        return parent.Width * .5;
    }), Constraint.RelativeToParent((parent) => {
        return parent.Height * .5;
    }));
```

StackLayout

`StackLayout` organizes views in a one-dimensional line ("stack"), either horizontally or vertically. `Views` in a `StackLayout` can be sized based on the space in the layout using layout options. Positioning is determined by the order views were added to the layout and the layout options of the views.







Usage in XAML

```
<StackLayout>
    <Label Text="This will be on top" />
    <Button Text="This will be on the bottom" />
</StackLayout>
```

Usage in code

```
StackLayout stackLayout = new StackLayout
{
    Spacing = 0,
    VerticalOptions = LayoutOptions.FillAndExpand,
    Children =
    {
        new Label
        {
            Text = "StackLayout",
            HorizontalOptions = LayoutOptions.Start
        },
        new Label
        {
            Text = "stacks its children",
```

```
            HorizontalOptions = LayoutOptions.Center
        },
        new Label
        {
            Text = "vertically",
            HorizontalOptions = LayoutOptions.End
        },
        new Label
        {
            Text = "by default,",
            HorizontalOptions = LayoutOptions.Center
        },
        new Label
        {
            Text = "but horizontal placement",
            HorizontalOptions = LayoutOptions.Start
        },
        new Label
        {
            Text = "can be controlled with",
            HorizontalOptions = LayoutOptions.Center
        },
        new Label
        {
            Text = "the HorizontalOptions property.",
            HorizontalOptions = LayoutOptions.End
        },
        new Label
        {
            Text = "An Expand option allows one or more children " +
                   "to occupy the an area within the remaining " +
                   "space of the StackLayout after it's been sized " +
                   "to the height of its parent.",
            VerticalOptions = LayoutOptions.CenterAndExpand,
            HorizontalOptions = LayoutOptions.End
        },
        new StackLayout
        {
            Spacing = 0,
            Orientation = StackOrientation.Horizontal,
            Children =
            {
                new Label
                {
                    Text = "Stacking",
                },
                new Label
                {
                    Text = "can also be",
                    HorizontalOptions = LayoutOptions.CenterAndExpand
                },
                new Label
                {
                    Text = "horizontal.",
                },
            }
        }
    }
};
```

Read Xamarin Forms Layouts online: https://riptutorial.com/xamarin-forms/topic/6273/xamarin-

forms-layouts

Chapter 37: Xamarin Gesture

Examples

Tap Gesture

With the Tap Gesture, you can make any UI-Element clickable (Images, Buttons, StackLayouts, ...):

(1) In code, using the event:

```
var tapGestureRecognizer = new TapGestureRecognizer();
tapGestureRecognizer.Tapped += (s, e) => {
    // handle the tap
};
image.GestureRecognizers.Add(tapGestureRecognizer);
```

(2) In code, using `ICommand` (with MVVM-Pattern, for example):

```
var tapGestureRecognizer = new TapGestureRecognizer();
tapGestureRecognizer.SetBinding (TapGestureRecognizer.CommandProperty, "TapCommand");
image.GestureRecognizers.Add(tapGestureRecognizer);
```

(3) Or in Xaml (with event and `ICommand`, only one is needed):

```
<Image Source="tapped.jpg">
    <Image.GestureRecognizers>
        <TapGestureRecognizer Tapped="OnTapGestureRecognizerTapped" Command="{Binding
TapCommand"} />
  </Image.GestureRecognizers>
</Image>
```

Read Xamarin Gesture online: https://riptutorial.com/xamarin-forms/topic/7994/xamarin-gesture

Chapter 38: Xamarin Gesture

Examples

Gesture Event

When we put the control of Label, the Label does not provide any event. <Label x:Name="lblSignUp Text="Dont't have account?"/> as shown the Label only display purpose only.

When the user want to replace Button with Label, then we give the event for Label. As shown below:

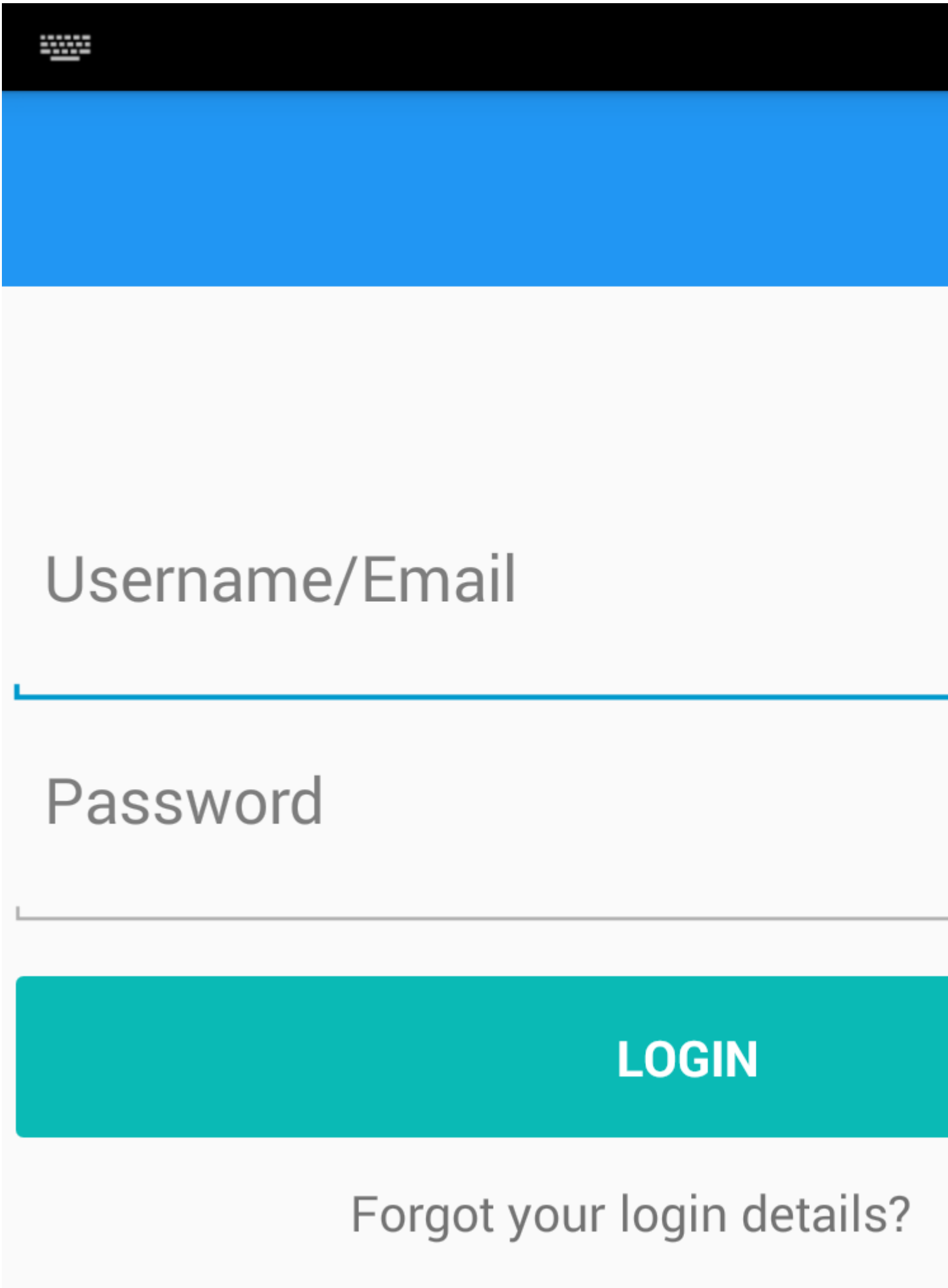
XAML

```
<Label x:Name="lblSignUp" Text="Don't have an account?" Grid.Row="8" Grid.Column="1"
Grid.ColumnSpan="2">
  <Label.GestureRecognizers>
    <TapGestureRecognizer
   Tapped="lblSignUp_Tapped"/>
  </Label.GestureRecognizers>
```

C#

```
var lblSignUp_Tapped = new TapGestureRecognizer();
lblSignUp_Tapped.Tapped += (s,e) =>
{
//
//  Do your work here.
//
};
lblSignUp.GestureRecognizers.Add(lblSignUp_Tapped);
```

The Screen Below shown the Label Event. Screen 1 : The Label "Don't have an account?" as shown in Bottom .


Username/Email
Password
LOGIN
Forgot your login details?

For more details: [https://developer.xamarin.com/guides/xamarin-forms/user-interface/gestures/tap/][1]

Read Xamarin Gesture online: https://riptutorial.com/xamarin-forms/topic/8009/xamarin-gesture

Chapter 39: Xamarin Plugin

Examples

Share Plugin

Simple way to share a message or link, copy text to clipboard, or open a browser in any Xamarin or Windows app.

Available on NuGet : https://www.nuget.org/packages/Plugin.Share/

XAML

