EBook Gratis

APRENDIZAJE Xamarin.Forms

Free unaffiliated eBook created from
Stack Overflow contributors.

#xamarin.forms

Tabla de contenido

Acerca de

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

Capítulo 1: Empezando con Xamarin.Forms

Observaciones

Xamarin.Forms permite crear aplicaciones de iOS, Android y Windows con grandes cantidades de código compartido, incluido el código UI o el marcado XAML UI. Las páginas y vistas de la aplicación se asignan a controles nativos en cada plataforma, pero se pueden personalizar para proporcionar una IU específica de la plataforma o para acceder a funciones específicas de la plataforma.

Versiones

Versión	Fecha de lanzamiento
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
1.3.4	2015-02-17

Versión	Fecha de lanzamiento
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

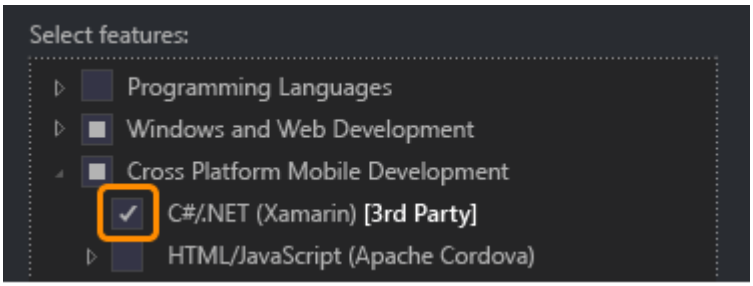
Instalación (Visual Studio)

Xamarin.Forms es un paquete de herramientas de UI respaldado de forma nativa y multiplataforma que permite a los desarrolladores crear fácilmente interfaces de usuario que pueden compartirse en Android, iOS, Windows y Windows Phone. Las interfaces de usuario se representan utilizando los controles nativos de la plataforma de destino, lo que permite que las aplicaciones Xamarin.Forms conserven el aspecto y la apariencia adecuados para cada plataforma.

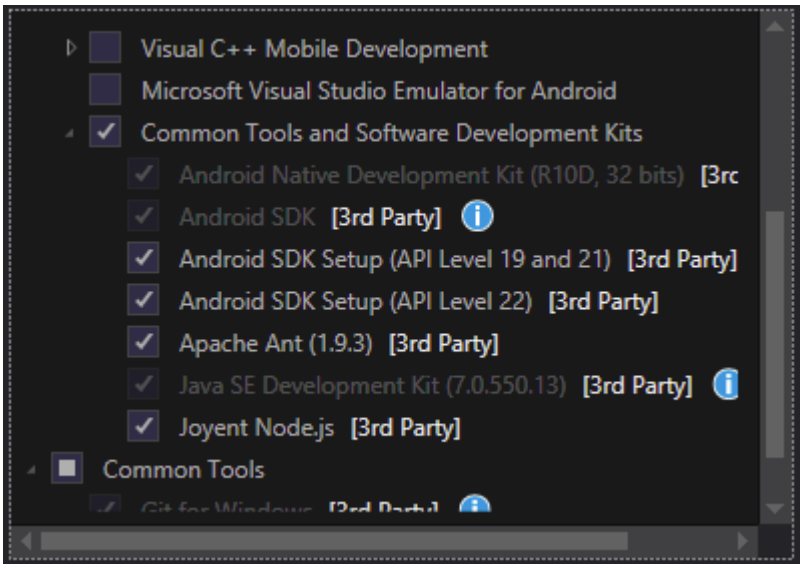
Xamarin Plugin para Visual Studio

Para comenzar con Xamarin.Forms para Visual Studio, necesita tener el complemento Xamarin. La forma más fácil de instalarlo es descargar e instalar la última versión de Visual Studio.

Si ya tiene instalada la última versión de Visual Studio, vaya a Panel de control> Programas y características, haga clic con el botón derecho en Visual Studio y haga clic en Cambiar. Cuando se abra el instalador, haga clic en Modificar y seleccione las herramientas de desarrollo móvil multiplataforma:



También puede seleccionar instalar el SDK de Android:



Desmarque si ya tiene instalado el SDK. Podrá configurar Xamarin para usar el SDK de Android existente más adelante.

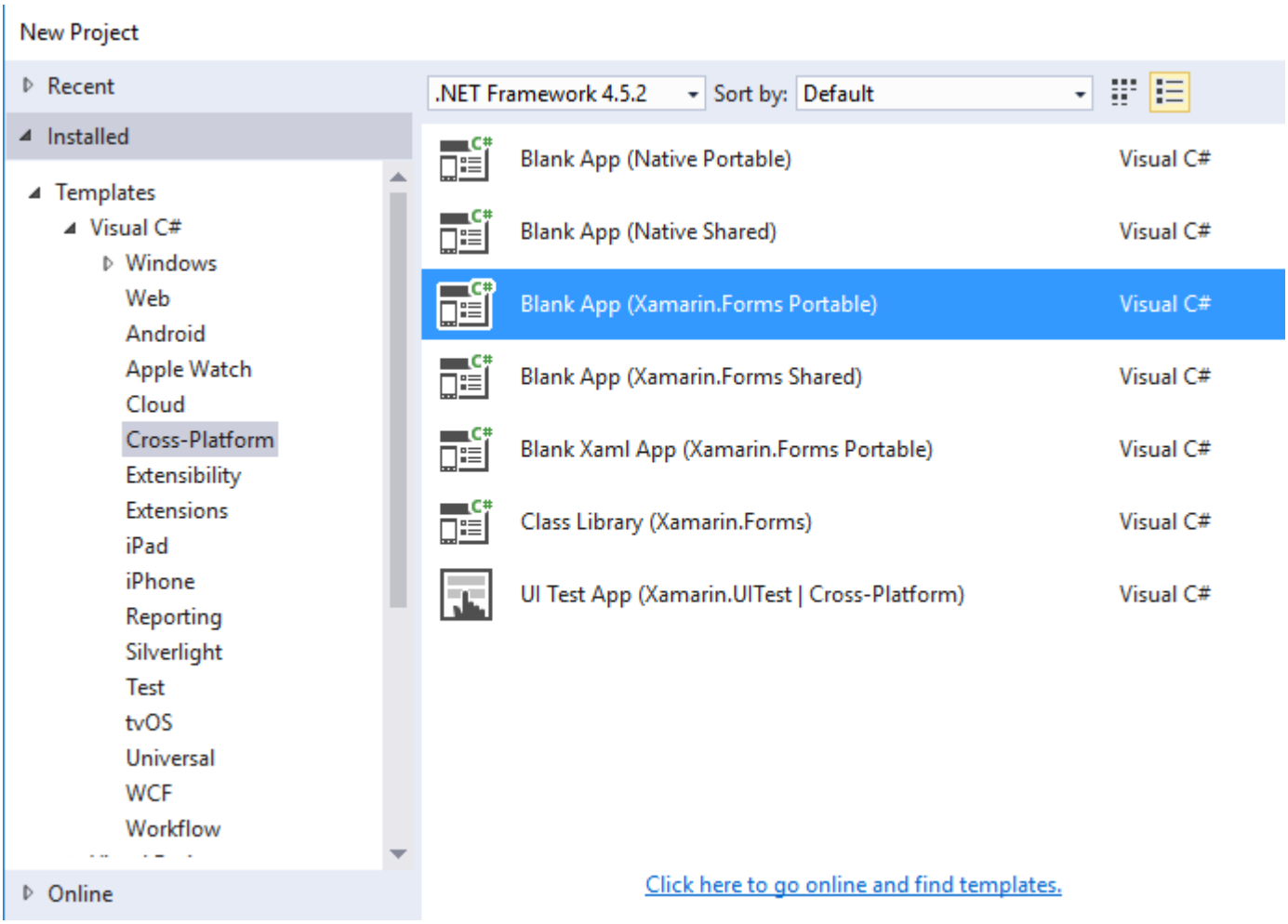
Xamarin.Formas

Xamarin.Forms es un conjunto de bibliotecas para su biblioteca de Clase portátil y ensamblajes nativos. La biblioteca Xamarin.Forms en sí está disponible como un paquete NuGet. Para agregarlo a su proyecto, simplemente use el comando regular `Install-Package` de la consola del Administrador de paquetes:

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

para todos sus ensamblajes iniciales (por ejemplo, MyProject, MyProject.Droid y MyProject.iOS).

La forma más fácil de comenzar con Xamarin.Forms es crear un proyecto vacío en Visual Studio:



Como puede ver, hay 2 opciones disponibles para crear la aplicación en blanco: portátil y compartida. Te recomiendo que comiences con Portable one porque es el más usado en el mundo real (se agregarán diferencias y más explicaciones).

Después de crear el proyecto, asegúrese de estar usando la última versión de Xamarin.Forms, ya que su plantilla inicial puede contener la anterior. Use su Consola del administrador de paquetes o la opción Administrar paquetes NuGet para actualizar a los últimos Xamarin.Forms (recuerde que es solo un paquete NuGet).

Si bien las plantillas de Visual Studio Xamarin.Forms crearán un proyecto de plataforma iOS para usted, deberá conectar Xamarin a un host de compilación Mac para poder ejecutar estos proyectos en el simulador de iOS o en dispositivos físicos.

Hola World Xamarin Forms: Visual Studio

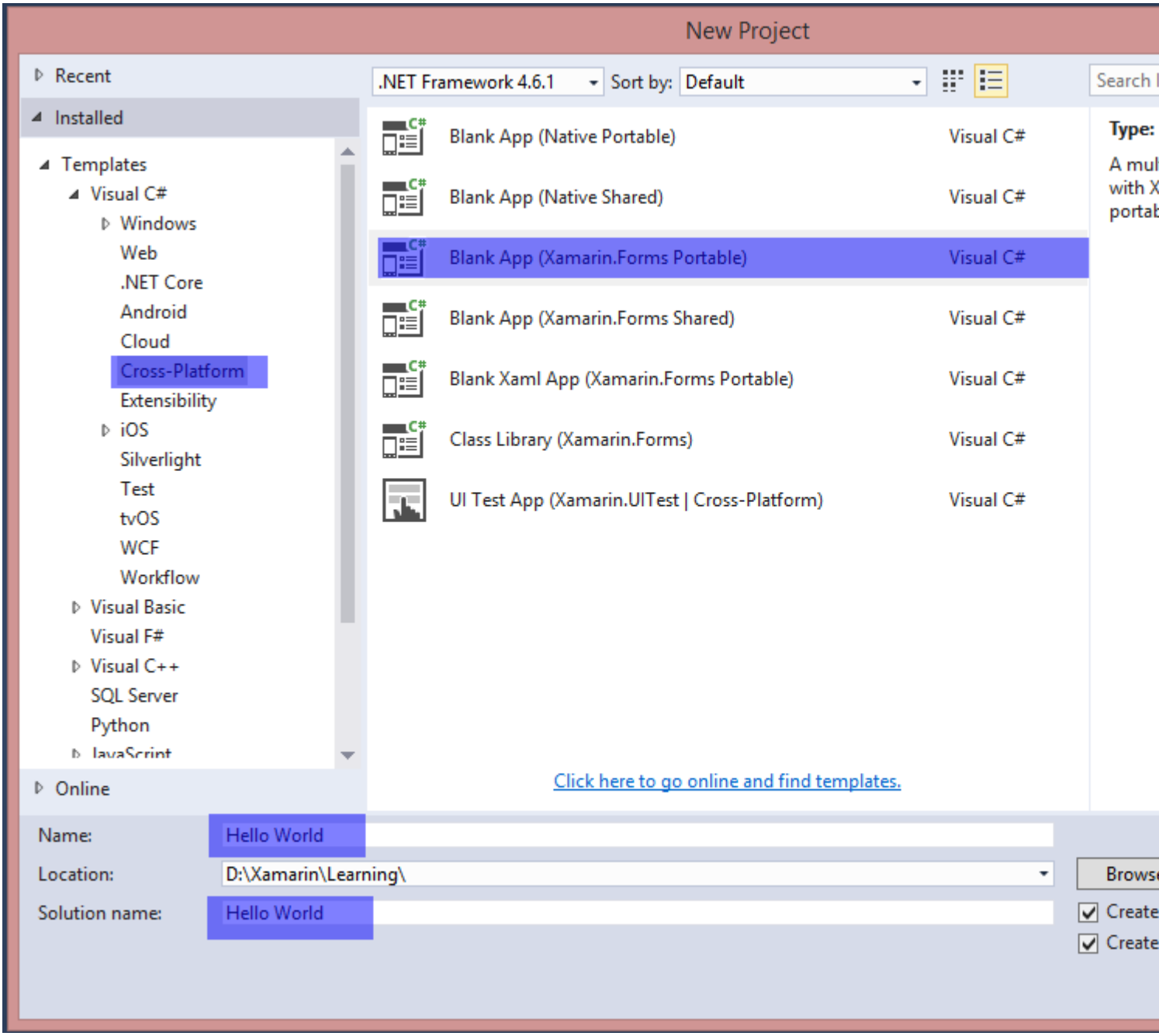
Después de instalar con éxito Xamarin como se describe en el primer ejemplo, es hora de iniciar la primera aplicación de ejemplo.

Paso 1: Creando un nuevo proyecto.

En Visual Studio, elija Nuevo -> Proyecto -> Visual C # -> Multiplataforma -> Aplicación en blanco (Xamarin.Forms Portable)

Nombre la aplicación "Hello World" y seleccione la ubicación para crear el proyecto y haga clic en Aceptar. Esto creará una solución para usted que contiene tres proyectos:

1. HelloWorld (aquí es donde se colocan su lógica y sus vistas, es decir, el proyecto portátil)
2. HelloWorld.Droid (el proyecto de Android)
3. HelloWorld.iOS (el proyecto iOS)



Paso 2: Investigando la muestra

Una vez creada la solución, una aplicación de ejemplo estará lista para ser implementada. Abra `App.cs` ubicado en la raíz del proyecto portátil e investigue el código. Como se ve a continuación, el `Content` de la muestra es un `StackLayout` que contiene una `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
        }
    }
}
```

Paso 3: Lanzar la aplicación

Ahora simplemente haga clic con el botón derecho en el proyecto que desea iniciar (`HelloWorld.Droid` o `HelloWorld.iOS`) y haga clic en `Set as StartUp Project` inicio. Luego, en la barra de herramientas de Visual Studio, haga clic en el botón `Start` (el botón triangular verde que se asemeja a un botón Reproducir) para iniciar la aplicación en el simulador / emulador seleccionado.

Lea Empezando con Xamarin.Forms en línea: https://riptutorial.com/es/xamarin-forms/topic/908/empezando-con-xamarin-forms

Capítulo 2: ¿Ciclo de vida de la aplicación Xamarin.Forms genérico? Dependiente de la plataforma!

Examples

El ciclo de vida de Xamarin.Forms no es el ciclo de vida real de la aplicación, sino una representación multiplataforma de la misma.

Veamos los métodos de ciclo de vida de las aplicaciones nativas para diferentes plataformas.

Androide.