```
<StackLayout Padding="20" Spacing="20">
      <Button StyleId="Text" Text="Share Text" Clicked="Button_OnClicked"/>
      <Button StyleId="Link" Text="Share Link" Clicked="Button_OnClicked"/>
      <Button StyleId="Browser" Text="Open Browser" Clicked="Button_OnClicked"/>
      <Label Text=""/>

 </StackLayout>
```

C#

```
 async void Button_OnClicked(object sender, EventArgs e)
        {
            switch (((Button)sender).StyleId)
            {
                case "Text":
                    await CrossShare.Current.Share("Follow @JamesMontemagno on Twitter",
"Share");
                    break;
                case "Link":
                    await CrossShare.Current.ShareLink("http://motzcod.es", "Checkout my
blog", "MotzCod.es");
                    break;
                case "Browser":
                    await CrossShare.Current.OpenBrowser("http://motzcod.es");
                    break;
            }
        }
```

ExternalMaps

External Maps Plugin Open external maps to navigate to a specific geolocation or address. Option to launch with navigation option on iOS as well.

Available on NuGet :[https://www.nuget.org/packages/Xam.Plugin.ExternalMaps/][1]

XAML

```
  <StackLayout Spacing="10" Padding="10">
```

```
      <Button x:Name="navigateAddress" Text="Navigate to Address"/>
      <Button x:Name="navigateLatLong" Text="Navigate to Lat|Long"/>
      <Label Text=""/>

    </StackLayout>
```

Code

```
namespace PluginDemo
{
    public partial class ExternalMaps : ContentPage
    {
        public ExternalMaps()
        {
            InitializeComponent();
            navigateLatLong.Clicked += (sender, args) =>
            {
                CrossExternalMaps.Current.NavigateTo("Space Needle", 47.6204, -122.3491);
            };

            navigateAddress.Clicked += (sender, args) =>
            {
                CrossExternalMaps.Current.NavigateTo("Xamarin", "394 pacific ave.", "San
Francisco", "CA", "94111", "USA", "USA");
            };
        }
    }
}
```

Geolocator Plugin

Easly access geolocation across Xamarin.iOS, Xamarin.Android and Windows.

Available Nuget: [https://www.nuget.org/packages/Xam.Plugin.Geolocator/][1]

XAML

```
 <StackLayout Spacing="10" Padding="10">
      <Button x:Name="buttonGetGPS" Text="Get GPS"/>
      <Label x:Name="labelGPS"/>
      <Button x:Name="buttonTrack" Text="Track Movements"/>
      <Label x:Name="labelGPSTrack"/>
      <Label Text=""/>

    </StackLayout>
```

Code

```
namespace PluginDemo
{
    public partial class GeolocatorPage : ContentPage
    {
        public GeolocatorPage()
        {
            InitializeComponent();
            buttonGetGPS.Clicked += async (sender, args) =>
```

```
            {
                try
                {
                    var locator = CrossGeolocator.Current;
                    locator.DesiredAccuracy = 1000;
                    labelGPS.Text = "Getting gps";

                    var position = await locator.GetPositionAsync(timeoutMilliseconds: 10000);

                    if (position == null)
                    {
                        labelGPS.Text = "null gps :(";
                        return;
                    }
                    labelGPS.Text = string.Format("Time: {0} \nLat: {1} \nLong: {2}
\nAltitude: {3} \nAltitude Accuracy: {4} \nAccuracy: {5} \nHeading: {6} \nSpeed: {7}",
                        position.Timestamp, position.Latitude, position.Longitude,
                        position.Altitude, position.AltitudeAccuracy, position.Accuracy,
position.Heading, position.Speed);

                }
                catch //(Exception ex)
                {
                   // Xamarin.Insights.Report(ex);
                   // await DisplayAlert("Uh oh", "Something went wrong, but don't worry we
captured it in Xamarin Insights! Thanks.", "OK");
                }
            };

            buttonTrack.Clicked += async (object sender, EventArgs e) =>
            {
                try
                {
                    if (CrossGeolocator.Current.IsListening)
                    {
                        await CrossGeolocator.Current.StopListeningAsync();
                        labelGPSTrack.Text = "Stopped tracking";
                        buttonTrack.Text = "Stop Tracking";
                    }
                    else
                    {
                        if (await CrossGeolocator.Current.StartListeningAsync(30000, 0))
                        {
                            labelGPSTrack.Text = "Started tracking";
                            buttonTrack.Text = "Track Movements";
                        }
                    }
                }
                catch //(Exception ex)
                {
                    //Xamarin.Insights.Report(ex);
                   // await DisplayAlert("Uh oh", "Something went wrong, but don't worry we
captured it in Xamarin Insights! Thanks.", "OK");
                }
            };
        }

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

```
            {
                CrossGeolocator.Current.PositionChanged +=
CrossGeolocator_Current_PositionChanged;
                CrossGeolocator.Current.PositionError +=
CrossGeolocator_Current_PositionError;
            }
            catch
            {
            }
        }

        void CrossGeolocator_Current_PositionError(object sender,
Plugin.Geolocator.Abstractions.PositionErrorEventArgs e)
        {

            labelGPSTrack.Text = "Location error: " + e.Error.ToString();
        }

        void CrossGeolocator_Current_PositionChanged(object sender,
Plugin.Geolocator.Abstractions.PositionEventArgs e)
        {
            var position = e.Position;
            labelGPSTrack.Text = string.Format("Time: {0} \nLat: {1} \nLong: {2} \nAltitude:
{3} \nAltitude Accuracy: {4} \nAccuracy: {5} \nHeading: {6} \nSpeed: {7}",
                position.Timestamp, position.Latitude, position.Longitude,
                position.Altitude, position.AltitudeAccuracy, position.Accuracy,
position.Heading, position.Speed);


        }

        protected override void OnDisappearing()
        {
            base.OnDisappearing();
            try
            {
                CrossGeolocator.Current.PositionChanged -=
CrossGeolocator_Current_PositionChanged;
                CrossGeolocator.Current.PositionError -=
CrossGeolocator_Current_PositionError;
            }
            catch
            {
            }
        }
    }
}
```

Media Plugin

Take or pick photos and videos from a cross platform API.

Available Nuget : [https://www.nuget.org/packages/Xam.Plugin.Media/][1]

XAML

```
<StackLayout Spacing="10" Padding="10">
      <Button x:Name="takePhoto" Text="Take Photo"/>
```

```
      <Button x:Name="pickPhoto" Text="Pick Photo"/>
      <Button x:Name="takeVideo" Text="Take Video"/>
      <Button x:Name="pickVideo" Text="Pick Video"/>
      <Label Text="Save to Gallery"/>
      <Switch x:Name="saveToGallery" IsToggled="false" HorizontalOptions="Center"/>
      <Label Text="Image will show here"/>
      <Image x:Name="image"/>
      <Label Text=""/>

    </StackLayout>
```

Code

```
namespace PluginDemo
{
    public partial class MediaPage : ContentPage
    {
        public MediaPage()
        {
            InitializeComponent();
            takePhoto.Clicked += async (sender, args) =>
            {

                if (!CrossMedia.Current.IsCameraAvailable ||
!CrossMedia.Current.IsTakePhotoSupported)
                {
                    await DisplayAlert("No Camera", ":( No camera avaialble.", "OK");
                    return;
                }
                try
                {
                    var file = await CrossMedia.Current.TakePhotoAsync(new
Plugin.Media.Abstractions.StoreCameraMediaOptions
                    {
                        Directory = "Sample",
                        Name = "test.jpg",
                        SaveToAlbum = saveToGallery.IsToggled
                    });

                    if (file == null)
                        return;

                    await DisplayAlert("File Location", (saveToGallery.IsToggled ?
file.AlbumPath : file.Path), "OK");

                    image.Source = ImageSource.FromStream(() =>
                    {
                        var stream = file.GetStream();
                        file.Dispose();
                        return stream;
                    });
                }
                catch //(Exception ex)
                {
                   // Xamarin.Insights.Report(ex);
                   // await DisplayAlert("Uh oh", "Something went wrong, but don't worry we
captured it in Xamarin Insights! Thanks.", "OK");
                }
            };
```

```
            pickPhoto.Clicked += async (sender, args) =>
            {
                if (!CrossMedia.Current.IsPickPhotoSupported)
                {
                    await DisplayAlert("Photos Not Supported", ":( Permission not granted to
photos.", "OK");
                    return;
                }
                try
                {
                    Stream stream = null;
                    var file = await CrossMedia.Current.PickPhotoAsync().ConfigureAwait(true);


                    if (file == null)
                        return;

                    stream = file.GetStream();
                    file.Dispose();

                    image.Source = ImageSource.FromStream(() => stream);

                }
                catch //(Exception ex)
                {
                   // Xamarin.Insights.Report(ex);
                   // await DisplayAlert("Uh oh", "Something went wrong, but don't worry we
captured it in Xamarin Insights! Thanks.", "OK");
                }
            };

            takeVideo.Clicked += async (sender, args) =>
            {
                if (!CrossMedia.Current.IsCameraAvailable ||
!CrossMedia.Current.IsTakeVideoSupported)
                {
                    await DisplayAlert("No Camera", ":( No camera avaialble.", "OK");
                    return;
                }

                try
                {
                    var file = await CrossMedia.Current.TakeVideoAsync(new
Plugin.Media.Abstractions.StoreVideoOptions
                    {
                        Name = "video.mp4",
                        Directory = "DefaultVideos",
                        SaveToAlbum = saveToGallery.IsToggled
                    });

                    if (file == null)
                        return;

                    await DisplayAlert("Video Recorded", "Location: " +
(saveToGallery.IsToggled ? file.AlbumPath : file.Path), "OK");

                    file.Dispose();

                }
                catch //(Exception ex)
                {
```

```
                    // Xamarin.Insights.Report(ex);
                    // await DisplayAlert("Uh oh", "Something went wrong, but don't worry we
captured it in Xamarin Insights! Thanks.", "OK");
                }
            };

            pickVideo.Clicked += async (sender, args) =>
            {
                if (!CrossMedia.Current.IsPickVideoSupported)
                {
                    await DisplayAlert("Videos Not Supported", ":( Permission not granted to
videos.", "OK");
                    return;
                }
                try
                {
                    var file = await CrossMedia.Current.PickVideoAsync();

                    if (file == null)
                        return;

                    await DisplayAlert("Video Selected", "Location: " + file.Path, "OK");
                    file.Dispose();

                }
                catch //(Exception ex)
                {
                    //Xamarin.Insights.Report(ex);
                    //await DisplayAlert("Uh oh", "Something went wrong, but don't worry we
captured it in Xamarin Insights! Thanks.", "OK");
                }
            };
        }
    }
}
```

Messaging Plugin

Messaging plugin for Xamarin and Windows to make a phone call, send a sms or send an e-mail using the default messaging applications on the different mobile platforms.

Available Nuget : [https://www.nuget.org/packages/Xam.Plugins.Messaging/][1]

XAML

```
 <StackLayout Spacing="10" Padding="10">
      <Entry Placeholder="Phone Number" x:Name="phone"/>
      <Button x:Name="buttonSms" Text="Send SMS"/>
      <Button x:Name="buttonCall" Text="Call Phone Number"/>
      <Entry Placeholder="E-mail Address" x:Name="email"/>
      <Button x:Name="buttonEmail" Text="Send E-mail"/>
      <Label Text=""/>

    </StackLayout>
```

Code

```
namespace PluginDemo
{
    public partial class MessagingPage : ContentPage
    {
        public MessagingPage()
        {
            InitializeComponent();
            buttonCall.Clicked += async (sender, e) =>
            {
                try
                {
                    // Make Phone Call
                    var phoneCallTask = MessagingPlugin.PhoneDialer;
                    if (phoneCallTask.CanMakePhoneCall)
                        phoneCallTask.MakePhoneCall(phone.Text);
                    else
                        await DisplayAlert("Error", "This device can't place calls", "OK");
                }
                catch
                {
                   // await DisplayAlert("Error", "Unable to perform action", "OK");
                }
            };

            buttonSms.Clicked += async (sender, e) =>
            {
                try
                {

                    var smsTask = MessagingPlugin.SmsMessenger;
                    if (smsTask.CanSendSms)
                        smsTask.SendSms(phone.Text, "Hello World");
                    else
                        await DisplayAlert("Error", "This device can't send sms", "OK");
                }
                catch
                {
                   // await DisplayAlert("Error", "Unable to perform action", "OK");
                }
            };

            buttonEmail.Clicked += async (sender, e) =>
            {
                try
                {
                    var emailTask = MessagingPlugin.EmailMessenger;
                    if (emailTask.CanSendEmail)
                        emailTask.SendEmail(email.Text, "Hello there!", "This was sent from
the Xamrain Messaging Plugin from shared code!");
                    else
                        await DisplayAlert("Error", "This device can't send emails", "OK");
                }
                catch
                {
//await DisplayAlert("Error", "Unable to perform action", "OK");
                }
            };
        }
    }
}
```

Permissions Plugin

Check to see if your users have granted or denied permissions for common permission groups on iOS and Android.

Additionally, you can request permissions with a simple cross-platform async/awaitified API.

Available Nuget : https://www.nuget.org/packages/Plugin.Permissions enter link description here
XAML

XAML

```
 <StackLayout Padding="30" Spacing="10">
      <Button Text="Get Location" Clicked="Button_OnClicked"></Button>
      <Label x:Name="LabelGeolocation"></Label>
      <Button Text="Calendar" StyleId="Calendar"
Clicked="ButtonPermission_OnClicked"></Button>
      <Button Text="Camera" StyleId="Camera" Clicked="ButtonPermission_OnClicked"></Button>
      <Button Text="Contacts" StyleId="Contacts"
Clicked="ButtonPermission_OnClicked"></Button>
      <Button Text="Microphone" StyleId="Microphone"
Clicked="ButtonPermission_OnClicked"></Button>
      <Button Text="Phone" StyleId="Phone" Clicked="ButtonPermission_OnClicked"></Button>
      <Button Text="Photos" StyleId="Photos" Clicked="ButtonPermission_OnClicked"></Button>
      <Button Text="Reminders" StyleId="Reminders"
Clicked="ButtonPermission_OnClicked"></Button>
      <Button Text="Sensors" StyleId="Sensors" Clicked="ButtonPermission_OnClicked"></Button>
      <Button Text="Sms" StyleId="Sms" Clicked="ButtonPermission_OnClicked"></Button>
      <Button Text="Storage" StyleId="Storage" Clicked="ButtonPermission_OnClicked"></Button>
      <Label Text=""/>

    </StackLayout>
```

Code

```
bool busy;
        async void ButtonPermission_OnClicked(object sender, EventArgs e)
        {
            if (busy)
                return;

            busy = true;
            ((Button)sender).IsEnabled = false;

            var status = PermissionStatus.Unknown;
            switch (((Button)sender).StyleId)
            {
                case "Calendar":
                    status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Calendar);
                    break;
                case "Camera":
                    status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Camera);
                    break;
                case "Contacts":
                    status = await
```

```
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Contacts);
                    break;
                case "Microphone":
                    status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Microphone);
                    break;
                case "Phone":
                    status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Phone);
                    break;
                case "Photos":
                    status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Photos);
                    break;
                case "Reminders":
                    status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Reminders);
                    break;
                case "Sensors":
                    status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Sensors);
                    break;
                case "Sms":
                    status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Sms);
                    break;
                case "Storage":
                    status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Storage);
                    break;
            }

            await DisplayAlert("Results", status.ToString(), "OK");

            if (status != PermissionStatus.Granted)
            {
                switch (((Button)sender).StyleId)
                {
                    case "Calendar":
                        status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Calendar))[Permission.Calendar];
                        break;
                    case "Camera":
                        status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Camera))[Permission.Camera];
                        break;
                    case "Contacts":
                        status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Contacts))[Permission.Contacts];
                        break;
                    case "Microphone":
                        status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Microphone))[Permission.Microphone];

                        break;
                    case "Phone":
                        status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Phone))[Permission.Phone];
                        break;
                    case "Photos":
                        status = (await
```

```
CrossPermissions.Current.RequestPermissionsAsync(Permission.Photos))[Permission.Photos];
                        break;
                    case "Reminders":
                        status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Reminders))[Permission.Reminders];
                        break;
                    case "Sensors":
                        status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Sensors))[Permission.Sensors];
                        break;
                    case "Sms":
                        status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Sms))[Permission.Sms];
                        break;
                    case "Storage":
                        status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Storage))[Permission.Storage];
                        break;
                }

                await DisplayAlert("Results", status.ToString(), "OK");

            }

            busy = false;
            ((Button)sender).IsEnabled = true;
        }

        async void Button_OnClicked(object sender, EventArgs e)
        {
            if (busy)
                return;

            busy = true;
            ((Button)sender).IsEnabled = false;

            try
            {
                var status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Location);
                if (status != PermissionStatus.Granted)
                {
                    if (await
CrossPermissions.Current.ShouldShowRequestPermissionRationaleAsync(Permission.Location))
                    {
                        await DisplayAlert("Need location", "Gunna need that location", "OK");
                    }

                    var results = await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Location);
                    status = results[Permission.Location];
                }

                if (status == PermissionStatus.Granted)
                {
                    var results = await CrossGeolocator.Current.GetPositionAsync(10000);
                    LabelGeolocation.Text = "Lat: " + results.Latitude + " Long: " +
results.Longitude;
                }
                else if (status != PermissionStatus.Unknown)
                {
```

```
                    await DisplayAlert("Location Denied", "Can not continue, try again.",
"OK");
                }
            }
            catch (Exception ex)
            {

                LabelGeolocation.Text = "Error: " + ex;
            }

            ((Button)sender).IsEnabled = true;
            busy = false;
        }
```