```
//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;
```

Y ahora los métodos del ciclo de vida de la aplicación **Xamarin.Forms** :

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

Lo que se puede ver fácilmente con solo observar las listas, la perspectiva del ciclo de vida de la aplicación multiplataforma Xamarin.Forms se simplifica enormemente. Le da la pista genérica sobre el estado de su aplicación, pero en la mayoría de los casos de producción tendrá que construir alguna lógica dependiente de la plataforma.

Lea ¿Ciclo de vida de la aplicación Xamarin.Forms genérico? Dependiente de la plataforma! en línea: https://riptutorial.com/es/xamarin-forms/topic/8329/-ciclo-de-vida-de-la-aplicacion-xamarin-forms-generico--dependiente-de-la-plataforma-

Capítulo 3: ¿Por qué Xamarin se forma y cuándo usar Xamarin?

Observaciones

Puede consultar la documentación oficial de Xamarin Forms para explorar más:

https://www.xamarin.com/forms

Examples

¿Por qué Xamarin se forma y cuándo usar Xamarin?

Xamarin se está volviendo más y más popular: es difícil decidir cuándo usar Xamarin.Forms y cuándo Xamarin.Platform (por lo tanto, Xamarin.iOS y Xamarin.Android).

En primer lugar, debe saber para qué tipo de aplicaciones puede utilizar Xamarin.Forms:

1. Prototipos: para visualizar cómo se verá su aplicación en los diferentes dispositivos.
2. Aplicaciones que no requieren una funcionalidad específica de la plataforma (como las API), pero aquí, tenga en cuenta que Xamarin está trabajando para proporcionar la mayor compatibilidad posible entre plataformas.
3. Aplicaciones donde compartir código es crucial, más importante que la interfaz de usuario.
4. Aplicaciones donde los datos mostrados son más importantes que la funcionalidad avanzada

También hay muchos otros factores:

1. Quién será responsable del desarrollo de la aplicación: si su equipo está formado por desarrolladores móviles con experiencia, podrán manejar Xamarin.Forms fácilmente. Pero si tiene un desarrollador por plataforma (desarrollo nativo), los formularios pueden ser un desafío mayor.
2. Tenga en cuenta que con Xamarin.Forms todavía puede encontrar algunos problemas: la plataforma Xamarin.Forms aún se está mejorando.
3. El desarrollo rápido a veces es muy importante: para reducir los costos y el tiempo, puede decidir utilizar los Formularios.
4. Al desarrollar aplicaciones empresariales sin ninguna funcionalidad avanzada, es mejor usar Xamarin.Forms: le permite compartir el código de modo, no en el área móvil, sino en general. Algunas partes del código se pueden compartir en muchas plataformas.

No debes usar Xamarin.Forms cuando:

1. Tienes que crear una funcionalidad personalizada y acceder a las API específicas de la plataforma.
2. Tienes que crear una interfaz de usuario personalizada para la aplicación móvil
3. Cuando alguna funcionalidad no está lista para Xamarin.Forms (como algún comportamiento específico en el dispositivo móvil)
4. Su equipo está formado por desarrolladores móviles específicos de la plataforma (desarrollo móvil en Java y / o Swift / Objective C)

Lea ¿Por qué Xamarin se forma y cuándo usar Xamarin? en línea: https://riptutorial.com/es/xamarin-forms/topic/6869/-por-que-xamarin-se-forma-y-cuando-usar-xamarin-

Capítulo 4: Acceso a funciones nativas con DependencyService

Observaciones

Si no desea que su código se rompa cuando no se encuentra ninguna implementación, primero verifique que `DependencyService` tiene una implementación disponible.

Puede hacerlo mediante una simple comprobación si no es `null` .

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

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

o, si su IDE es compatible con C # 6, con el operador condicional nulo:

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

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

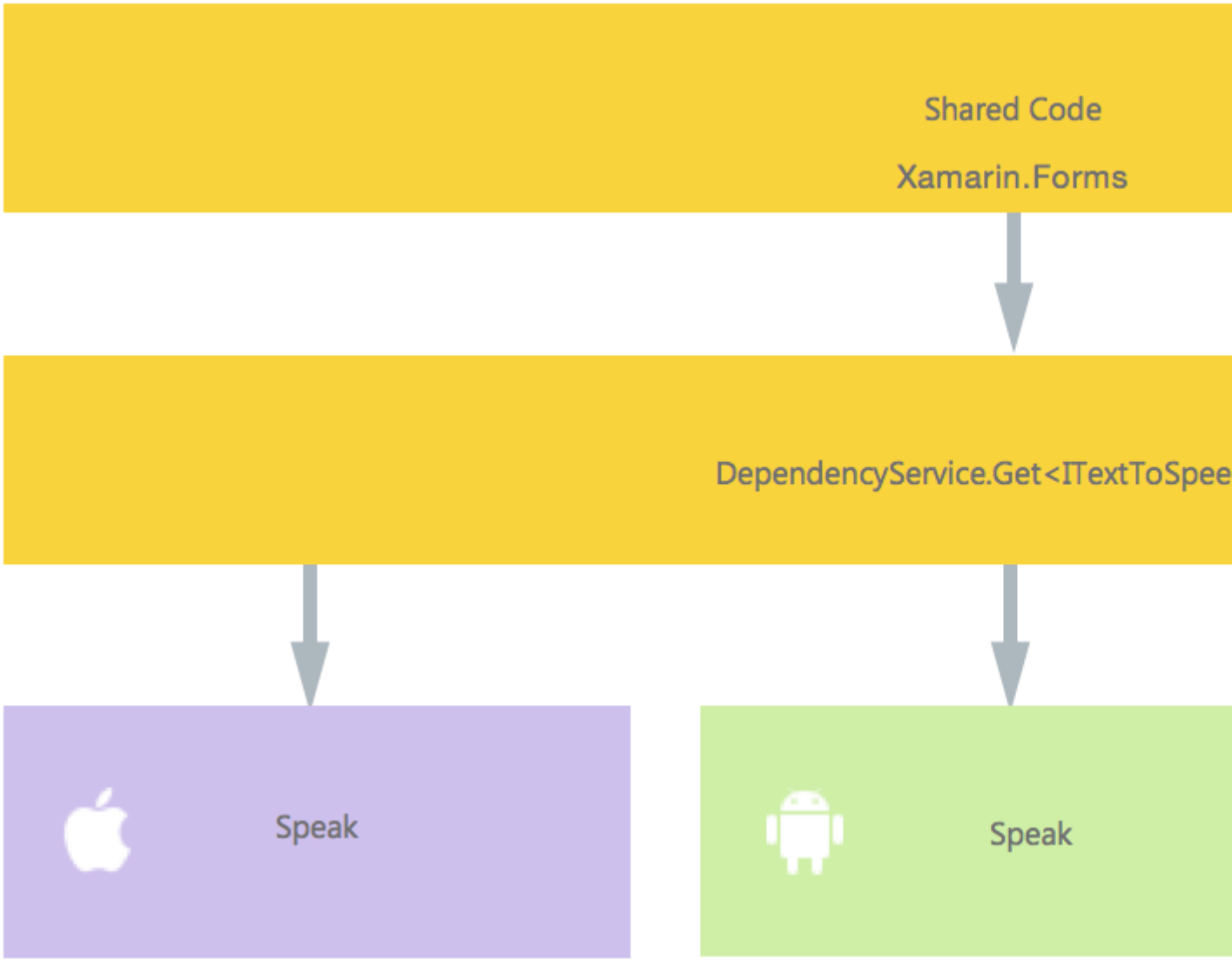
Si no hace esto y no se encuentra ninguna implementación en tiempo de ejecución, su código generará una excepción.

Examples

Implementando texto a voz

Un buen ejemplo de una característica que solicita código específico de plataforma es cuando desea implementar texto a voz (tts). Este ejemplo asume que está trabajando con código compartido en una biblioteca PCL.

Una vista general esquemática de nuestra solución se vería como la imagen de abajo.



En nuestro código compartido definimos una interfaz que está registrada con `DependencyService` . Aquí es donde haremos nuestros llamamientos. Definir una interfaz como la de abajo.

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

Ahora, en cada plataforma específica, necesitamos crear una implementación de esta interfaz. Vamos a empezar con la implementación de iOS.

Implementación de iOS

```
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);
    }
}
```

En el ejemplo de código anterior, observa que hay un código específico para iOS. Me gusta tipos como `AVSpeechSynthesizer` . Estos no funcionarían en código compartido.

Para registrar esta implementación con el Servicio de `DependencyService` Xamarin, agregue este atributo encima de la declaración de espacio de nombres.

```
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
```

Ahora, cuando realiza una llamada como esta en su código compartido, se inyecta la implementación correcta para la plataforma en la que está ejecutando su aplicación.

`DependencyService.Get<ITextToSpeech>()` . Más sobre esto más adelante.

Implementación de Android

La implementación de Android de este código se vería debajo.

```
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
}
```

Nuevamente, no olvide registrarlo en `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
```

Implementación de Windows Phone

Finalmente, para Windows Phone se puede usar este código.

```
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);
    }
}
```

Y una vez más no te olvides de registrarlo.

```
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
```

Implementando en Código Compartido

¡Ahora todo está listo para que funcione! Finalmente, en su código compartido ahora puede llamar a esta función utilizando la interfaz. En tiempo de ejecución, se inyectará la implementación que corresponda a la plataforma actual en la que se está ejecutando.

En este código, verá una página que podría estar en un proyecto de Xamarin Forms. Crea un botón que invoca el método `Speak()` usando el servicio `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;
}
```

El resultado será que cuando se ejecute la aplicación y se haga clic en el botón, se pronunciará el texto proporcionado.

Todo esto sin tener que hacer cosas difíciles como sugerencias del compilador y demás. Ahora tiene una forma uniforme de acceder a la funcionalidad específica de la plataforma a través del código independiente de la plataforma.

Obtención de los números de versión del SO de la aplicación y del dispositivo - Android e iOS - PCL

El siguiente ejemplo recopilará el número de versión del SO del dispositivo y la versión de la aplicación (que se define en las propiedades de cada proyecto) que se ingresa en **Nombre de versión** en Android y **Versión** en iOS.

Primero haz una interfaz en tu proyecto PCL:

```
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();
}
```

Ahora implementamos la interfaz en los proyectos de Android e iOS.

Androide:

```
[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; }
    }
```

```
}
```

Ahora para usar el código en un método:

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

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

Lea Acceso a funciones nativas con DependencyService en línea: https://riptutorial.com/es/xamarin-forms/topic/2409/acceso-a-funciones-nativas-con-dependencyservice

Capítulo 5: Ajustes visuales específicos de la plataforma.

Examples

Ajustes de idioma

Se pueden realizar ajustes específicos del idioma a partir del código C #, por ejemplo, para cambiar la orientación del diseño, ya sea que se muestre la vista o un teléfono o una tableta.

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

Esas funcionalidades también están disponibles directamente desde el código XAML:

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

Ajustes de plataforma

Se pueden hacer ajustes para plataformas específicas desde el código C #, por ejemplo, para cambiar el relleno de todas las plataformas específicas.

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

También está disponible un método auxiliar para declaraciones de C # acortadas:

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

Esas funcionalidades también están disponibles directamente desde el código XAML:

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

Usando estilos

Al trabajar con XAML, el uso de un `Style` centralizado le permite actualizar un conjunto de vistas con `Style` desde un solo lugar. Todos los ajustes de idioma y plataforma también se pueden integrar a sus estilos.

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

Usando vistas personalizadas

Puede crear vistas personalizadas que se pueden integrar en su página gracias a esas herramientas de ajuste.

Seleccione `File > New > File... > Forms > Forms ContentView (Xaml)` y cree una vista para cada diseño específico: `TabletHome.xaml` y `PhoneHome.xaml` .

Luego seleccione `File > New > File... > Forms > Forms ContentPage HomePage.cs` contenido de `File > New > File... > Forms > Forms ContentPage` y cree un `HomePage.cs` que contenga:

```
using Xamarin.Forms;

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

    }
}
```

Ahora tiene una página de `HomePage` que crea una jerarquía de vista diferente para los idiomas de `Phone` y `Tablet` .

Lea Ajustes visuales específicos de la plataforma. en línea: https://riptutorial.com/es/xamarin-forms/topic/5012/ajustes-visuales-especificos-de-la-plataforma-

Capítulo 6: Almacenamiento en caché

Examples

Almacenamiento en caché utilizando Akavache

Acerca de Akavache

Akavache es una biblioteca increíblemente útil que proporciona una funcionalidad de alcance para el almacenamiento en caché de sus datos. Akavache proporciona una interfaz de almacenamiento de valor clave y funciona en la parte superior de SQLite3. No necesita mantener su esquema sincronizado, ya que en realidad es una solución No-SQL que lo hace perfecto para la mayoría de las aplicaciones móviles, especialmente si necesita que su aplicación se actualice a menudo sin pérdida de datos.

Recomendaciones para Xamarin

Akavache es definitivamente la mejor biblioteca de almacenamiento en caché para la aplicación Xamarin si solo no necesita operar con datos fuertemente relativos, binarios o cantidades realmente grandes de datos. Usa Akavache en los siguientes casos:

- Necesita su aplicación para almacenar en caché los datos durante un período de tiempo determinado (puede configurar el tiempo de espera de caducidad para cada entidad que se está guardando;
- Quieres que tu aplicación funcione sin conexión;
- Es difícil determinar y congelar el esquema de sus datos. Por ejemplo, tienes listas que contienen diferentes objetos escritos;
- Es suficiente para que tenga un simple acceso de valor clave a los datos y no necesita hacer consultas complejas.

Akavache no es una "bala de plata" para el almacenamiento de datos, así que piense dos veces antes de usarlo en los siguientes casos:

- Sus entidades de datos tienen muchas relaciones entre sí;
- Realmente no necesitas tu aplicación para trabajar sin conexión;
- Tienes una gran cantidad de datos para guardar localmente;
- Necesita migrar sus datos de una versión a otra;
- Debe realizar consultas complejas típicas de SQL como agrupación, proyecciones, etc.

  En realidad, puede migrar manualmente sus datos con solo leerlos y escribirlos con campos actualizados.

Ejemplo simple

La interacción con Akavache se realiza principalmente a través de un objeto llamado `BlobCache` .

La mayoría de los métodos de Akavache devuelven observables reactivos, pero también puede esperarlos gracias a los métodos de extensión.

```
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!"));
```

Manejo de errores

```
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()));
```

Lea Almacenamiento en caché en línea: https://riptutorial.com/es/xamarin-forms/topic/3644/almacenamiento-en-cache

Capítulo 7: AppSettings Reader en Xamarin.Forms

Examples

Leyendo el archivo app.config en un proyecto Xamarin.Forms Xaml

Si bien cada plataforma móvil ofrece su propia API de administración de configuraciones, no hay formas integradas de leer configuraciones desde un buen archivo antiguo de .NET estilo app.config xml; Esto se debe a una serie de buenas razones, en particular la api de administración de configuración de .net framework en el lado pesado, y cada plataforma tiene su propia api de sistema de archivos.

Así que construimos una biblioteca PCLAppConfig simple, bien empaquetada en nuget para su consumo inmediato.

Esta biblioteca hace uso de la hermosa biblioteca PCLStorage

En este ejemplo, se supone que está desarrollando un proyecto Xamarin.Forms Xaml, en el que tendría que acceder a la configuración desde su modelo de vista compartido.

1. Inicialice ConfigurationManager.AppSettings en cada uno de los proyectos de su plataforma, justo después de la declaración 'Xamarin.Forms.Forms.Init', como se indica a continuación:

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. Agregue un archivo app.config a su proyecto PCL compartido y agregue sus entradas de configuración de aplicaciones, como haría con cualquier archivo app.config

```
<configuration>
    <appSettings>
```

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

3. Agregue este archivo PCL app.config **como un archivo vinculado** en todos sus proyectos de plataforma. Para Android, asegúrese de configurar la acción de compilación en **'AndroidAsset'** , para UWP configure la acción de compilación en **'Contenido'**

4. Acceda a su configuración: `ConfigurationManager.AppSettings["config.text"];`

Lea AppSettings Reader en Xamarin.Forms en línea: https://riptutorial.com/es/xamarin-forms/topic/5911/appsettings-reader-en-xamarin-forms

Capítulo 8: Base de datos SQL y API en Xamarin Forms.

Observaciones

Crea tu propia api con la base de datos Microsoft SQL y implícalos en la aplicación de formularios Xamarin.

Examples

Crear API utilizando la base de datos SQL e implementar en formularios Xamarin,

Código fuente de blog

Lea Base de datos SQL y API en Xamarin Forms. en línea: https://riptutorial.com/es/xamarin-forms/topic/6513/base-de-datos-sql-y-api-en-xamarin-forms-

Capítulo 9: CarouselView - Versión de prelanzamiento

Observaciones

CarouselView es un control de Xamarin que puede contener cualquier tipo de vista. Este control previo al lanzamiento solo se puede usar en proyectos de Xamarin Forms.

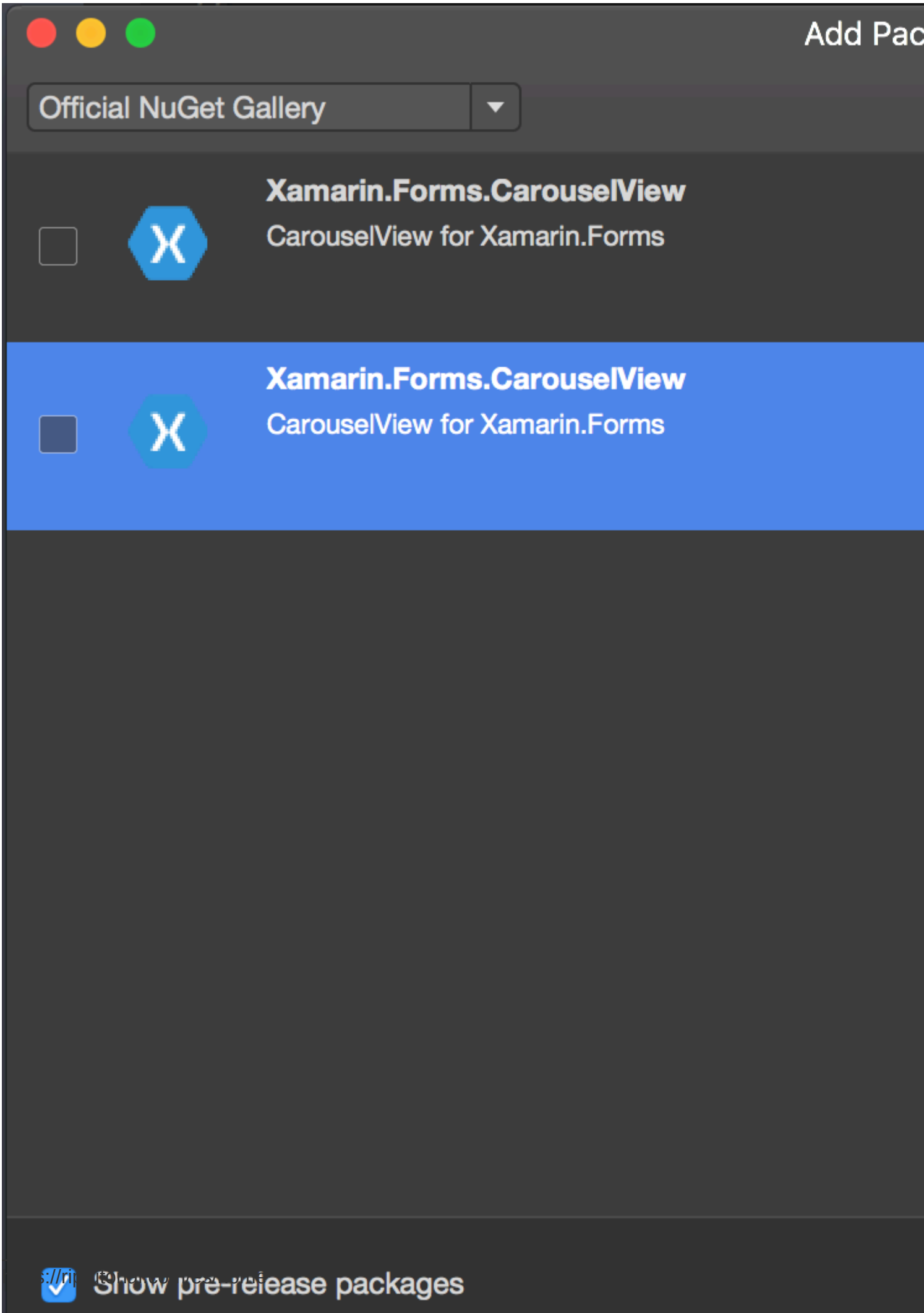
En el ejemplo proporcionado por James Montemagno , en el blog de Xamarin, CarouselView se usa para mostrar imágenes.

En este momento, CarouselView no está integrado en Xamarin.Forms. Para usar esto en su (s) proyecto (s), tendrá que agregar el paquete NuGet (vea el ejemplo anterior).

Examples

Importar CarouselView

La forma más sencilla de importar CarouselView es usar el Administrador de paquetes NuGet en 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/es/xamarin-forms/topic/6094/carouselview---version-de-prelanzamiento

Capítulo 10: Complemento Xamarin

Examples

Compartir Plugin

Una forma sencilla de compartir un mensaje o enlace, copiar texto al portapapeles o abrir un navegador en cualquier aplicación de Xamarin o Windows.

Disponible en 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>
```

DO#

```
 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;
            }
        }
```

Mapas externos

Complemento de mapas externos Abra los mapas externos para navegar a una geolocalización o dirección específica. Opción para iniciar con la opción de navegación en iOS también.

Disponible en 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>
```

Código

```
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");
            };
        }
    }
}
```

Plugin Geolocator

Acceda fácilmente a la geolocalización a través de Xamarin.iOS, Xamarin.Android y Windows.

Nuget disponible: [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>
```

Código

```
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
            {
            }
        }
    }
}
```

Plugin de medios

Toma o elige fotos y videos de una API multiplataforma.

Nuget disponible: [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>
```

Código

```
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");
                }
            };
        }
    }
}
```

Complemento de mensajería

Complemento de mensajería para Xamarin y Windows para hacer una llamada telefónica, enviar un SMS o enviar un correo electrónico utilizando las aplicaciones de mensajería predeterminadas en las diferentes plataformas móviles.

Nuget disponible: [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>
```

Código

```
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");
                }
            };
        }
    }
}
```

Complemento de permisos

Compruebe si sus usuarios han otorgado o denegado permisos para grupos de permisos comunes en iOS y Android.

Además, puede solicitar permisos con una simple API async / awaitified multiplataforma.

Nuget disponible: https://www.nuget.org/packages/Plugin.Permissions ingrese la descripción del enlace aquí **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>
```

Código

```
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;
        }
```

Lea Complemento Xamarin en línea: https://riptutorial.com/es/xamarin-forms/topic/7017/complemento-xamarin

Capítulo 11: Comportamiento específico de la plataforma

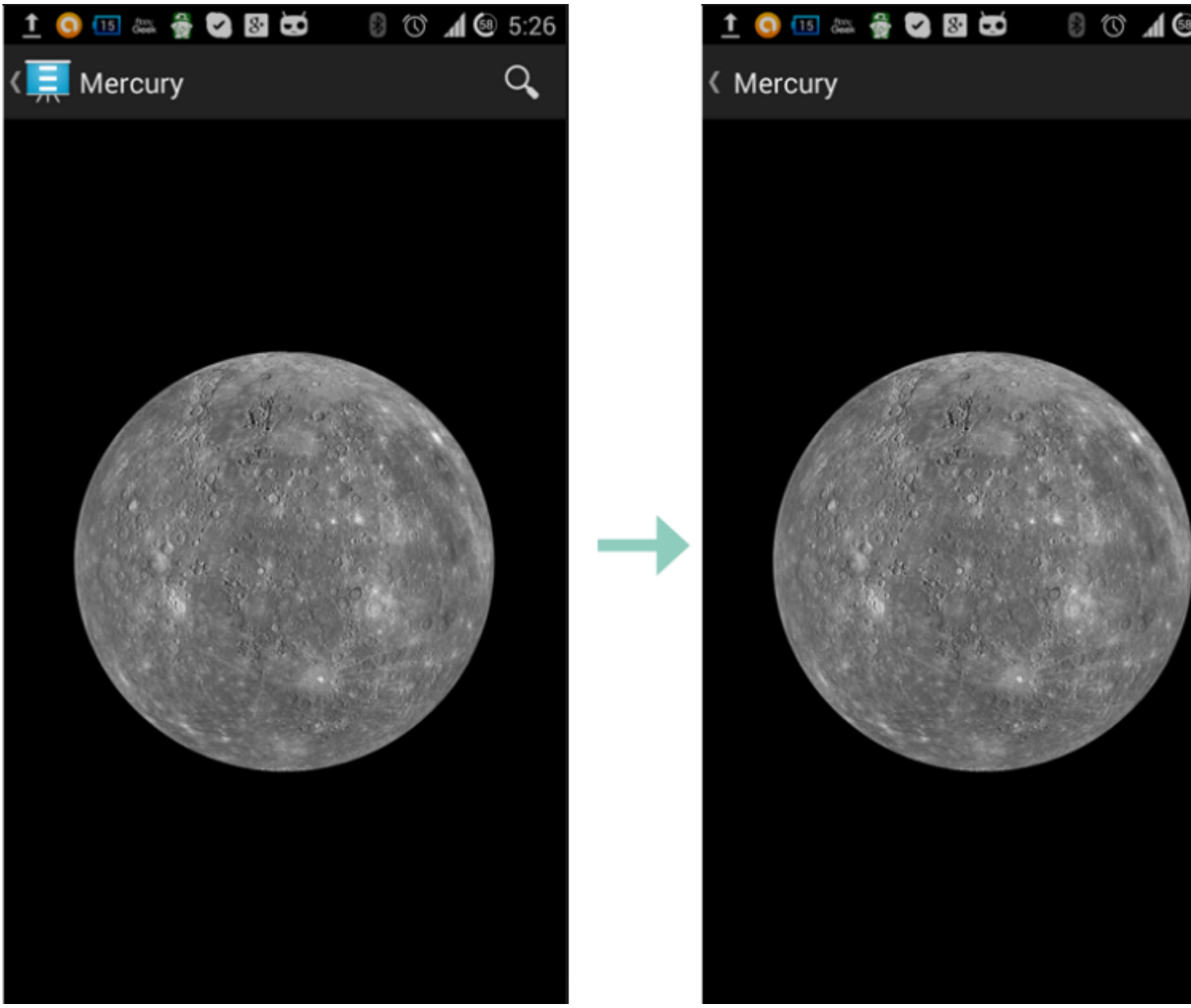
Observaciones

Plataformas de destino

```
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

Eliminando el icono en el encabezado de navegación en Anroid



Usando una pequeña imagen transparente llamada empty.png

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

Aumentar el tamaño de letra de la etiqueta en iOS

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

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

Lea Comportamiento específico de la plataforma en línea: https://riptutorial.com/es/xamarin-forms/topic/6636/comportamiento-especifico-de-la-plataforma

Capítulo 12: Creando controles personalizados

Examples

Cree un control de entrada personalizado de Xamarin Forms (no se requiere nativo)

A continuación se muestra un ejemplo de un control personalizado de Xamarin Forms puro. No se está haciendo una representación personalizada para esto, pero podría implementarse fácilmente; de hecho, en mi propio código, uso este mismo control junto con un procesador personalizado tanto para la `Label` como para la `Entry` .

El control personalizado es un `ContentView` con una `Label` , `Entry` y un `BoxView` dentro de él, que se mantiene en su lugar utilizando 2 `StackLayout` s. También definimos múltiples propiedades `TextChanged` así como un evento `TextChanged` .

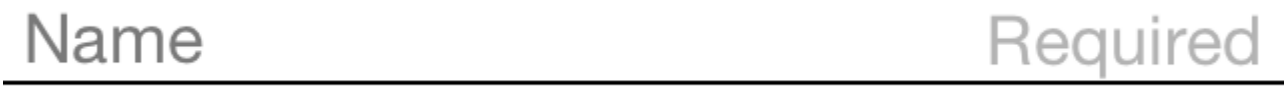
Las propiedades personalizables vinculables funcionan definiéndose como están debajo y teniendo los elementos dentro del control (en este caso, una `Label` y una `Entry`) vinculados a las propiedades personalizables vinculables. Algunas de las propiedades `BindingPropertyChangedDelegate` también necesitan implementar un `BindingPropertyChangedDelegate` para que los elementos delimitados cambien sus valores.

```
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;
    }
}
```

Y aquí hay una imagen del producto final en iOS (la imagen muestra cómo se ve cuando se usa un renderizador personalizado para la `Label` y `Entry` que se usa para eliminar el borde en iOS y para especificar una fuente personalizada para ambos elementos):



Un problema que encontré fue hacer que el `BoxView.BackgroundColor` cambiara cuando `UnderlineColor` cambiaba. Incluso después de enlazar la propiedad `BackgroundColor` `BoxView` , no cambiaría hasta que agregue el delegado `UnderlineColorChanged` .

Etiqueta con colección enlazable de Spans

Creé una etiqueta personalizada con una envoltura alrededor de la propiedad `FormattedText` :

```
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;
    }
}
```

`TextComponent` colección de `TextComponent` personalizados:

```
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); }
    }
}
```

Y cuando la colección de componentes de texto cambia o la propiedad de `Text` de los

componentes separados, reconstruyo la propiedad `FormattedText` de la `Label` base.

Y como lo usé en `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>
```

Código de página:

```
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;
        });
    }
}
```

Crear un control de entrada personalizado con una propiedad MaxLength

El control de `Entry` formularios Xamarin no tiene una propiedad `MaxLength` . Para lograr esto, puede extender la `Entry` como se muestra a continuación, agregando una propiedad de Bindable `MaxLength` . Luego, solo debe suscribirse al evento `TextChanged` en la `Entry` y validar la longitud del

`Text` cuando se llame:

```
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;
    }
}
```

Uso en 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>
```

Lea Creando controles personalizados en línea: https://riptutorial.com/es/xamarin-forms/topic/3913/creando-controles-personalizados

Capítulo 13: Creando controles personalizados

Introducción

Cada vista de `Xamarin.Forms` tiene un renderizador que acompaña a cada plataforma que crea una instancia de un control nativo. Cuando una Vista se representa en la plataforma específica, se `ViewRenderer` una instancia de la clase `ViewRenderer` .

El proceso para hacer esto es el siguiente:

Crea un control personalizado de Xamarin.Forms.

Consume el control personalizado de Xamarin.Forms.

Crea el renderizador personalizado para el control en cada plataforma.

Examples

Implementando un Control CheckBox

En este ejemplo, implementaremos una casilla de verificación personalizada para Android e iOS.

Creando el control personalizado

```
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);
        }
    }
}
```

Comenzaremos con el Representador personalizado de Android creando una nueva clase (`CheckboxCustomRenderer`) en la parte de `Android` de nuestra solución.

Algunos detalles importantes a tener en cuenta:

- Necesitamos marcar la parte superior de nuestra clase con el atributo ExportRenderer para que el renderizador se registre con `Xamarin.Forms` . De esta manera, `Xamarin.Forms` usará este renderizador cuando intente crear nuestro objeto `Checkbox` en `Android` .
- Estamos haciendo la mayor parte de nuestro trabajo en el método `OnElementChanged` , donde instanciamos y configuramos nuestro control nativo.

Consumiendo el control personalizado

```
<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>
```

Creando el renderizador personalizado en cada plataforma

El proceso para crear la clase de renderizador personalizado es el siguiente:

1. Cree una subclase de la `ViewRenderer<T1,T2>` que represente el control personalizado. El primer argumento de tipo debe ser el control personalizado para el que se encuentra el renderizador, en este caso `CheckBox` . El segundo argumento de tipo debe ser el control nativo que implementará el control personalizado.
2. Reemplace el método `OnElementChanged` que representa el control personalizado y la lógica de escritura para personalizarlo. Este método se llama cuando se `Xamarin.Forms` control

`Xamarin.Forms` correspondiente.

3. Agregue un atributo `ExportRenderer` a la clase de renderizador personalizado para especificar que se usará para renderizar el control personalizado `Xamarin.Forms` . Este atributo se utiliza para registrar el renderizador personalizado con `Xamarin.Forms` .