Read Xamarin Plugin online: https://riptutorial.com/xamarin-forms/topic/7017/xamarin-plugin

Chapter 40: Xamarin Relative Layout

Remarks

The usage of *ForceLayout* in this case

Label's and button's size change according to the text inside of them. Therefore when the children are added to layout, their size remains 0 in both width and height. For example:

```
relativeLayout.Children.Add(label,
    Constraint.RelativeToParent(parent => label.Width));
```

Above expression will return 0 because width is 0 at the moment. In order to work around this, we need to listen for *SizeChanged* event and when the size changes we should force the layout in order to redraw it.

```
label.SizeChanged += (s, e) => relativeLayout.ForceLayout();
```

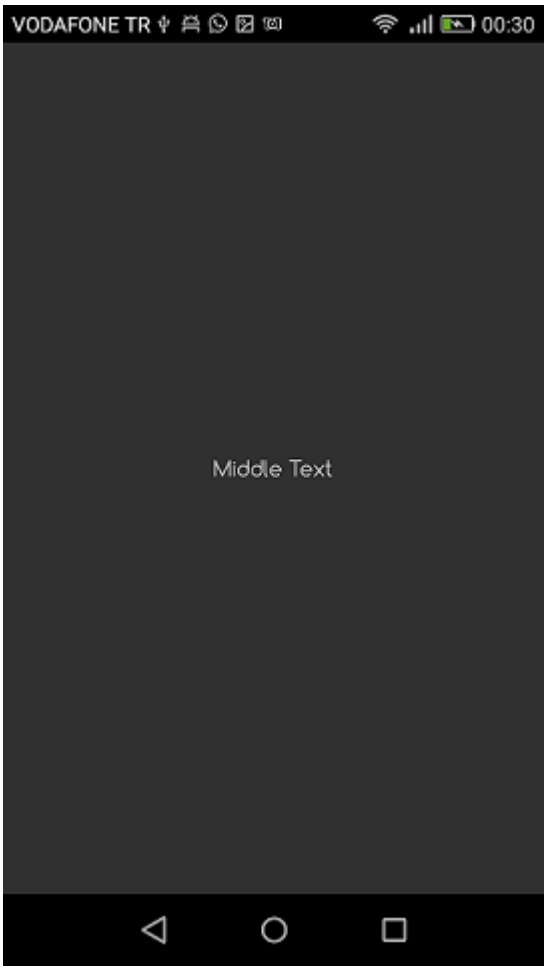
For a view like *BoxView* this is unnecessary. Because we can define their sizes on instantiation. The other things is, in both cases we can define their width and height as a constraint when we are adding them to the layout. For example:

```
relativeLayout.Children.Add(label,
    Constraint.Constant(0),
    Constraint.Constant(0),
    //Width constraint
    Constraint.Constant(30),
    //Height constraint
    Constraint.Constant(40));
```

This will add the label to the point 0, 0. The label's width and height will be 30 and 40. However, if the text is too long, some of it might not show. If your label has or might have high height, you can use *LineBreakMode* property of label. Which can wrap the text. There are a lot of options in *LineBreakMode enum*.

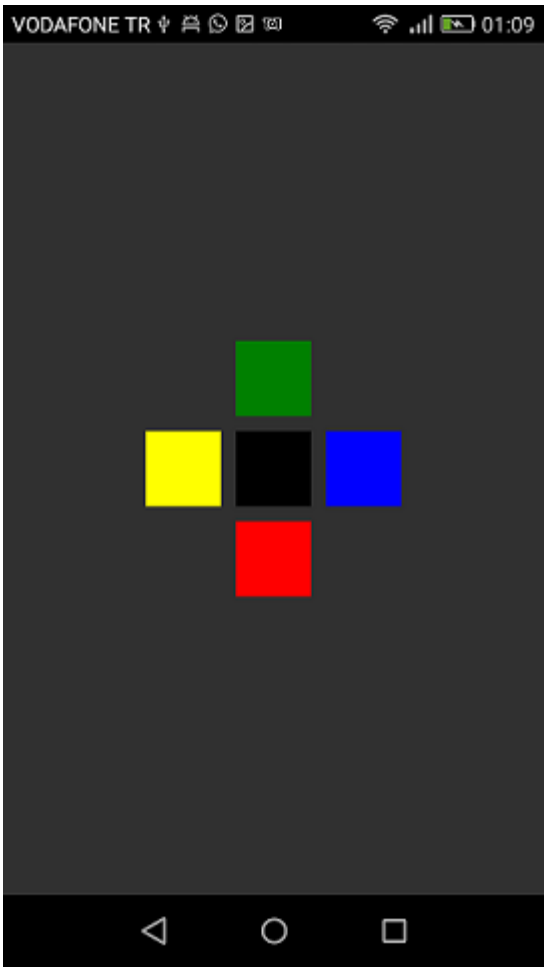
Examples

Page with an simple label on the middle




```
public class MyPage : ContentPage
{
    RelativeLayout _layout;
    Label MiddleText;

    public MyPage()
    {
        _layout = new RelativeLayout();

        MiddleText = new Label
        {
            Text = "Middle Text"
        };

        MiddleText.SizeChanged += (s, e) =>
        {
            //We will force the layout so it will know the actual width and height of the
label
            //Otherwise width and height of the label remains 0 as far as layout knows
            _layout.ForceLayout();
        };

        _layout.Children.Add(MiddleText
            Constraint.RelativeToParent(parent => parent.Width / 2 - MiddleText.Width / 2),
            Constraint.RelativeToParent(parent => parent.Height / 2 - MiddleText.Height / 2));

        Content = _layout;
    }
}
```

Box after box




```
public class MyPage : ContentPage
{
    RelativeLayout _layout;

    BoxView centerBox;
    BoxView rightBox;
    BoxView leftBox;
    BoxView topBox;
    BoxView bottomBox;

    const int spacing = 10;
    const int boxSize = 50;

    public MyPage()
    {
        _layout = new RelativeLayout();

        centerBox = new BoxView
        {
            BackgroundColor = Color.Black
        };

        rightBox = new BoxView
        {
            BackgroundColor = Color.Blue,
            //You can both set width and hight here
            //Or when adding the control to the layout
```

```
            WidthRequest = boxSize,
            HeightRequest = boxSize
        };

        leftBox = new BoxView
        {
            BackgroundColor = Color.Yellow,
            WidthRequest = boxSize,
            HeightRequest = boxSize
        };

        topBox = new BoxView
        {
            BackgroundColor = Color.Green,
            WidthRequest = boxSize,
            HeightRequest = boxSize
        };

        bottomBox = new BoxView
        {
            BackgroundColor = Color.Red,
            WidthRequest = boxSize,
            HeightRequest = boxSize
        };

        //First adding center box since other boxes will be relative to center box
        _layout.Children.Add(centerBox,
            //Constraint for X, centering it horizontally
            //We give the expression as a paramater, parent is our layout in this case
            Constraint.RelativeToParent(parent => parent.Width / 2 - boxSize / 2),
            //Constraint for Y, centering it vertically
            Constraint.RelativeToParent(parent => parent.Height / 2 - boxSize / 2),
            //Constraint for Width
            Constraint.Constant(boxSize),
            //Constraint for Height
            Constraint.Constant(boxSize));

        _layout.Children.Add(leftBox,
            //The x constraint will relate on some level to centerBox
            //Which is the first parameter in this case
            //We both need to have parent and centerBox, which will be called sibling,
            //in our expression paramters
            //This expression will be our second paramater
            Constraint.RelativeToView(centerBox, (parent, sibling) => sibling.X - spacing -
boxSize),
            //Since we only need to move it left,
            //it's Y constraint will be centerBox' position at Y axis
            Constraint.RelativeToView(centerBox, (parent, sibling) => sibling.Y)
            //No need to define the size constraints
            //Since we initialize them during instantiation
        );

        _layout.Children.Add(rightBox,
            //The only difference hear is adding spacing and boxSize instead of substracting
them
            Constraint.RelativeToView(centerBox, (parent, sibling) => sibling.X + spacing +
boxSize),
            Constraint.RelativeToView(centerBox, (parent, sibling) => sibling.Y)
        );

        _layout.Children.Add(topBox,
```

```
            //Since we are going to move it vertically this thime
            //We need to do the math on Y Constraint
            //In this case, X constraint will be centerBox' position at X axis
            Constraint.RelativeToView(centerBox, (parent, sibling) => sibling.X),
            //We will do the math on Y axis this time
            Constraint.RelativeToView(centerBox, (parent, sibling) => sibling.Y  - spacing -
boxSize)
        );

        _layout.Children.Add(bottomBox,
            Constraint.RelativeToView(centerBox, (parent, sibling) => sibling.X),
            Constraint.RelativeToView(centerBox, (parent, sibling) => sibling.Y + spacing +
boxSize)
        );

        Content = _layout;
    }
}
```

Read Xamarin Relative Layout online: https://riptutorial.com/xamarin-forms/topic/6583/xamarin-relative-layout

Chapter 41: Xamarin.Forms Cells

Examples

EntryCell

An EntryCell is a Cell that combines the capabilities of a Label and an Entry. The EntryCell can be useful in scenarios when building some functionality within your application to gather data from the user. They can easily be placed into a TableView and be treated as a simple form.

XAML

```
<EntryCell Label="Type Something"
Placeholder="Here"/>
```

Code

```
var entryCell = new EntryCell {
Label = "Type Something",
Placeholder = "Here"
};
```

SwitchCell

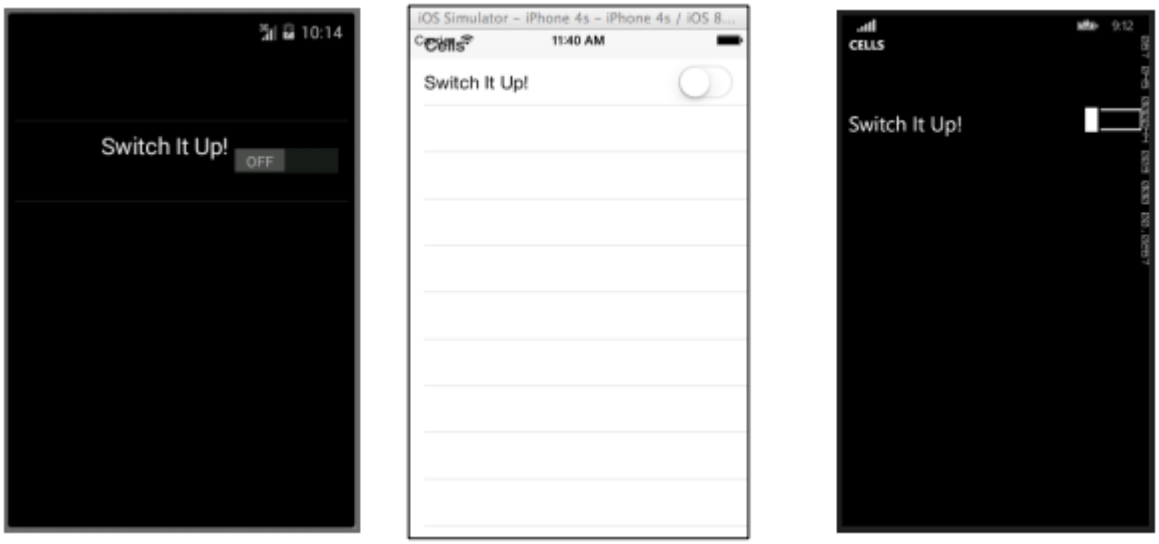
A SwitchCell is a Cell that combines the capabilities of a Label and an on-off switch. A SwitchCell can be useful for turning on and off functionality, or even user preferences or configuration options.

XAML

```
<SwitchCell Text="Switch It Up!" />
```

Code

```
var switchCell = new SwitchCell {
Text = "Switch It Up!"
};
```



TextCell

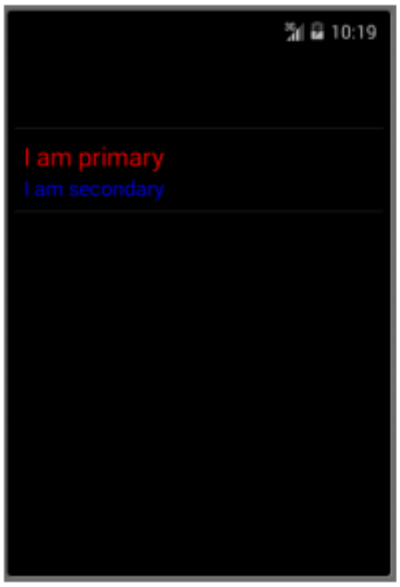
A TextCell is a Cell that has two separate text areas for displaying data. A TextCell is typically used for information purposes in both TableView and ListView controls. The two text areas are aligned vertically to maximize the space within the Cell. This type of Cell is also commonly used to display hierarchical data, so when the user taps this cell, it will navigate to another page.

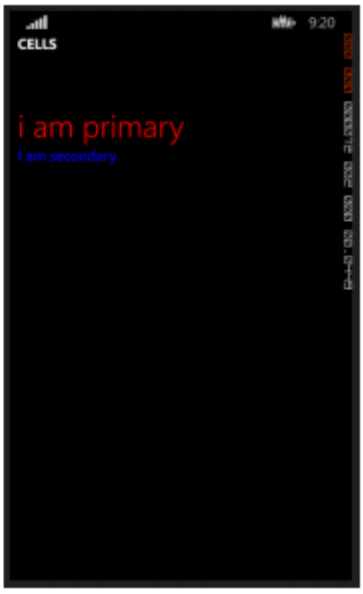
XAML

```
<TextCell Text="I am primary"
TextColor="Red"
Detail="I am secondary"
DetailColor="Blue"/>
```

Code

```
var textCell = new TextCell {
Text = "I am primary",
TextColor = Color.Red,
Detail = "I am secondary",
DetailColor = Color.Blue
};
```





ImageCell

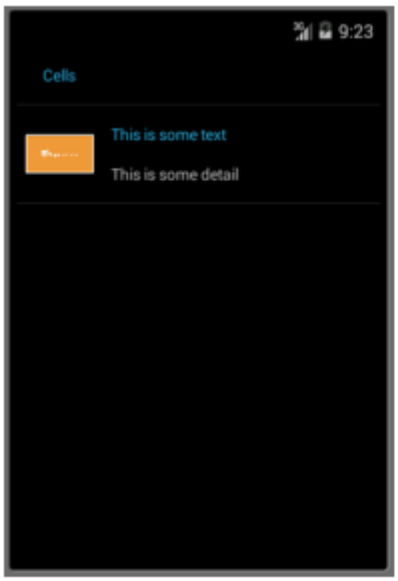
An ImageCell is exactly what it sounds like. It is a simple Cell that contains only an Image. This control functions very similarly to a normal Image control, but with far fewer bells and whistles.

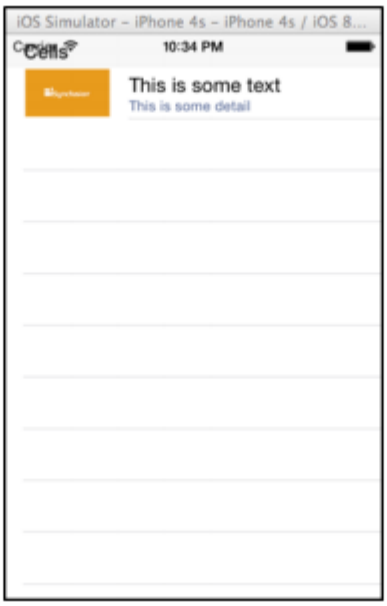
XAML

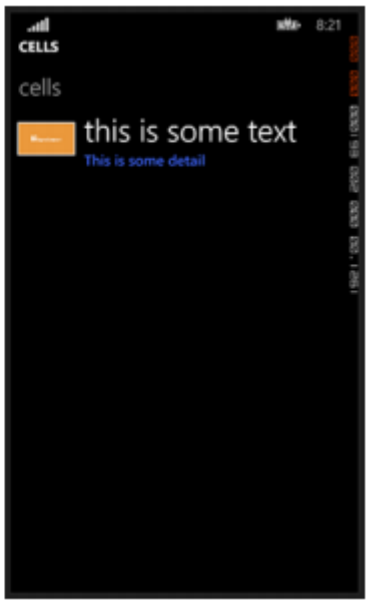
```
<ImageCell ImageSource="http://d2g29cya9iq7ip.cloudfront.net/content/imag
es/company/aboutus-video-bg.png?v=25072014072745")),
 Text="This is some text"
 Detail="This is some detail" />
```

Code

```
var imageCell = new ImageCell {
ImageSource = ImageSource.FromUri(new Uri("http://d2g29cya9iq7ip.clou
109
dfront.net/content/images/company/aboutus-videobg.png?v=25072014072745")),
 Text = "This is some text",
 Detail = "This is some detail"
};
```







ViewCell

You can consider a ViewCell a blank slate. It is your personal canvas to create a Cell that looks exactly the way you want it. You can even compose it of instances of multiple other View objects put together with Layout controls. You are only limited by your imagination. And maybe screen size.