Creando el renderizador personalizado para 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;
            }
        }
    }
}
```

Creando el renderizador personalizado para iOS

Dado que en iOS no hay una casilla de verificación integrada, primero crearemos una `CheckBoxView` y luego crearemos un renderizador para nuestra casilla de verificación de Xamarin.Forms.

`CheckBoxView` se basa en dos imágenes: checked_checkbox.png y unchecked_checkbox.png, por lo que la propiedad `Color` se ignorará.

La vista CheckBox:

```
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
```

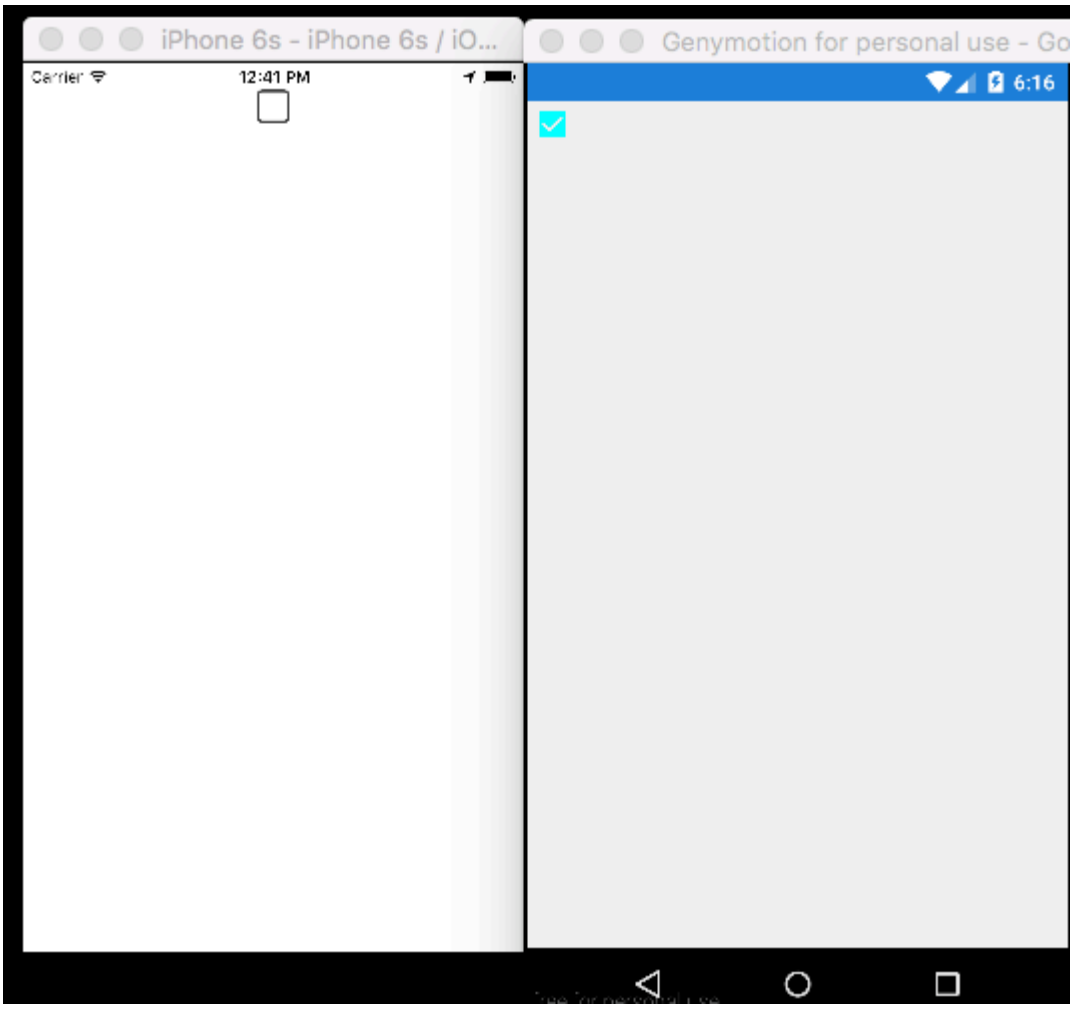
```
        {
            set
            {
                SetTitle(value, UIControlState.Normal);
            }
        }

        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);
        }
    }
}
```

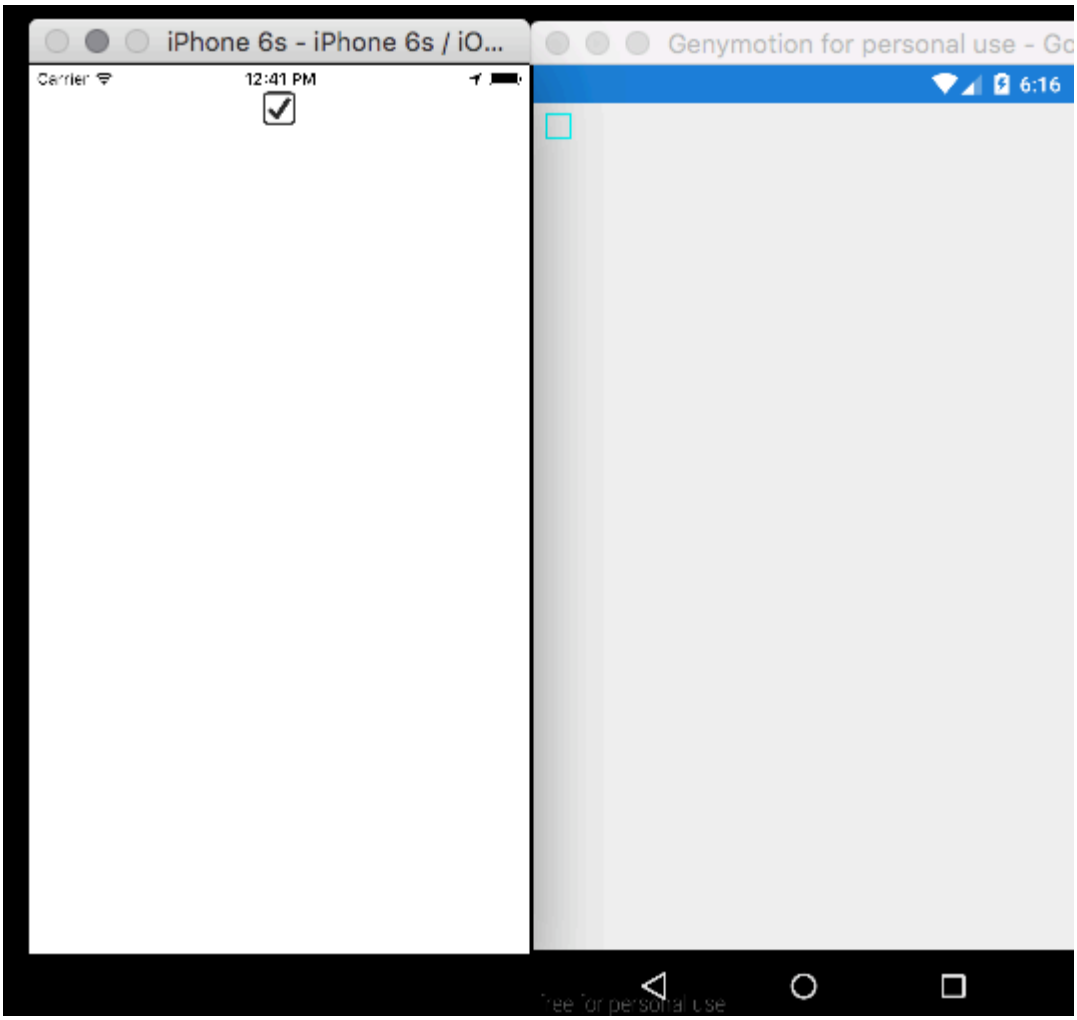
El renderizador personalizado CheckBox:

```
[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;
            }
        }
    }
}
```

Resultado:


iPhone 6s - iPhone 6s / iO...
Carrier
12:41 PM
Genymotion for personal use - Go
6:16



Lea Creando controles personalizados en línea: https://riptutorial.com/es/xamarin-forms/topic/5975/creando-controles-personalizados

Capítulo 14: Creando controles personalizados

Examples

Creando un botón personalizado

```
/// <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
        }
    }
}
```

Uso en 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>
```

Ahora, puedes usar tus propiedades en c #:

```
exampleControl.IntData
```

Tenga en cuenta que debe especificar dónde se ubica su clase TurboButton en su proyecto. Lo he hecho en esta línea:

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

Puedes cambiar libremente "customControls" a otro nombre. Depende de usted cómo lo llamará.

Lea Creando controles personalizados en línea: https://riptutorial.com/es/xamarin-forms/topic/6592/creando-controles-personalizados

Capítulo 15: DependenciaServicio

Observaciones

Cuando usa `DependencyService` , normalmente necesita 3 partes:

- **Interfaz** : esto define las funciones que desea abstraer.
- **Implementación de plataforma** : una clase dentro de cada proyecto específico de plataforma que implementa la interfaz definida previamente.
- **Registro** : cada clase de implementación específica de la plataforma debe registrarse con `DependencyService` través de un atributo de metadatos. Esto permite que `DependencyService` encuentre su implementación en tiempo de ejecución.

Al utilizar `DependencyService` , debe proporcionar una implementación para cada plataforma que elija. Cuando no se proporciona una implementación, la aplicación fallará en el tiempo de ejecución.

Examples

Interfaz

La interfaz define el comportamiento que desea exponer a través del `DependencyService` . Un ejemplo de uso de `DependencyService` es un servicio de Text-to-Speech. Actualmente no hay abstracción para esta función en Xamarin.Forms, por lo que necesita crear el suyo propio. Comience por definir una interfaz para el comportamiento:

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

Como definimos nuestra interfaz, podemos codificar contra ella desde nuestro código compartido.

Nota: Las clases que implementan la interfaz deben tener un constructor sin parámetros para trabajar con `DependencyService` .

implementación de iOS

La interfaz que definió debe implementarse en cada plataforma específica. Para iOS esto se hace a través del marco `AVFoundation` . La siguiente implementación de la interfaz `ITextToSpeech` maneja hablar un texto dado en inglés.

```
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);
    }
}
```

Cuando haya creado su clase, debe permitir que `DependencyService` descubra en tiempo de ejecución. Esto se hace agregando un atributo `[assembly]` encima de la definición de clase y fuera de cualquier definición de espacio de nombres.

```
using AVFoundation;
using DependencyServiceSample.iOS;

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

Este atributo registra la clase con `DependencyService` para que pueda usarse cuando se necesita una instancia de la interfaz `ITextToSpeech` .

Código compartido

Una vez que haya creado y registrado sus clases específicas de la plataforma, puede comenzar a conectarlas a su código compartido. La siguiente página contiene un botón que activa la funcionalidad de texto a voz usando una oración predefinida. Utiliza `DependencyService` para recuperar una implementación específica de la `ITextToSpeech` de `ITextToSpeech` en tiempo de ejecución utilizando los SDK nativos.

```
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;
}
```

Cuando ejecute esta aplicación en un dispositivo iOS o Android y toque el botón, escuchará que la aplicación dice la oración dada.

Implementación de Android

La implementación específica de Android es un poco más compleja porque te obliga a heredar de un objeto `Java.Lang.Object` nativo y te obliga a implementar la interfaz `IOnInitListener` . Android requiere que proporcione un contexto de Android válido para muchos de los métodos SDK que expone. Xamarin.Forms expone un objeto `Forms.Context` que le proporciona un contexto de Android que puede usar en tales casos.

```
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
}
```

Cuando haya creado su clase, debe permitir que `DependencyService` descubra en tiempo de ejecución. Esto se hace agregando un atributo `[assembly]` encima de la definición de clase y fuera de cualquier definición de espacio de nombres.

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


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

Este atributo registra la clase con `DependencyService` para que pueda usarse cuando se necesita una instancia de la interfaz `ITextToSpeech` .

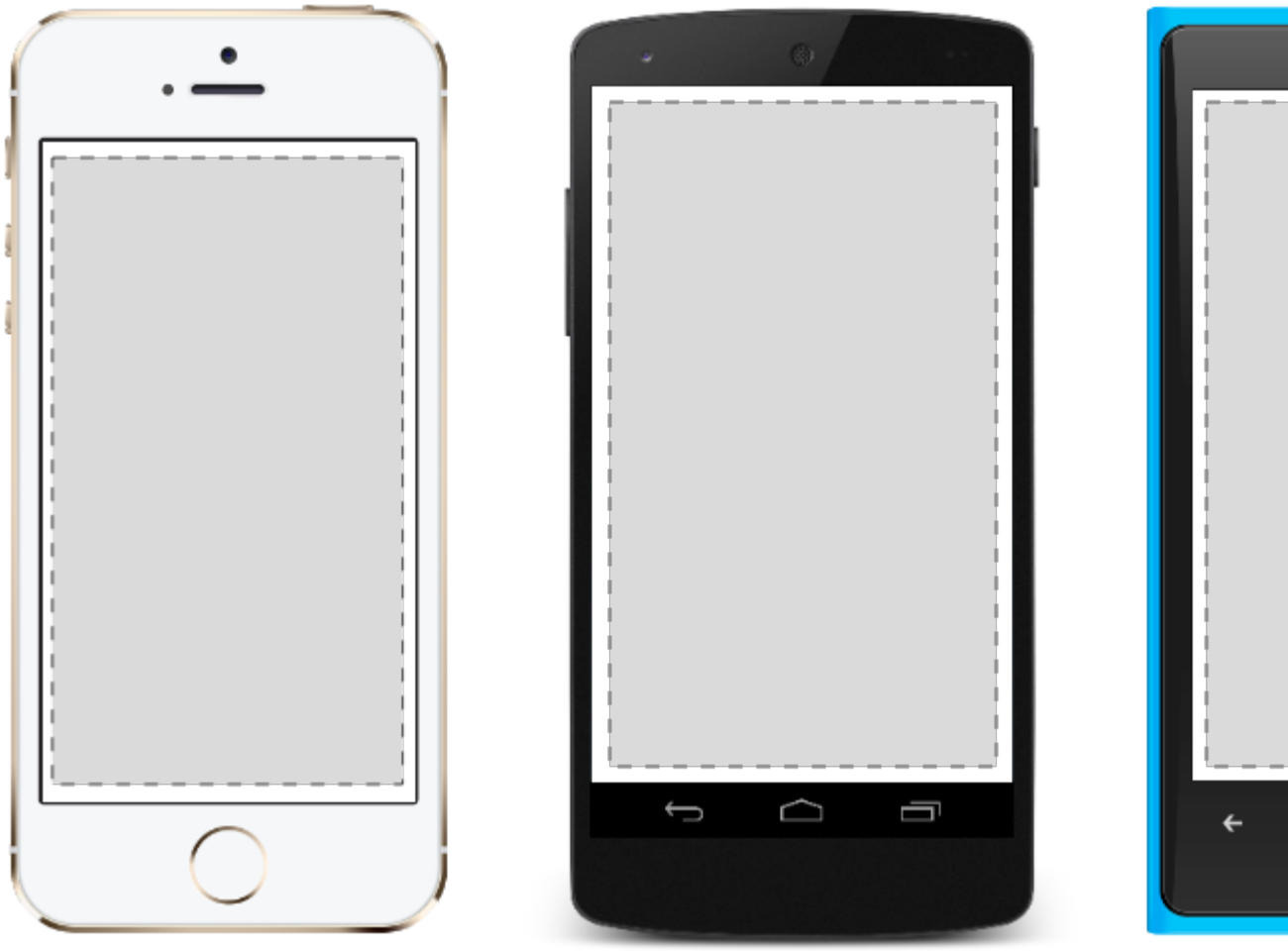
Lea DependenciaServicio en línea: https://riptutorial.com/es/xamarin-forms/topic/2508/dependenciaservicio

Capítulo 16: Diseños de formas de Xamarin

Examples

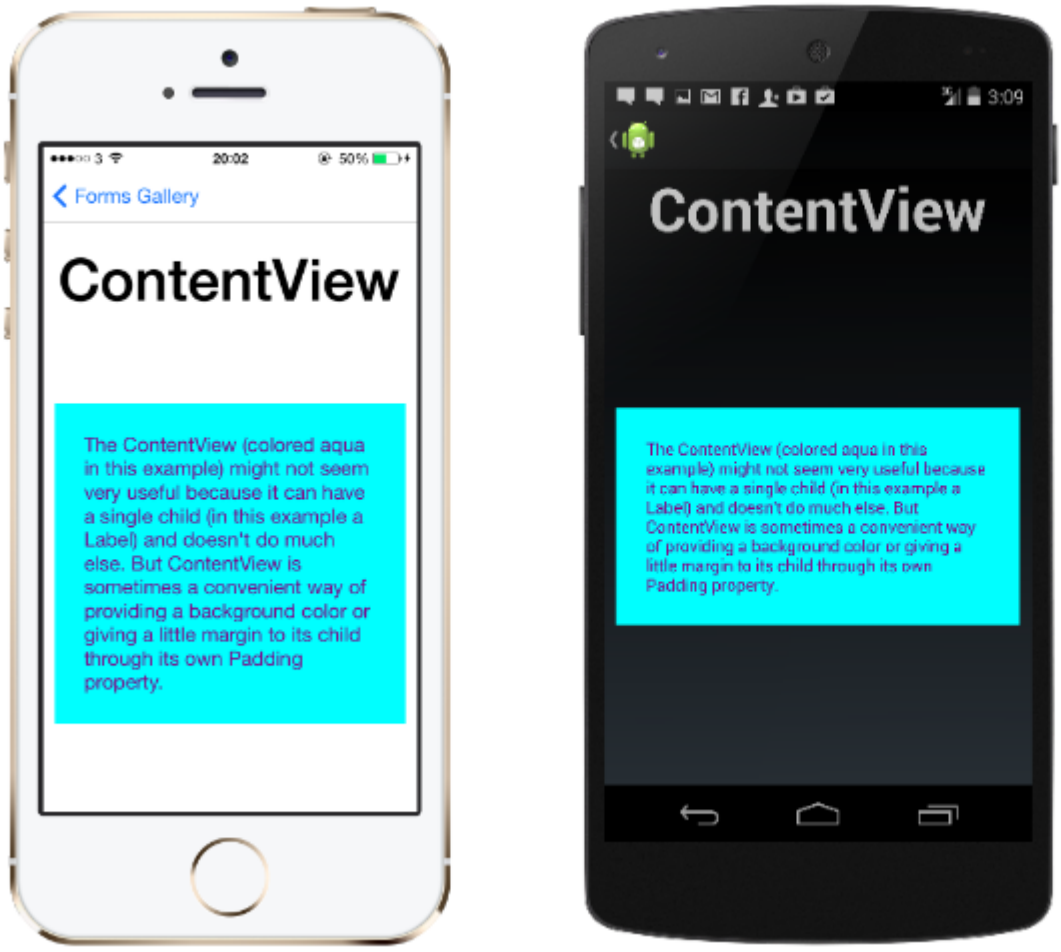
ContentPresenter

Un gestor de diseño para vistas templadas. Se utiliza dentro de un ControlTemplate para marcar donde aparece el contenido que se presentará.



ContentView

Un elemento con un solo contenido. ContentView tiene muy poco uso propio. Su propósito es servir como una clase base para vistas compuestas definidas por el usuario.



XAML

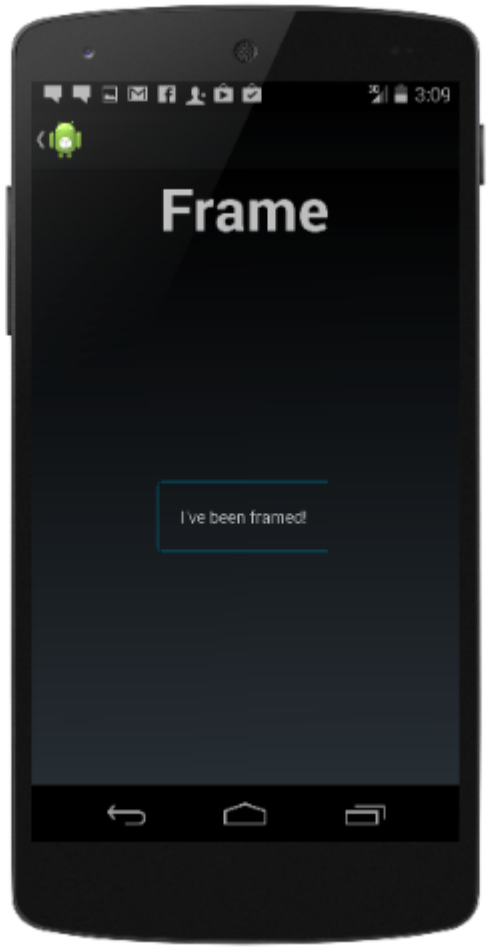
```
<ContentView>
<Label Text="Hi, I'm a simple Label inside of a simple ContentView"
HorizontalOptions="Center"
VerticalOptions="Center"/>
</ContentView>
```

Código