XAML

```
<ViewCell>
<ViewCell.View>
<StackLayout>
<Button Text="My Button"/>

<Label Text="My Label"/>
<Entry Text="And some other stuff"/>
</StackLayout>
</ViewCell.View>
</ViewCell>
```

Code

```
var button = new Button { Text = "My Button" };
var label = new Label { Text = "My Label" };
var entry = new Entry { Text ="And some other stuff" };
var viewCell = new ViewCell {
View = new StackLayout {
Children = { button, label, entry }
}
};
```

Read Xamarin.Forms Cells online: https://riptutorial.com/xamarin-forms/topic/7370/xamarin-forms-cells

Chapter 42: Xamarin.Forms Page

Examples

TabbedPage

A TabbedPage is similar to a NavigationPage in that it allows for and manages simple navigation between several child Page objects. The difference is that generally speaking, each platform displays some sort of bar at the top or bottom of the screen that displays most, if not all, of the available child Page objects. In Xamarin.Forms applications, a TabbedPage is generally useful when you have a small predefined number of pages that users can navigate between, such as a menu or a simple wizard that can be positioned at the top or bottom of the screen.

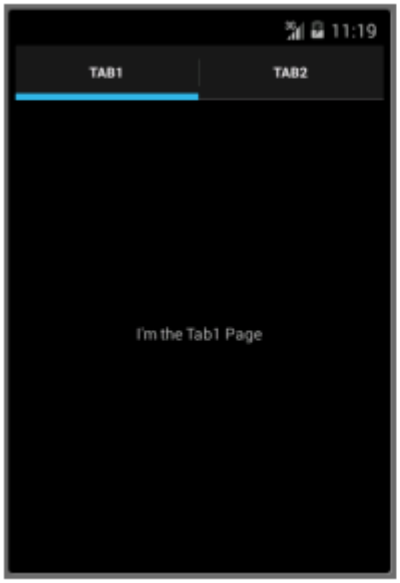
XAML

```
<?xml version="1.0" encoding="utf-8" ?>
<TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            x:Class="XamlBasics.SampleXaml">
  <TabbedPage.Children>
    <ContentPage Title="Tab1">
      <Label Text="I'm the Tab1 Page"
             HorizontalOptions="Center"
             VerticalOptions="Center"/>
    </ContentPage>
    <ContentPage Title="Tab2">
    <Label Text="I'm the Tab2 Page"
           HorizontalOptions="Center"
           VerticalOptions="Center"/>
    </ContentPage>
  </TabbedPage.Children>
</TabbedPage>
```

Code

```
var page1 = new ContentPage {
Title = "Tab1",
Content = new Label {
Text = "I'm the Tab1 Page",
HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center
}
};
var page2 = new ContentPage {
Title = "Tab2",
Content = new Label {
Text = "I'm the Tab2 Page",
HorizontalOptions = LayoutOptions.Center,
66
VerticalOptions = LayoutOptions.Center
}
```

```
};
var tabbedPage = new TabbedPage {
Children = { page1, page2 }
};
```



ContentPage

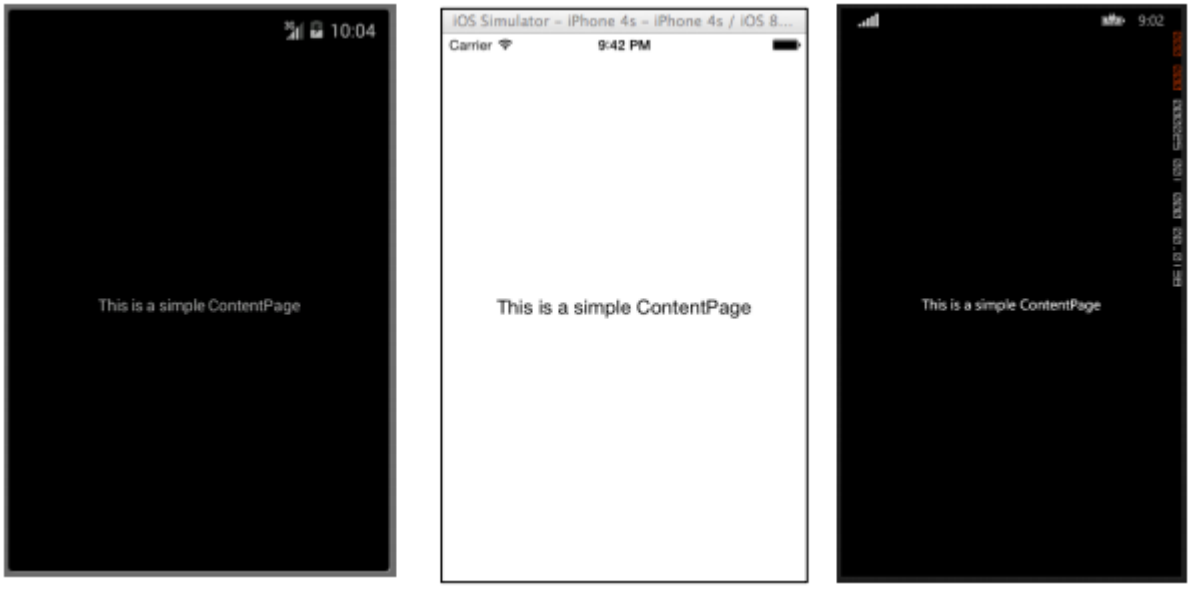
ContentPage: Displays a single View.

XAML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="XamlBasics.SampleXaml">
<Label Text="This is a simple ContentPage"
HorizontalOptions="Center"
VerticalOptions="Center" />
</ContentPage>
```

Code

```
var label = new Label {
Text = "This is a simple ContentPage",
HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center
};
var contentPage = new ContentPage {
Content = label
};
```



MasterDetailPage

MasterDetailPage: Manages two separate Pages (panes) of information.

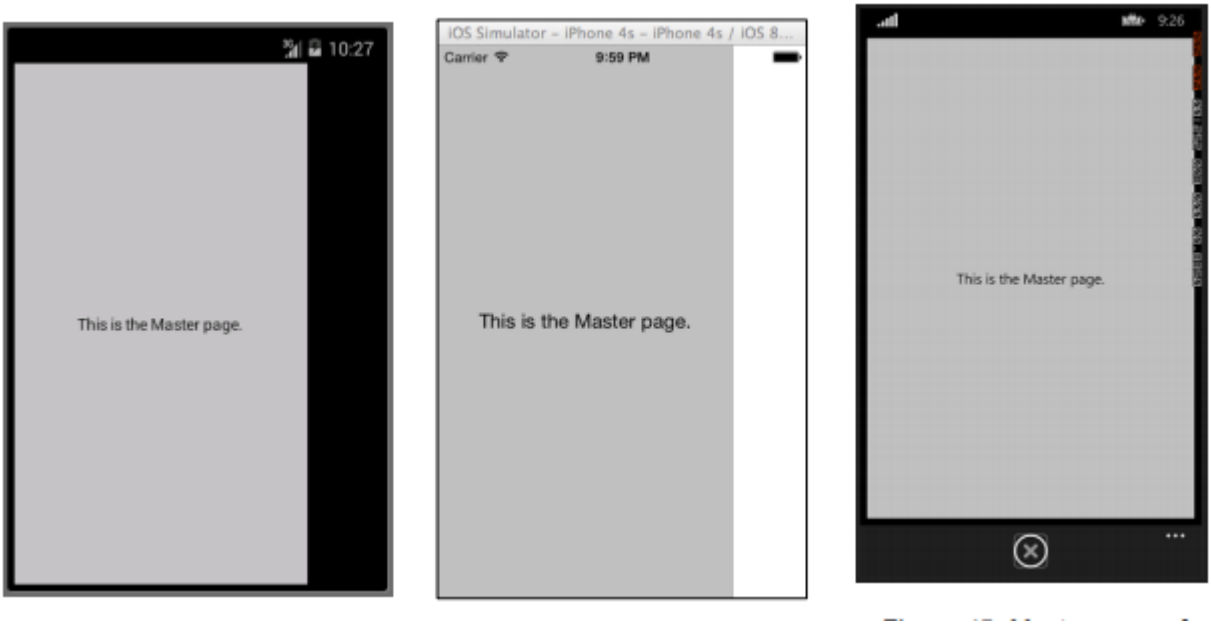
XAML