```
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
}
};
```

Cuadro

Un elemento que contiene un solo hijo, con algunas opciones de encuadre. Frame tiene un Xamarin.Forms.Layout.Padding predeterminado de 20.



XAML

```
<Frame>
<Label Text="I've been framed!"
HorizontalOptions="Center"
VerticalOptions="Center" />
</Frame>
```

Código

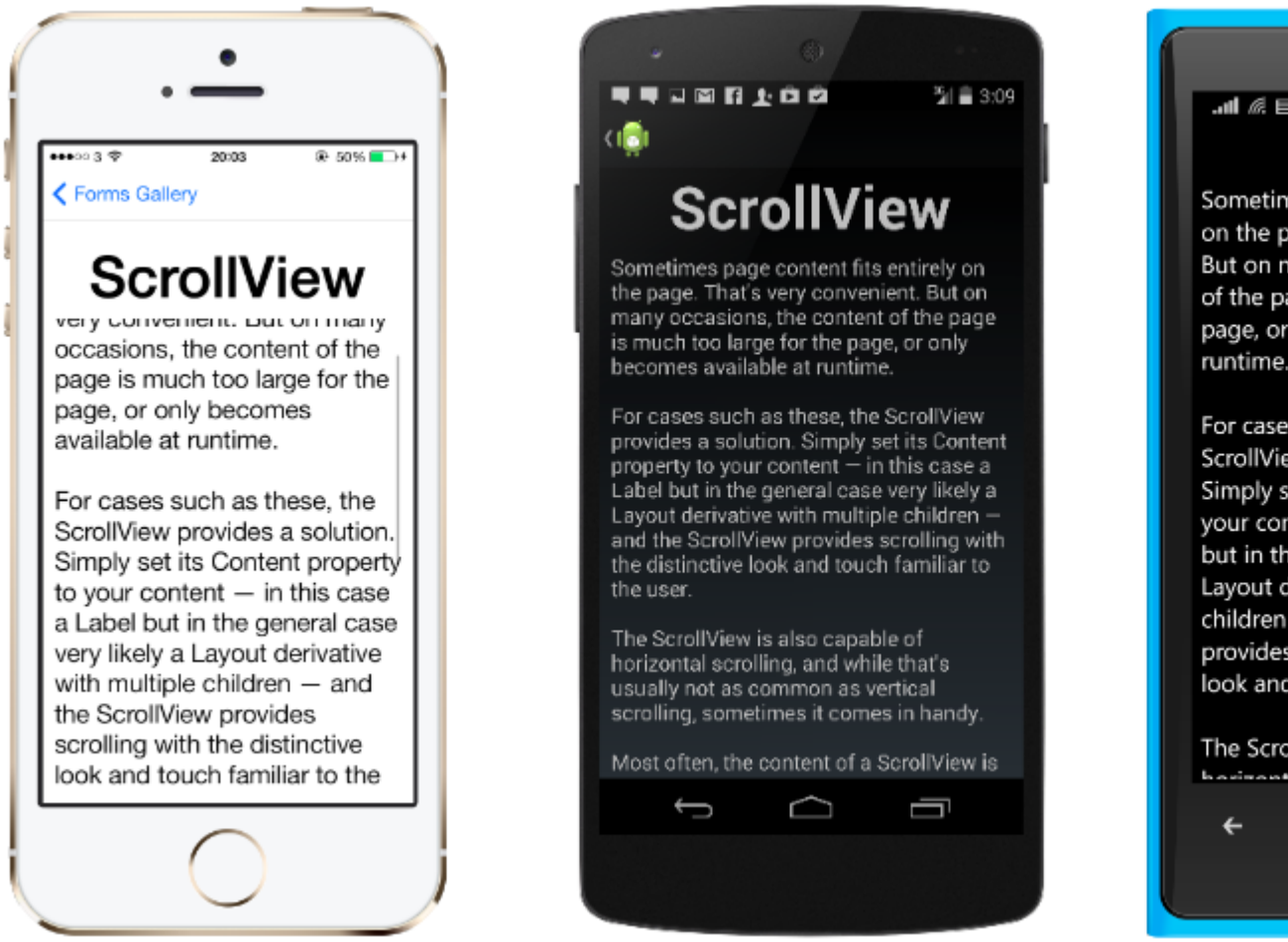
```
var frameView = new Frame {
Content = new Label {
 Text = "I've been framed!",
HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center
},
OutlineColor = Color.Red
};
```

ScrollView

Un elemento capaz de desplazarse si su `Content` requiere.

`ScrollView` contiene diseños y les permite desplazarse fuera de la pantalla. `ScrollView` también se usa para permitir que las vistas se muevan automáticamente a la parte visible de la pantalla

cuando se muestra el teclado.



Nota: `ScrollViews` no debe estar anidado. Además, `ScrollViews` no debe estar anidado con otros controles que proporcionen desplazamiento, como `ListView` y `WebView` .

Un `ScrollView` es fácil de definir. En XAML:

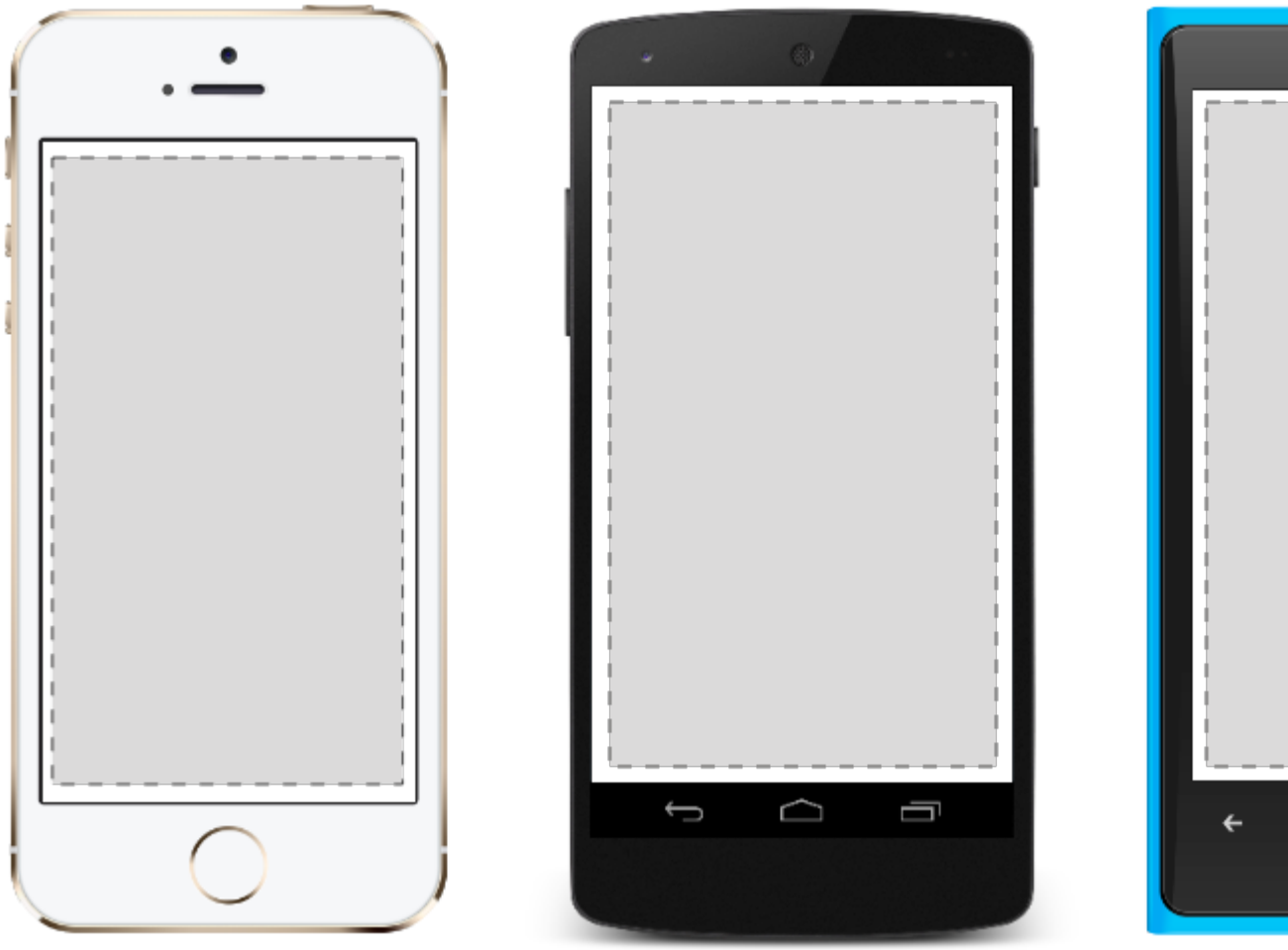
```
<ContentPage.Content>
    <ScrollView>
        <StackLayout>
            <BoxView BackgroundColor="Red" HeightRequest="600" WidthRequest="150" />
            <Entry />
        </StackLayout>
    </ScrollView>
</ContentPage.Content>
```

La misma definición en el código:

```
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());
```

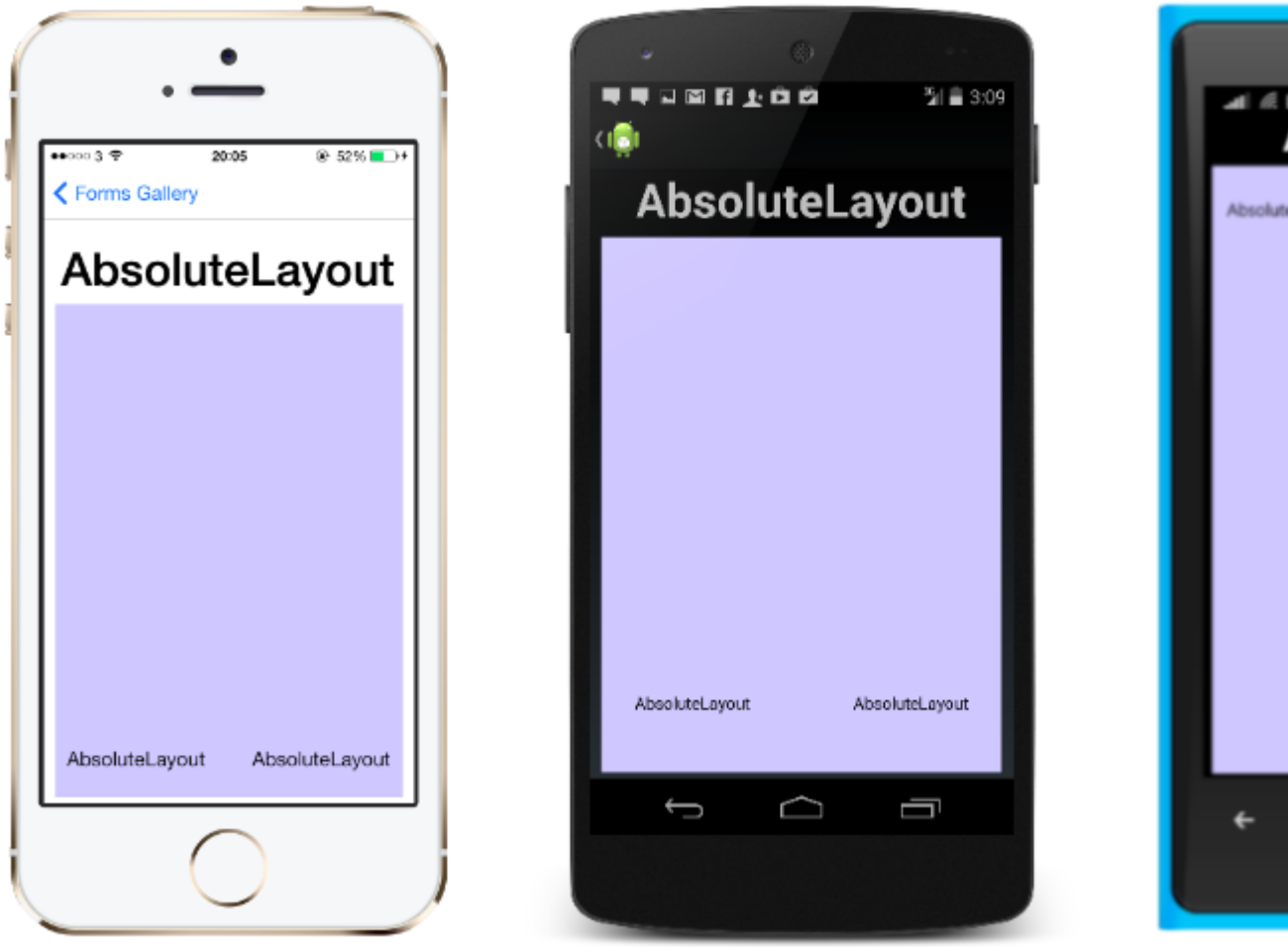
Vista Templated

Un elemento que muestra contenido con una plantilla de control y la clase base para `ContentView` .



AbsoluteLayout

`AbsoluteLayout` coloca y dimensiona elementos secundarios proporcionales a su propio tamaño y posición o por valores absolutos. Las vistas secundarias pueden posicionarse y dimensionarse utilizando valores proporcionales o valores estáticos, y los valores proporcionales y estáticos se pueden mezclar.



Una definición de un `AbsoluteLayout` en XAML se ve así:

```
<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>
```

El mismo diseño se vería así en código:

```
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);
```

El control AbsoluteLayout en Xamarin.Forms le permite especificar dónde exactamente en la pantalla desea que aparezcan los elementos secundarios, así como su tamaño y forma (límites).

Hay algunas maneras diferentes de establecer los límites de los elementos secundarios en función de la enumeración AbsoluteLayoutFlags que se utilizan durante este proceso. La enumeración **AbsoluteLayoutFlags** contiene los siguientes valores:

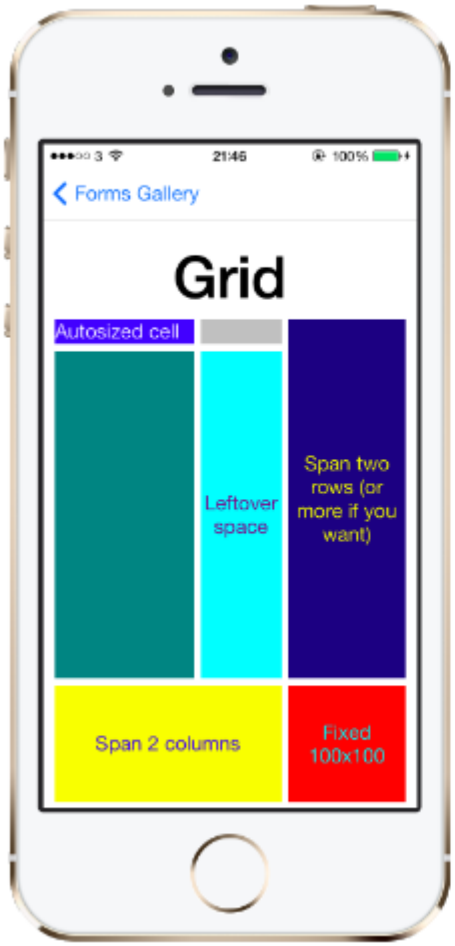
- **Todos** : Todas las dimensiones son proporcionales.
- **AlturaProporcional** : La altura es proporcional al diseño.
- **Ninguna** : no se hace ninguna interpretación.
- **PositionProportional** : Combina XProportional y YProportional.
- **SizeProportional** : Combina WidthProportional y HeightProportional.
- **WidthProportional** : El ancho es proporcional al diseño.
- **XProporcional** : la propiedad X es proporcional al diseño.
- **YProporcional** : la propiedad Y es proporcional al diseño.

El proceso de trabajar con el diseño del contenedor AbsoluteLayout puede parecer un poco contradictorio al principio, pero con un poco de uso se volverá familiar. Una vez que haya creado los elementos secundarios, para establecerlos en una posición absoluta dentro del contenedor, deberá seguir tres pasos. Deseará establecer los indicadores asignados a los elementos utilizando el método **AbsoluteLayout.SetLayoutFlags ()** . También querrá usar el método **AbsoluteLayout.SetLayoutBounds ()** para dar a los elementos sus límites. Finalmente, querrá agregar los elementos secundarios a la colección Children. Como Xamarin.Forms es una capa de

abstracción entre Xamarin y las implementaciones específicas del dispositivo, los valores posicionales pueden ser independientes de los píxeles del dispositivo. Aquí es donde las banderas de diseño mencionadas anteriormente entran en juego. Puede elegir cómo el proceso de diseño de los controles de Xamarin.Forms debe interpretar los valores que define.

Cuadrícula

Un diseño que contiene vistas dispuestas en filas y columnas.



Esta es una definición típica de `Grid` en 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>
```

La misma `Grid` definida en el código se ve así:

```
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) });
```

Para agregar elementos a la cuadrícula: En `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>
```

En el código C #:

```
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);
```

Para `Height` y `Width` una cantidad de unidades disponibles.

- **Automático** : ajusta automáticamente el tamaño para adaptarse al contenido de la fila o columna. Especificado como GridUnitType.Auto en C # o como Auto en XAML.
- **Proporcional** : dimensiona columnas y filas como proporción del espacio restante. Especificado como un valor y GridUnitType.Star en C # y como # * en XAML, siendo # el valor deseado. Especificar una fila / columna con * hará que llene el espacio disponible.
- **Absoluto** : dimensiona columnas y filas con valores específicos de altura y ancho fijos. Especificado como un valor y GridUnitType.Absolute en C # y como # en XAML, siendo # su valor deseado.

Nota: Los valores de ancho de las columnas se establecen como Auto de forma predeterminada en Xamarin.Forms, lo que significa que el ancho se determina a partir del tamaño de los elementos secundarios. Tenga en cuenta que esto difiere de la implementación de XAML en las plataformas de Microsoft, donde el ancho predeterminado es *, que llenará el espacio disponible.

Disposición relativa

Un `Layout` que utiliza `Constraints` para diseñar sus hijos.

`RelativeLayout` se usa para posicionar y ajustar el tamaño de las vistas en relación con las propiedades del diseño o las vistas de hermanos. A diferencia de `AbsoluteLayout` , `RelativeLayout` no tiene el concepto de ancla móvil y no tiene facilidades para posicionar elementos en relación con los bordes inferior o derecho del diseño. `RelativeLayout` admite elementos de posicionamiento fuera de sus propios límites.

Un RelativeLayout en XAML es así:

```
<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
```

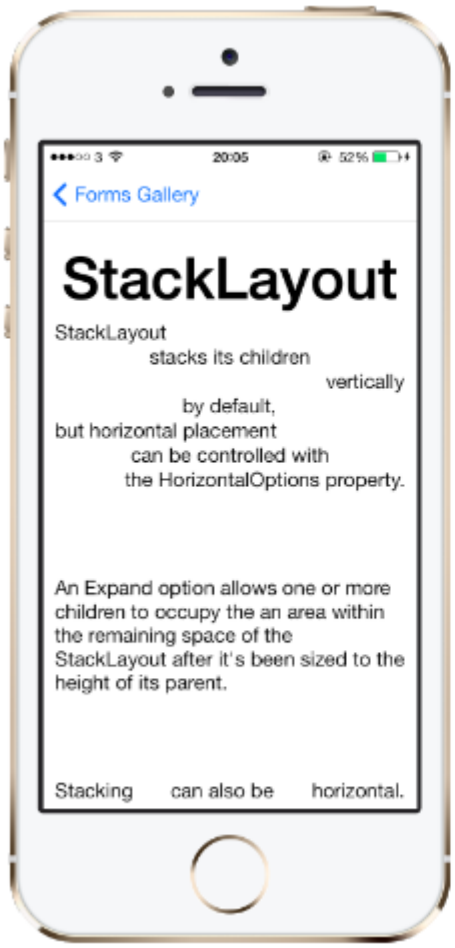
```
            Type=RelativeToParent,Property=Height,Factor=.8,Constant=0}" />
    <BoxView Color="Blue"
        RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToView,
            ElementName=redBox,Property=Y,Factor=1,Constant=20}"
        RelativeLayout.XConstraint="{ConstraintExpression 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>
```

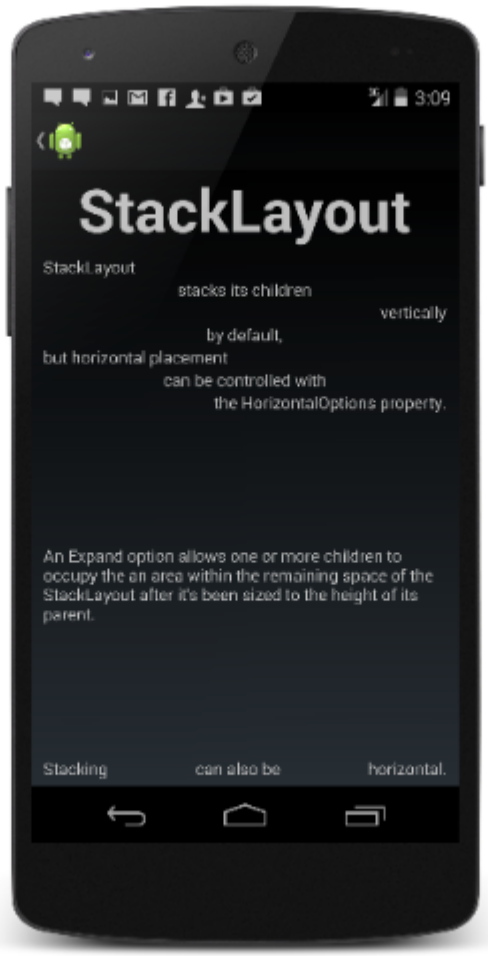
El mismo diseño se puede lograr con este código:

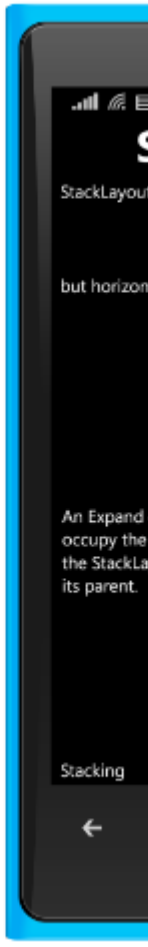
```
layout.Children.Add (redBox, Constraint.RelativeToParent ((parent) => {
        return parent.X;
    }), Constraint.RelativeToParent ((parent) => {
        return parent.Y * .15;
    }), Constraint.RelativeToParent((parent) => {
        return parent.Width;
    }), Constraint.RelativeToParent((parent) => {
        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` organiza las vistas en una línea unidimensional ("stack"), ya sea horizontal o verticalmente. `Views` en un `StackLayout` se pueden dimensionar según el espacio en el diseño utilizando las opciones de diseño. El posicionamiento está determinado por el orden en que se agregaron las vistas al diseño y las opciones de diseño de las vistas.







Uso en XAML

```
<StackLayout>
    <Label Text="This will be on top" />
    <Button Text="This will be on the bottom" />
</StackLayout>
```

Uso en codigo

```
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.",
                },
            }
        }
    }
};
```

Lea Diseños de formas de Xamarin en línea: https://riptutorial.com/es/xamarin-

forms/topic/6273/disenos-de-formas-de-xamarin

Capítulo 17: Disparadores y comportamientos

Examples

Ejemplo de Trigger de Formas Xamarin

`Trigger` son una forma fácil de agregar cierta capacidad de respuesta de UX a su aplicación. Una manera fácil de hacer esto es agregar un `Trigger` que cambie el `TextColor` del `TextColor` una `Label` en función de si su `Entry` relacionada tiene texto ingresado o no.

El uso de un `Trigger` para esto permite que `Label.TextColor` cambie de gris (cuando no se ingresa texto) a negro (tan pronto como los usuarios ingresan texto):

Convertidor (a cada convertidor se le asigna una variable de `Instance` que se usa en el enlace para que no se cree una nueva instancia de la clase cada vez que se usa):

```
/// <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 (el código XAML usa `x:Name` de la `Entry` para averiguar en la propiedad `Entry.Text` tiene más de 3 caracteres de longitud):

```
<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 no se necesita con frecuencia, pero hay algunas situaciones en las que es muy útil. MultiTrigger se comporta de manera similar a Trigger o Data Trigger pero tiene múltiples condiciones. Todas las condiciones deben ser ciertas para que un Setter dispare. Aquí hay un ejemplo simple:

```
<!-- 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>
```

El ejemplo tiene dos entradas diferentes, teléfono y correo electrónico, y una de ellas debe completarse. El MultiTrigger desactiva el botón de envío cuando ambos campos están vacíos.

Lea Disparadores y comportamientos en línea: https://riptutorial.com/es/xamarin-forms/topic/6012/disparadores-y-comportamientos

Capítulo 18: Disposición relativa de Xamarin

Observaciones

El uso de *ForceLayout* en este caso.

Las etiquetas y el tamaño de los botones cambian de acuerdo con el texto dentro de ellos. Por lo tanto, cuando los niños se agregan al diseño, su tamaño permanece en 0 tanto en ancho como en alto. Por ejemplo:

```
relativeLayout.Children.Add(label,
    Constraint.RelativeToParent(parent => label.Width));
```

La expresión anterior devolverá 0 porque el ancho es 0 en este momento. Para *solucionar* esto, necesitamos escuchar el evento *SizeChanged* y cuando el tamaño cambie, debemos forzar el diseño para volver a dibujarlo.

```
label.SizeChanged += (s, e) => relativeLayout.ForceLayout();
```

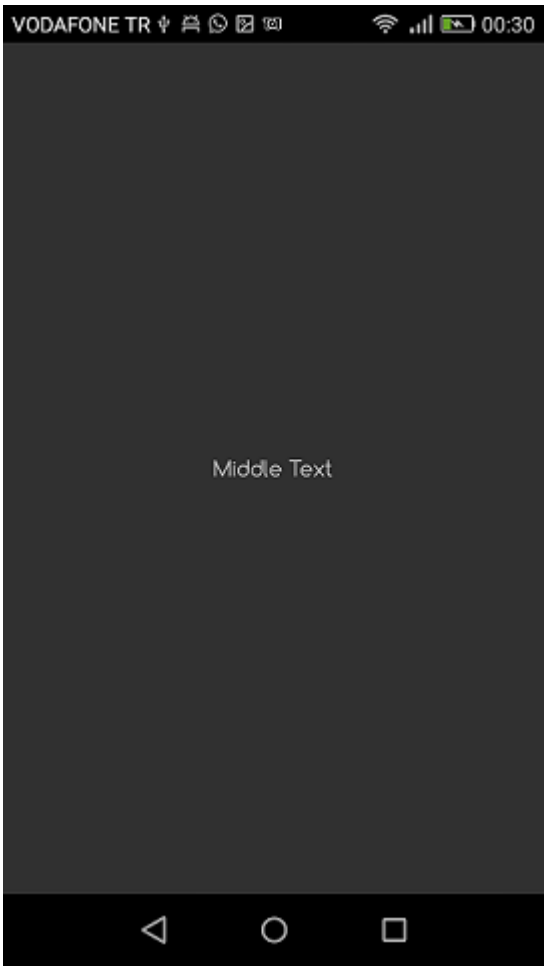
Para una vista como *BoxView* esto es innecesario. Porque podemos definir sus tamaños en la instanciación. Lo otro es que, en ambos casos, podemos definir su ancho y alto como una restricción cuando los estamos agregando al diseño. Por ejemplo:

```
relativeLayout.Children.Add(label,
    Constraint.Constant(0),
    Constraint.Constant(0),
    //Width constraint
    Constraint.Constant(30),
    //Height constraint
    Constraint.Constant(40));
```

Esto agregará la etiqueta al punto 0, 0. El ancho y el alto de la etiqueta serán 30 y 40. Sin embargo, si el texto es demasiado largo, es posible que parte de la misma no se muestre. Si su etiqueta tiene o podría tener una altura alta, puede usar la propiedad *LineBreakMode* de la etiqueta. Que puede envolver el texto. Hay muchas opciones en *LineBreakMode enum* .

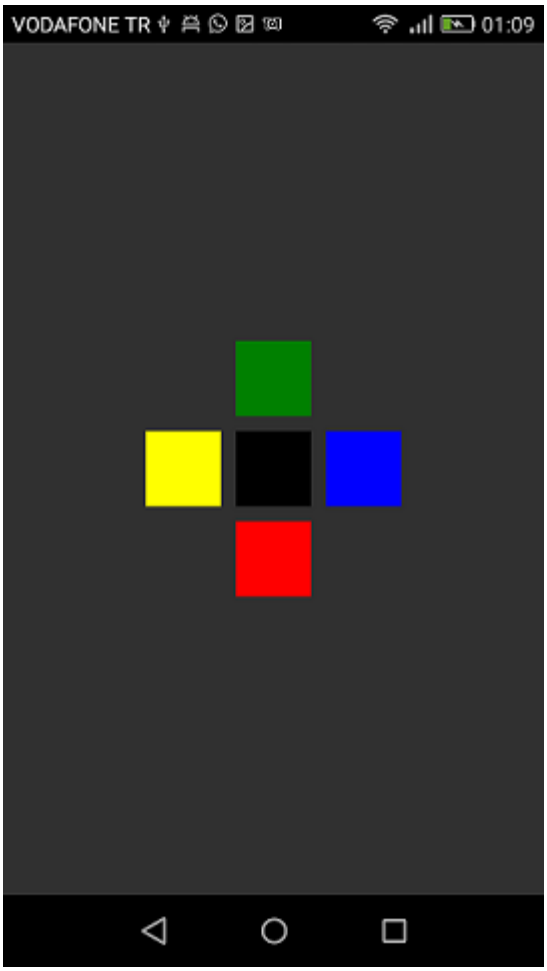
Examples

Página con una etiqueta simple en el medio.




```
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;
    }
}
```

Caja tras caja




```
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;
    }
}
```

Lea Disposición relativa de Xamarin en línea: https://riptutorial.com/es/xamarin-forms/topic/6583/disposicion-relativa-de-xamarin

Capítulo 19: Efectos

Introducción

Efectos simplifica las personalizaciones específicas de la plataforma. Cuando es necesario modificar las propiedades de un control de formularios de Xamarin, se pueden usar los efectos. Cuando es necesario anular los métodos de Xamarin Forms Control, se pueden usar renderizadores personalizados

Examples

Agregar efecto específico de plataforma para un control de entrada

1. Cree una nueva aplicación Xamarin Forms utilizando el archivo PCL -> Nueva solución -> Aplicación multiplataforma -> Xamarin Forms -> Aplicación Forms; Nombra el proyecto como `EffectsDemo`
2. Bajo el proyecto iOS, agregue una nueva clase de `Effect` que herede de la clase `PlatformEffect` y anule los métodos `OnAttached` , `OnDetached` y `OnElementPropertyChanged` Observe los dos atributos `ResolutionGroupName` y `ExportEffect` , estos son necesarios para consumir este efecto del PCL / proyecto compartido.

- `OnAttached` es el método donde la lógica de personalización entra en `OnAttached`
- `OnDetached` es el método en el que se realiza la limpieza y la `OnDetached` del registro
- `OnElementPropertyChanged` es el método que se desencadena en los cambios de propiedad de diferentes elementos. Para identificar la propiedad correcta, verifique el cambio de propiedad exacto y agregue su lógica. En este ejemplo, `OnFocus` dará el color `Blue` y `OutofFocus` dará `Red` color `Red`

```
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);
     }
 }
```

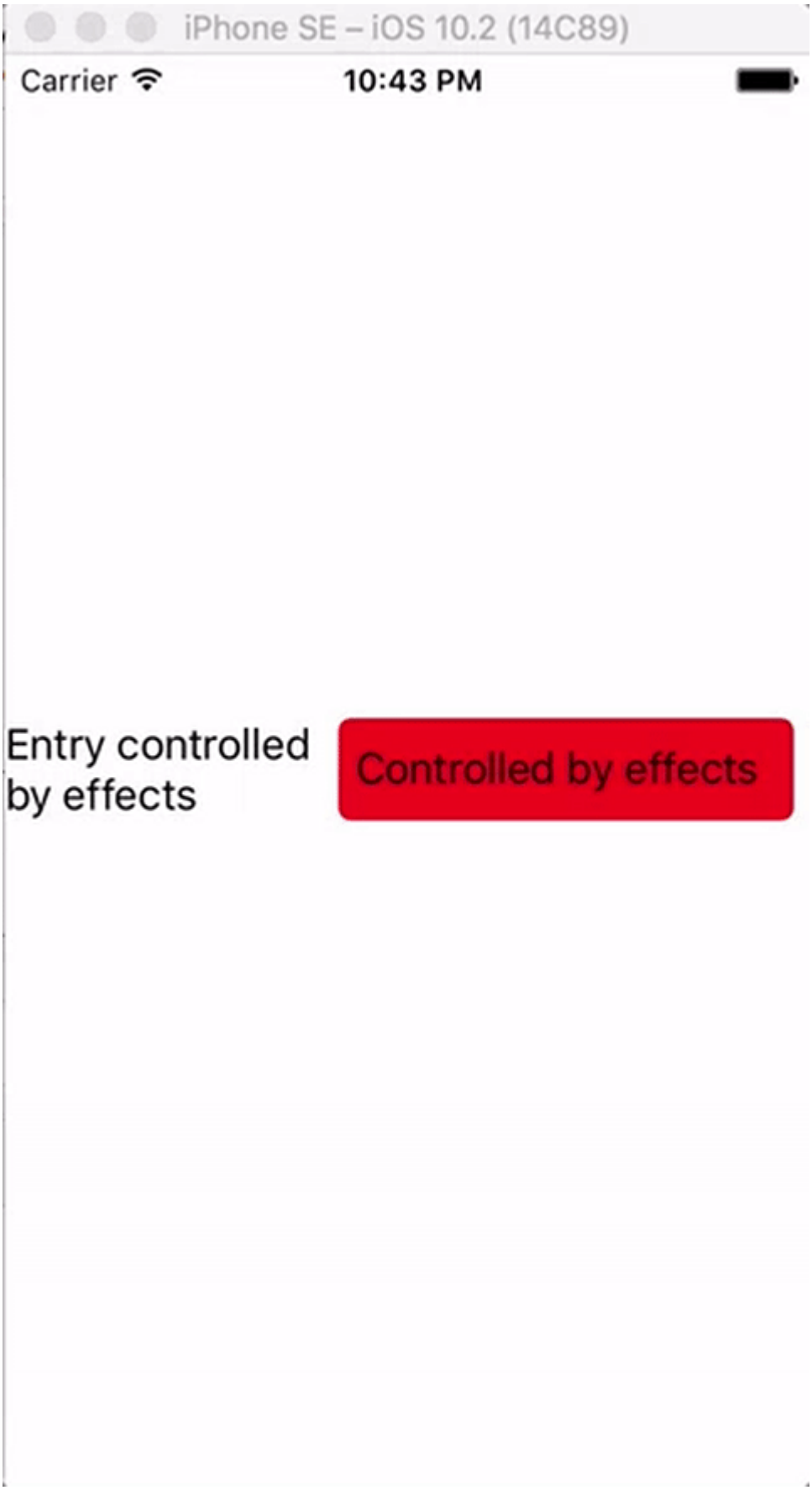
}}

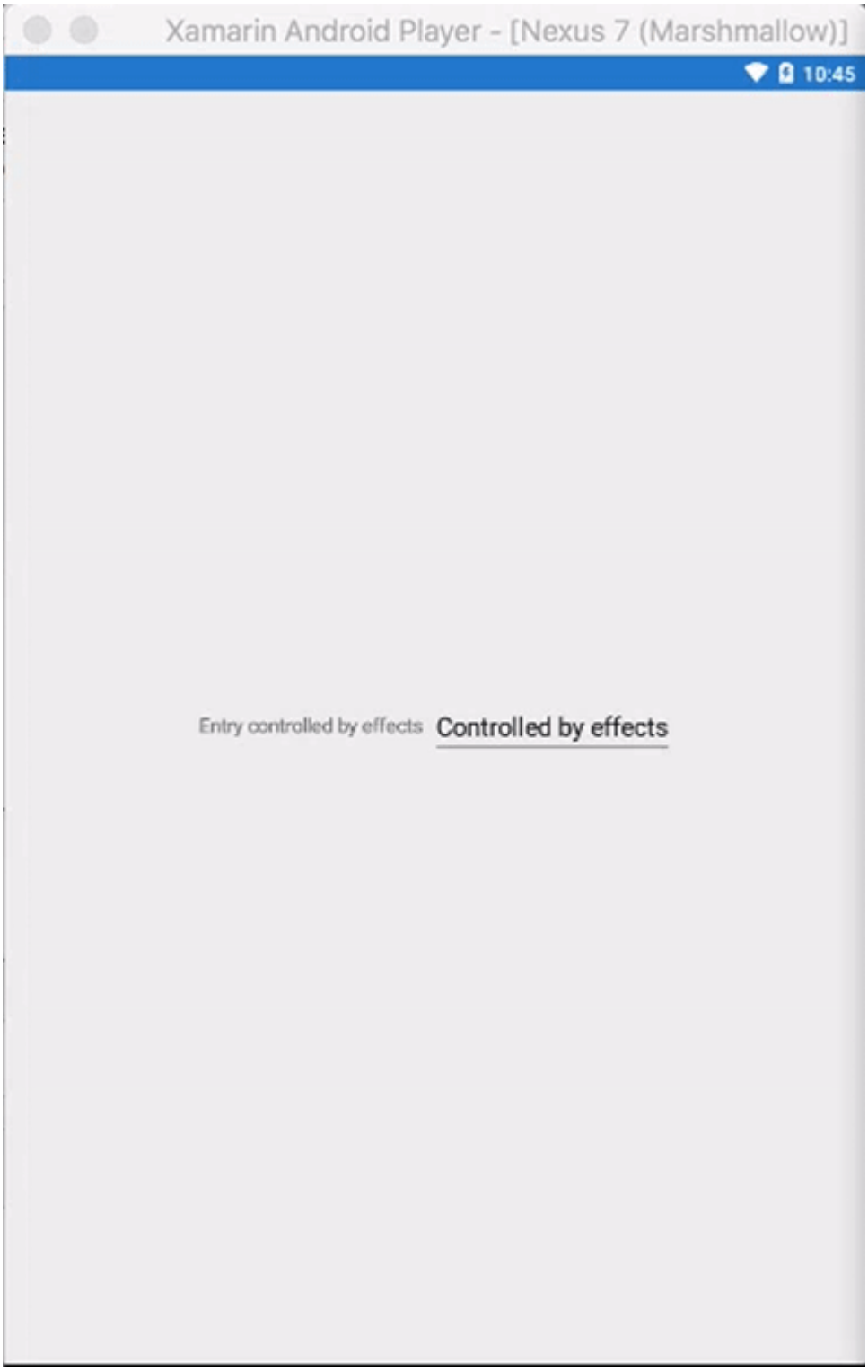
3. Para consumir este efecto en la aplicación, en el proyecto `PCL` , cree una nueva clase llamada `FocusEffect` que herede de `RoutingEffect` . Esto es esencial para que el PCL ejemplifique la implementación específica del efecto en la plataforma. Código de muestra a continuación:

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

4. Agrega el efecto al control de `Entry` en el 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



Dado que el efecto se implementó solo en la versión de iOS, cuando la aplicación se ejecuta en el `iOS Simulator` al enfocar los cambios de color de fondo de la `Entry` y no sucede nada en el `Android Emulator` ya que el `Effect` no se creó en el proyecto `Droid`

Lea Efectos en línea: https://riptutorial.com/es/xamarin-forms/topic/9252/efectos

Capítulo 20: El enlace de datos

Observaciones

Posibles excepciones

System.ArrayTypeMismatchException: se intentó acceder a un elemento como un tipo incompatible con la matriz.

Esta excepción puede ocurrir cuando se intenta vincular una colección a una propiedad no vinculable cuando está habilitada la precompilación XAML. Un ejemplo común es intentar enlazar con `Picker.Items` . Vea abajo.

System.ArgumentException: el objeto de tipo 'Xamarin.Forms.Binding' no se puede convertir al tipo 'System.String'.

Esta excepción puede ocurrir cuando se intenta vincular una colección a una propiedad no vinculable cuando la precompilación de XAML está deshabilitada. Un ejemplo común es intentar enlazar con `Picker.Items` . Vea abajo.

El `Picker.Items` propiedad no es enlazable

Este código causará un error:

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

La excepción puede ser una de las siguientes:

> System.ArrayTypeMismatchException: se intentó acceder a un elemento como un tipo incompatible con la matriz.

o

> System.ArgumentException: el objeto de tipo 'Xamarin.Forms.Binding' no se puede convertir al tipo 'System.String'.

Específicamente, la propiedad `Items` no es vinculable. Las soluciones incluyen crear su propio control personalizado o usar un control de terceros, como `BindablePicker` de FreshEssentials . Después de instalar el paquete NuGet de FreshEssentials en su proyecto, el control

`BindablePicker` del paquete con una propiedad de `ItemsSource` está disponible:

```
<?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

Enlace básico a 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));
        }
    }
}
```

Lea El enlace de datos en línea: https://riptutorial.com/es/xamarin-forms/topic/3915/el-enlace-de-datos

Capítulo 21: Examen de la unidad

Examples

Probando los modelos de vista.

Antes que empecemos...

En términos de capas de aplicación, su ViewModel es una clase que contiene toda la lógica empresarial y las reglas que hacen que la aplicación haga lo que debe de acuerdo con los requisitos. También es importante hacerlo lo más independiente posible, reduciendo las referencias a la IU, la capa de datos, las funciones nativas y las llamadas a la API, etc. Todo esto hace que su máquina virtual sea verificable.
En resumen, su ViewModel:

- No debe depender de las clases de UI (vistas, páginas, estilos, eventos);
- No debe usar datos estáticos de otras clases (tanto como pueda);
- Debe implementar la lógica de negocios y preparar los datos que se deben en la interfaz de usuario;
- Debe utilizar otros componentes (base de datos, HTTP, específicos de la interfaz de usuario) a través de interfaces que se resuelven utilizando la inyección de dependencia.

  Su ViewModel también puede tener propiedades de otros tipos de máquinas virtuales. Por ejemplo, `ContactsPageViewModel` tendrá una propiedad de tipo de colección como `ObservableCollection<ContactListItemViewModel>`

Requisitos de negocio

Digamos que tenemos la siguiente funcionalidad para implementar:

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

Después de aclarar la historia del usuario definimos los siguientes escenarios:

```
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
```

Nos quedaremos solo con estos dos escenarios. Por supuesto, debería haber muchos más casos y debería definirlos todos antes de la codificación real, pero ahora es suficiente para que nos familiaricemos con las pruebas unitarias de los modelos de vista.

Sigamos el enfoque clásico de TDD y comencemos escribiendo una clase vacía que se está probando. Luego, escribiremos pruebas y las haremos verdes implementando la funcionalidad empresarial.

Clases comunes

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

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

Servicios

¿Recuerda que nuestro modelo de vista no debe utilizar las clases de UI y HTTP directamente? Debe definirlos como abstracciones en lugar de depender de los detalles de la implementación .

```
/// <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);
}
```

Construyendo el trozo de ViewModel

Ok, vamos a tener la clase de página para la pantalla de inicio de sesión, pero comencemos con ViewModel primero:

```
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);
    }
}
```

Definimos dos propiedades de `string` y un comando para enlazar en la interfaz de usuario. No describiremos cómo construir una clase de página, el marcado XAML y vincularlo con ViewModel en este tema, ya que no tienen nada específico.

¿Cómo crear una instancia LoginPageViewModel?