```
<?xml version="1.0" encoding="utf-8" ?>
<MasterDetailPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="XamlBasics.SampleXaml">
<MasterDetailPage.Master>
<ContentPage Title = "Master" BackgroundColor = "Silver">
<Label Text="This is the Master page."
TextColor = "Black"
HorizontalOptions="Center"
VerticalOptions="Center" />
</ContentPage>
</MasterDetailPage.Master>
<MasterDetailPage.Detail>
<ContentPage>
<Label Text="This is the Detail page."
HorizontalOptions="Center"
VerticalOptions="Center" />
</ContentPage>
</MasterDetailPage.Detail>
</MasterDetailPage>
```

Code

```
var masterDetailPage = new MasterDetailPage {
Master = new ContentPage {
Content = new Label {
Title = "Master",
BackgroundColor = Color.Silver,

TextColor = Color.Black,
Text = "This is the Master page.",
HorizontalOptions = LayoutOptions.Center,
```

```
VerticalOptions = LayoutOptions.Center
}
},
Detail = new ContentPage {
Content = new Label {
Title = "Detail",
Text = "This is the Detail page.",
HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center
}
}
};
```



Read Xamarin.Forms Page online: https://riptutorial.com/xamarin-forms/topic/7018/xamarin-forms-page

Chapter 43: Xamarin.Forms Views

Examples

Button

The **Button** is probably the most common control not only in mobile applications, but in any applications that have a UI. The concept of a button has too many purposes to list here. Generally speaking though, you will use a button to allow users to initiate some sort of action or operation within your application. This operation could include anything from basic navigation within your app, to submitting data to a web service somewhere on the Internet.

XAML

```
<Button
    x:Name="MyButton"
    Text="Click Me!"
    TextColor="Red"
    BorderColor="Blue"
    VerticalOptions="Center"
    HorizontalOptions="Center"
    Clicked="Button_Clicked"/>
```

XAML Code-Behind

```
public void Button_Clicked( object sender, EventArgs args )
{
    MyButton.Text = "I've been clicked!";
}
```

Code

```
var button = new Button( )
{
    Text = "Hello, Forms !",
    VerticalOptions = LayoutOptions.CenterAndExpand,
    HorizontalOptions = LayoutOptions.CenterAndExpand,
    TextColor = Color.Red,
    BorderColor = Color.Blue,
};

button.Clicked += ( sender, args ) =>
{
    var b = (Button) sender;
    b.Text = "I've been clicked!";
};
```

DatePicker

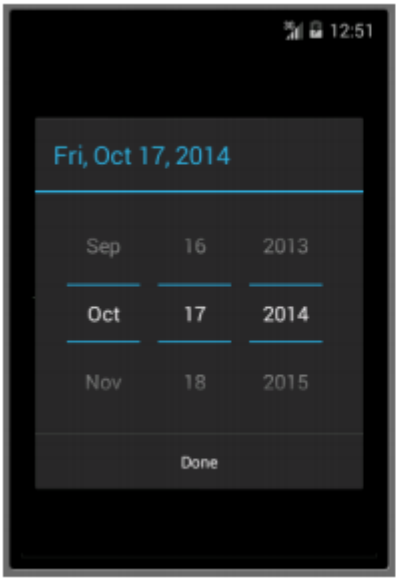
Quite often within mobile applications, there will be a reason to deal with dates. When working with dates, you will probably need some sort of user input to select a date. This could occur when working with a scheduling or calendar app. In this case, it is best to provide users with a specialized control that allows them to interactively pick a date, rather than requiring users to manually type a date. This is where the DatePicker control is really useful.

XAML

```
<DatePicker Date="09/12/2014" Format="d" />
```

Code

```
var datePicker = new DatePicker{
Date = DateTime.Now,
Format = "d"
};
```



Entry

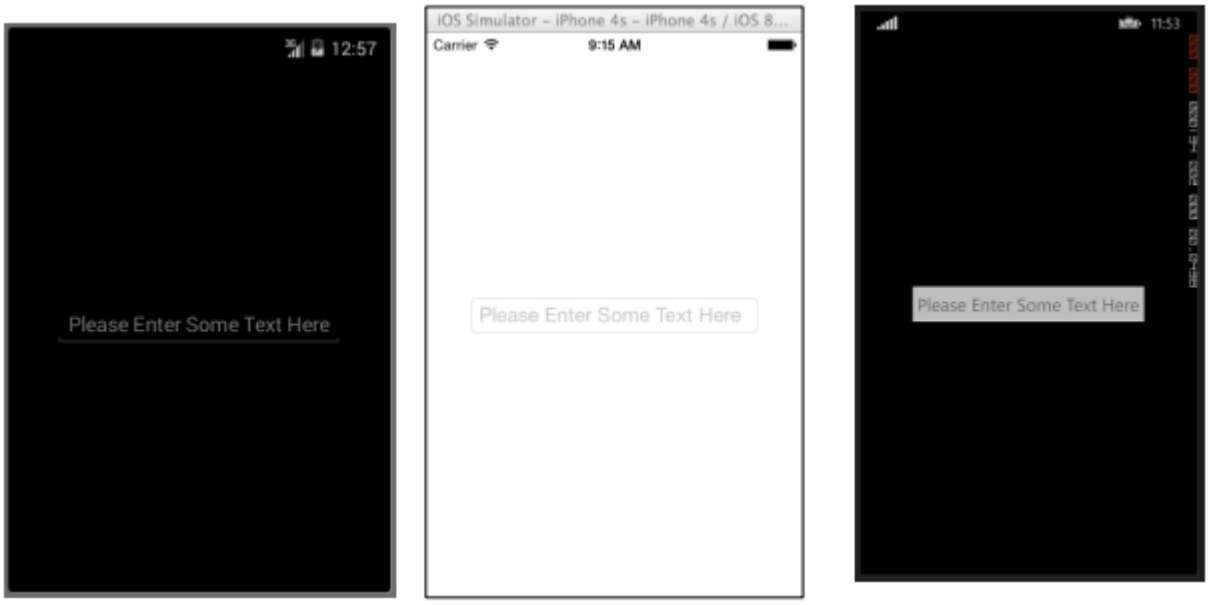
The Entry View is used to allow users to type a single line of text. This single line of text can be used for multiple purposes including entering basic notes, credentials, URLs, and more. This View is a multi-purpose View, meaning that if you need to type regular text or want to obscure a password, it is all done through this single control.

XAML

```
<Entry Placeholder="Please Enter Some Text Here"
HorizontalOptions="Center"
VerticalOptions="Center"
Keyboard="Email"/>
```

Code

```
var entry = new Entry {
Placeholder = "Please Enter Some Text Here",
HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center,
Keyboard = Keyboard.Email
};
```



Editor

The Editor is very similar to the Entry in that it allows users to enter some free-form text. The difference is that the Editor allows for multi-line input whereas the Entry is only used for single line input. The Entry also provides a few more properties than the Editor to allow further customization of the View.

XAML

```
<Editor HorizontalOptions="Fill"
VerticalOptions="Fill"
 Keyboard="Chat"/>
```

Code

```
var editor = new Editor {
HorizontalOptions = LayoutOptions.Fill,
VerticalOptions = LayoutOptions.Fill,
Keyboard = Keyboard.Chat
};
```

Image

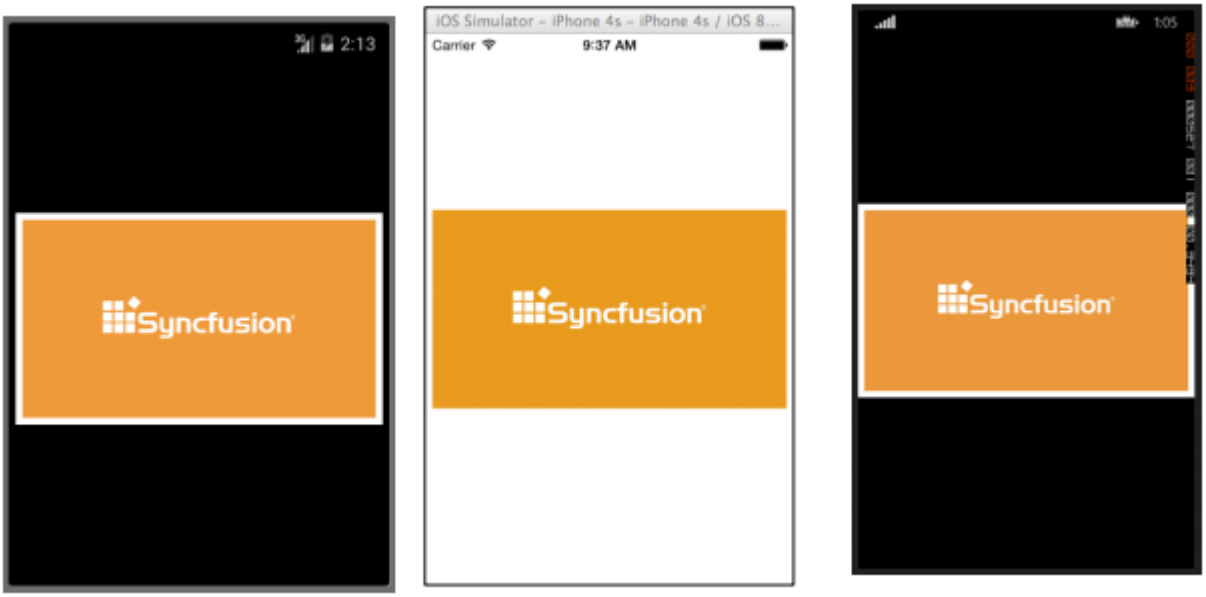
Images are very important parts of any application. They provide the opportunity to inject additional visual elements as well as branding into your application. Not to mention that images are typically more interesting to look at than text or buttons. You can use an Image as a standalone element within your application, but an Image element can also be added to other View elements such as a Button.

XAML

```
<Image Aspect="AspectFit" Source="http://d2g29cya9iq7ip.cloudfront.net/co
ntent/images/company/aboutus-video-bg.png?v=25072014072745"/>
```

Code

```
var image = new Image {
Aspect = Aspect.AspectFit,
Source = ImageSource.FromUri(new Uri("http://d2g29cya9iq7ip.cloudfron
t.net/content/images/company/aboutus-video-bg.png?v=25072014072745"))
};
```



Label

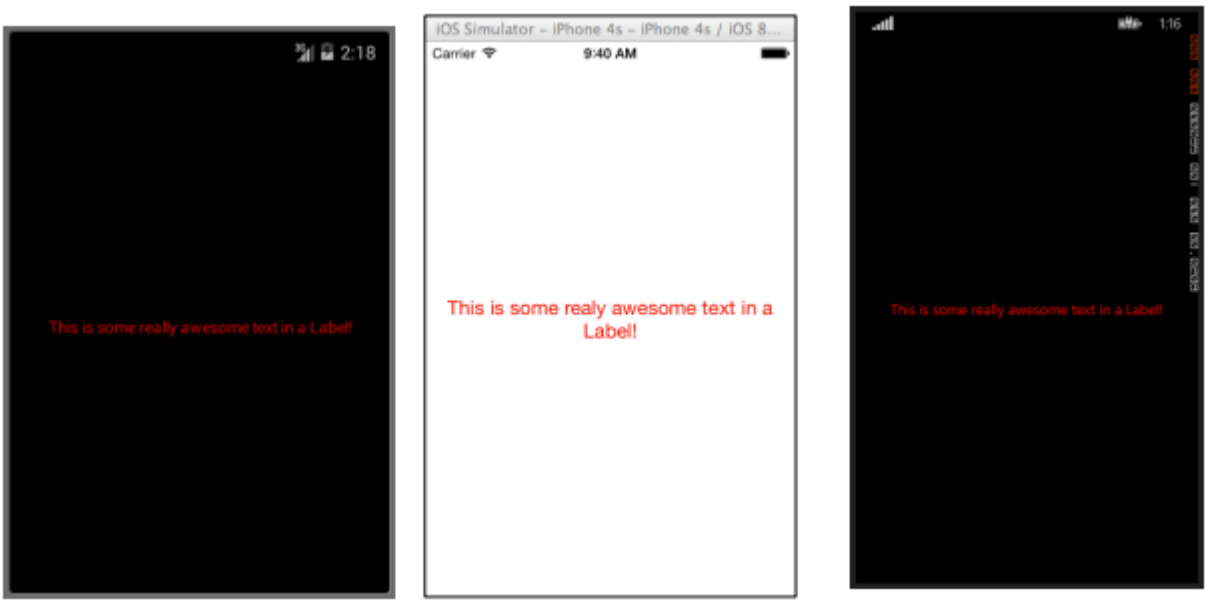
Believe it or not, the Label is one of the most crucial yet underappreciated View classes not only in Xamarin.Forms, but in UI development in general. It is seen as a rather boring line of text, but without that line of text it would be very difficult to convey certain ideas to the user. Label controls can be used to describe what the user should enter into an Editor or Entry control. They can describe a section of the UI and give it context. They can be used to show the total in a calculator app. Yes, the Label is truly the most versatile control in your tool bag that may not always spark a lot of attention, but it is the first one noticed if it isn't there.

XAML

```
<Label Text="This is some really awesome text in a Label!"
TextColor="Red"
XAlign="Center"
YAlign="Center"/>
```

Code

```
var label = new Label {
Text = "This is some really awesome text in a Label!",
TextColor = Color.Red,
XAlign = TextAlignment.Center,
YAlign = TextAlignment.Center
};
```



Read Xamarin.Forms Views online: https://riptutorial.com/xamarin-forms/topic/7369/xamarin-forms-views

Credits

S. No	Chapters	Contributors
1	Getting started with Xamarin.Forms	Akshay Kulkarni, chrisntr, Community, Demitrian, hankide, jdstaerk, Manohar, patridge, Sergey Metlov, spaceplane
2	Accessing native features with DependencyService	Gerald Versluis, hankide, hvaughan3, Sergey Metlov
3	AppSettings Reader in Xamarin.Forms	Ben Ishiyama-Levy, GvSharma
4	BDD Unit Testing in Xamarin.Forms	Ben Ishiyama-Levy
5	Caching	Sergey Metlov
6	CarouselView - Pre-release version	dpserge
7	Contact Picker - Xamarin Forms (Android and iOS)	Pucho Eric
8	Creating custom controls	hvaughan3, spaceplane, Yehor Hromadskyi
9	Custom Fonts in Styles	Roma Rudyak
10	Custom Renderers	Bonelol, hankide, Nicolas Bodin-Ripert, Nicolas Bodin-Ripert, nishantvodoo, Yehor Hromadskyi, Zverev Eugene
11	Data Binding	Andrew, Matthew, Yehor Hromadskyi
12	Dependency Services	RIYAZ
13	DependencyService	Steven Thewissen
14	Display Alert	aboozz pallikara, GvSharma, Sreeraj, Yehor Hromadskyi
15	Effects	Swaminathan Vetri
16	Exception handling	Yehor Hromadskyi

17	Generic Xamarin.Forms app lifecycle? Platform-dependant!	Zverev Eugene
18	Gestures	doerig, Gerald Versluis, Michael Rumpler
19	MessagingCenter	Gerald Versluis
20	Navigation in Xamarin.Forms	Fernando Arreguín, jimmgarr, Lucas Moura Veloso, Paul, Sergey Metlov, Taras Shevchuk, Willian D. Andrade
21	OAuth2	Eng Soon Cheah
22	Platform specific visual adjustments	Alois, GalaxiaGuy, Paul
23	Platform-specific behaviour	Ege Aydın
24	Push Notifications	Gerald Versluis, user1568891
25	SQL Database and API in Xamarin Forms.	RIYAZ
26	Triggers & Behaviours	hamalaiv, hvaughan3
27	Unit Testing	jerone, Sergey Metlov
28	Using ListViews	cvanbeek
29	Why Xamarin Forms and When to use Xamarin Forms	Daniel Krzyczkowski, mike
30	Working with local databases	Luis Beltran, Manohar
31	Working with Maps	Taras Shevchuk
32	Xamarin Forms Layouts	Eng Soon Cheah, Gerald Versluis, Lucas Moura Veloso
33	Xamarin Gesture	Joehl
34	Xamarin Plugin	Eng Soon Cheah
35	Xamarin Relative Layout	Ege Aydın

36	Xamarin.Forms Cells	Eng Soon Cheah
37	Xamarin.Forms Page	Eng Soon Cheah
38	Xamarin.Forms Views	Aaron Thompson, Eng Soon Cheah