Creo que probablemente estabas creando las máquinas virtuales solo con el constructor. Ahora, como puede ver, nuestra máquina virtual depende de que se inyecten 2 servicios como parámetros de constructor, por lo que no podemos simplemente hacer `var viewModel = new LoginPageViewModel()` . Si no está familiarizado con la inyección de dependencia , es el mejor momento para aprender sobre él. La prueba de unidad adecuada es imposible sin conocer y seguir este principio.

Pruebas

Ahora vamos a escribir algunas pruebas de acuerdo con los casos de uso mencionados anteriormente. En primer lugar, debe crear un nuevo conjunto (solo una biblioteca de clases o seleccionar un proyecto de prueba especial si desea usar las herramientas de prueba de unidad de Microsoft). Asígnele un nombre similar a `ProjectName.Tests` y agregue una referencia a su proyecto PCL original.

En este ejemplo voy a usar NUnit y Moq, pero puede continuar con las libs de prueba que desee. No habrá nada especial con ellos.

Ok, esa es la clase de prueba:

```
[TestFixture]
public class LoginPageViewModelTest
{
}
```

Pruebas de escritura

Aquí están los métodos de prueba para los dos primeros escenarios. Intente mantener 1 método de prueba por 1 resultado esperado y no verifique todo en una prueba. Eso le ayudará a recibir informes más claros sobre lo que ha fallado en el código.

```
[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;
    }
}
```

Y aquí vamos:

Ahora el objetivo es escribir la implementación correcta para el método de `Login` de `Login` de ViewModel y eso es todo.

Implementación de la lógica de negocios.

```
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);
```

```
    }
}
```

Y después de ejecutar las pruebas de nuevo:

Ahora puede seguir cubriendo su código con nuevas pruebas, lo que lo hace más estable y seguro para la regresión.

Lea Examen de la unidad en línea: https://riptutorial.com/es/xamarin-forms/topic/3529/examen-de-la-unidad

Capítulo 22: Fuentes personalizadas en estilos

Observaciones

Recursos para mirar:

- Estilos de Xamarin
- Uso de fuentes personalizadas en iOS y Android con Xamarin.Forms
- Renderizadores personalizados
- Diccionarios de recursos
- Propiedades adjuntas

Examples

Acceso a fuentes personalizadas en Syles

Xamarin.Forms proporciona un gran mecanismo para diseñar su aplicación multiplataforma con estilos globales.

En el mundo móvil, su aplicación debe ser bonita y sobresalir de las demás aplicaciones. Uno de estos caracteres son las fuentes personalizadas utilizadas en la aplicación.

Con la potencia de XAML Styling en Xamarin.Forms acaba de crear un estilo base para todas las etiquetas con sus fuentes personalizadas.

Para incluir fuentes personalizadas en su proyecto iOS y Android, siga la guía en Uso de fuentes personalizadas en iOS y Android con Xamarin.Forms publicado por Gerald.

Declare Style en la sección de recursos del archivo App.xaml. Esto hace que todos los estilos sean visibles globalmente.

Desde la publicación de Gerald anterior, necesitamos usar la propiedad StyleId pero no es una propiedad vinculable, así que para usarla en Style Setter necesitamos crear una propiedad adjuntada para ella:

```
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);
        }
    }
}
```

Luego agregue el estilo en el recurso App.xaml:

```
<?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>
```

De acuerdo con la publicación anterior, debemos crear un Renderizador personalizado para la etiqueta que hereda de LabelRenderer en la plataforma Android.

```
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;
        }
    }
}
```

Para la plataforma iOS no se requieren renderizadores personalizados.

Ahora puedes obtener el estilo en el marcado de tu página:

Para etiqueta específica

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

O aplique el estilo a todas las etiquetas en la página creando Estilo Basado en 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>
```

Lea Fuentes personalizadas en estilos en línea: https://riptutorial.com/es/xamarin-forms/topic/4854/fuentes-personalizadas-en-estilos

Capítulo 23: Gesto de Xamarin

Examples

Toque gesto

Con el gesto de toque, puede hacer que se pueda hacer clic en cualquier elemento de la interfaz de usuario (imágenes, botones, apilamientos, ...):

(1) En código, usando el evento:

```
var tapGestureRecognizer = new TapGestureRecognizer();
tapGestureRecognizer.Tapped += (s, e) => {
    // handle the tap
};
image.GestureRecognizers.Add(tapGestureRecognizer);
```

(2) En el código, usando `ICommand` (con MVVM-Pattern , por ejemplo):

```
var tapGestureRecognizer = new TapGestureRecognizer();
tapGestureRecognizer.SetBinding (TapGestureRecognizer.CommandProperty, "TapCommand");
image.GestureRecognizers.Add(tapGestureRecognizer);
```

(3) O en Xaml (con evento e `ICommand` , solo se necesita uno):

```
<Image Source="tapped.jpg">
    <Image.GestureRecognizers>
        <TapGestureRecognizer Tapped="OnTapGestureRecognizerTapped" Command="{Binding
TapCommand"} />
  </Image.GestureRecognizers>
</Image>
```

Lea Gesto de Xamarin en línea: https://riptutorial.com/es/xamarin-forms/topic/7994/gesto-de-xamarin

Capítulo 24: Gesto de Xamarin

Examples

Evento gestual

Cuando ponemos el control de Etiqueta, la Etiqueta no proporciona ningún evento. <Etiqueta x: Nombre = "lblSignUp Text =" ¿No tienes cuenta? "/> Como se muestra, la etiqueta solo muestra el propósito.

Cuando el usuario desea reemplazar el botón con la etiqueta, le damos el evento para la etiqueta. Como se muestra abajo:

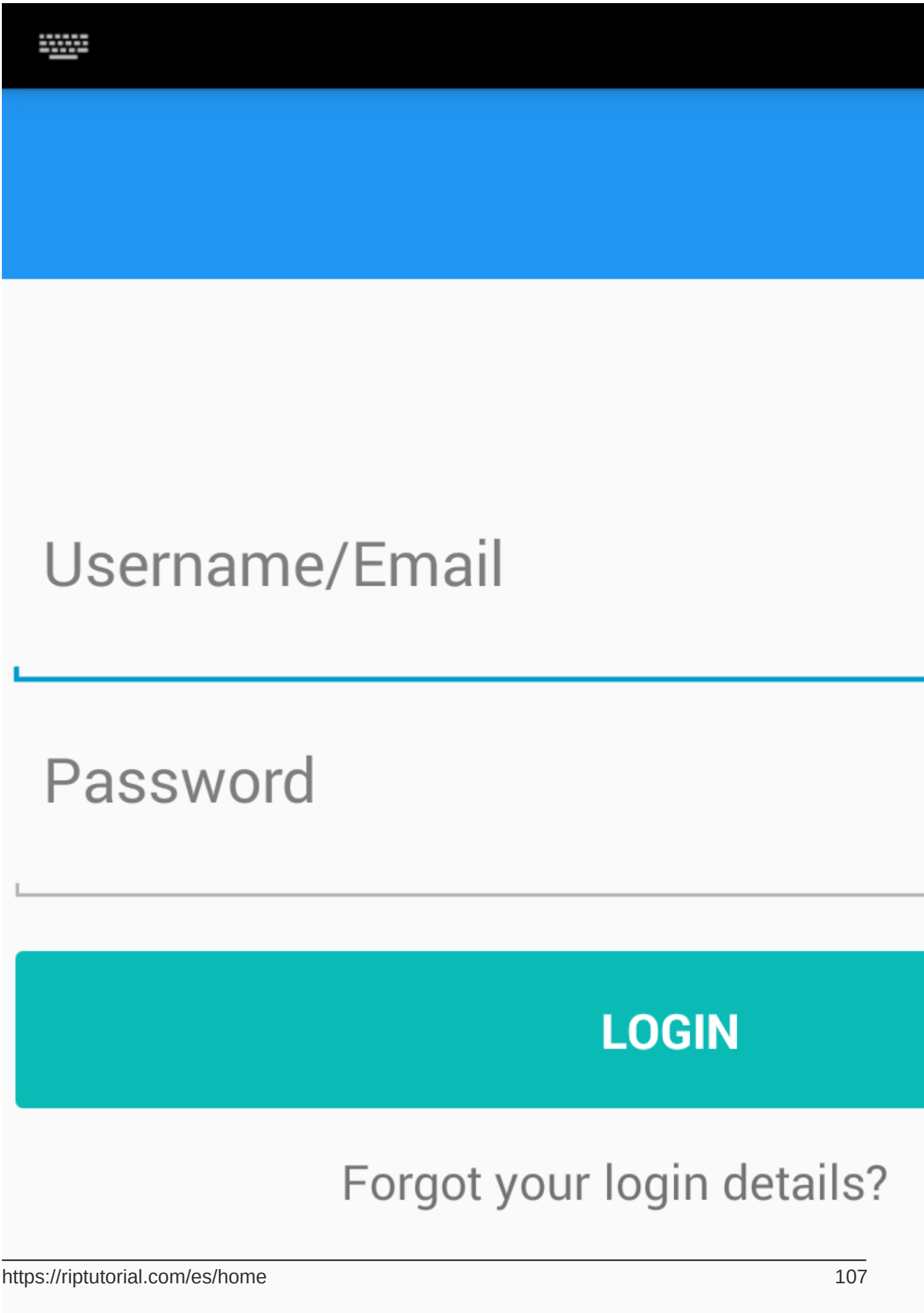
XAML

```
<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>
```

DO#

```
var lblSignUp_Tapped = new TapGestureRecognizer();
lblSignUp_Tapped.Tapped += (s,e) =>
{
//
//  Do your work here.
//
};
lblSignUp.GestureRecognizers.Add(lblSignUp_Tapped);
```

La pantalla de abajo muestra el evento de etiqueta. Pantalla 1: La etiqueta "¿No tienes una cuenta?" como se muestra en la parte inferior.


Username/Email
Password
LOGIN
Forgot your login details?

https://riptutorial.com/es/xamarin-forms/topic/8009/gesto-de-xamarin

Capítulo 25: Gestos

Examples

Haga que una imagen pueda ser ejecutada agregando un TapGestureRecognizer

Hay un par de reconocedores predeterminados disponibles en Xamarin.Forms, uno de ellos es el `TapGestureRecognizer` .

Puedes añadirlos a prácticamente cualquier elemento visual. Echa un vistazo a una implementación simple que se une a una `Image` . Aquí es cómo hacerlo en código.

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

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

O en XAML:

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

Aquí el comando se establece mediante el enlace de datos. Como puede ver, también puede configurar `NumberOfTapsRequired` para habilitarlo para más toques antes de que actúe. El valor predeterminado es 1 toque.

Otros gestos son Pinch y Pan.

Acercar una imagen con el gesto de pellizco

Para poder hacer que una `Image` (o cualquier otro elemento visual) pueda hacer zoom, debemos agregarle un `PinchGestureRecognizer` . Aquí está cómo hacerlo en código:

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

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

Pero también se puede hacer desde XAML:

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

En el controlador de eventos que se acompaña, debe proporcionar el código para ampliar su imagen. Por supuesto, otros usos pueden ser implementados también.

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

Otros gestos son Tap y Pan.

Muestra todo el contenido de la imagen ampliada con PanGestureRecognizer

Cuando tenga una `Image` ampliada (u otro contenido) puede arrastrarla alrededor de la `Image` para mostrar todo su contenido en el estado ampliado.

Esto se puede lograr mediante la implementación de PanGestureRecognizer. Desde el código se ve así:

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

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

Esto también se puede hacer desde XAML:

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

En el evento de código subyacente ahora puede manejar el panorama en consecuencia. Use esta firma de método para manejarlo:

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

Coloque un pin donde el usuario tocó la pantalla con MR.Gestures

Los Xamarins incorporados en los reconocedores de gestos proporcionan un manejo táctil muy básico. Por ejemplo, no hay forma de obtener la posición de un dedo que toca. MR.Gestures es un componente que agrega 14 eventos diferentes de manejo táctil. La posición de los dedos que tocan es parte de los `EventArgs` pasados a todos los eventos de MR.Gestures.

Si desea colocar un pin en cualquier lugar de la pantalla, la forma más sencilla es usar un `MR.Gestures.AbsoluteLayout` que controla el evento `Tapping` .

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

Como puede ver, el `Tapping="OnTapping"` también se parece más a .NET que a la sintaxis de Xamarins con los `GestureRecognizers` anidados. Esa sintaxis fue copiada de iOS y huele un poco a los desarrolladores de .NET.

En su código, detrás de usted, podría agregar el controlador `OnTapping` esta manera:

```
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);
    }
}
```

En lugar del evento `Tapping` , también puede usar `TappingCommand` y enlazar con su ViewModel, pero eso complicaría las cosas en este sencillo ejemplo.

Se pueden encontrar más muestras de MR.Gestures en la aplicación GestureSample en GitHub y en el sitio web de MR.Gestures . Estos también muestran cómo utilizar todos los demás eventos táctiles con controladores de eventos, comandos, MVVM, ...

Lea Gestos en línea: https://riptutorial.com/es/xamarin-forms/topic/3914/gestos

Capítulo 26: Manejo de excepciones

Examples

Una forma de informar acerca de las excepciones en iOS

Vaya al archivo `Main.cs` en el **proyecto iOS** y cambie el código existente, como se presenta a continuación:

```
    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` interfaz `ILittleWatson` , utilizada en código portátil, podría tener este aspecto:

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

    void SaveReport(Exception ex);
}
```

Implementación para **proyecto iOS** :

```
[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);
        }
    }
}
```

Y luego, en algún lugar, donde comienza la aplicación, pon:

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

Lea Manejo de excepciones en línea: https://riptutorial.com/es/xamarin-forms/topic/6428/manejo-de-excepciones

Capítulo 27: MessagingCenter

Introducción

Xamarin.Forms tiene un mecanismo de mensajería incorporado para promover el código desacoplado. De esta manera, los modelos de visualización y otros componentes no necesitan conocerse entre sí. Pueden comunicarse por un simple contrato de mensajería.

Hay básicamente dos ingredientes principales para usar el `MessagingCenter` .

Suscribir escuche los mensajes con cierta firma (el contrato) y ejecute el código cuando reciba un mensaje. Un mensaje puede tener múltiples suscriptores.

Enviar enviando un mensaje para que los suscriptores actúen.

Examples

Ejemplo simple

Aquí veremos un ejemplo simple del uso de MessagingCenter en Xamarin.Forms.

Primero, echemos un vistazo a la suscripción a un mensaje. En el modelo `FooMessaging` nos suscribimos a un mensaje que viene de la `MainPage` . El mensaje debe ser "Hola" y cuando lo recibimos, registramos un controlador que establece la propiedad `Greeting` . Por último, `this` significa que la instancia actual de `FooMessaging` está registrando para este mensaje.

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

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

Para enviar un mensaje que active esta funcionalidad, necesitamos tener una página llamada `MainPage` e implementar un código como el de abajo.

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

En nuestra `MainPage` tenemos un botón con un controlador que envía un mensaje. `this` debería ser una instancia de `MainPage` .

Pasando argumentos

También puede pasar argumentos con un mensaje para trabajar.

Usaremos las clasificadas de nuestro ejemplo anterior y las ampliaremos. En la parte receptora, justo detrás de la llamada al método `Subscribe` , agregue el tipo de argumento que espera. Asegúrese también de declarar los argumentos en la firma del controlador.

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

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

Al enviar un mensaje, asegúrese de incluir el valor del argumento. Además, aquí agrega el tipo justo detrás del método de `Send` y agrega el valor del argumento.

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

En este ejemplo, se utiliza una cadena simple, pero también puede usar cualquier otro tipo de objetos (complejos).

Darse de baja

Cuando ya no necesite recibir mensajes, simplemente puede darse de baja. Puedes hacerlo así:

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

Cuando está proporcionando argumentos, debe darse de baja de la firma completa, como esto:

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

Lea MessagingCenter en línea: https://riptutorial.com/es/xamarin-forms/topic/9672/messagingcenter

Capítulo 28: Mostrar alerta

Examples

DisplayAlert

Un cuadro de alerta puede aparecer en una `Xamarin.Forms Page` por el método, `DisplayAlert` . Podemos proporcionar un Título, Cuerpo (Texto para ser alertado) y uno / dos botones de acción. `Page` ofrece dos anulaciones del método `DisplayAlert` .

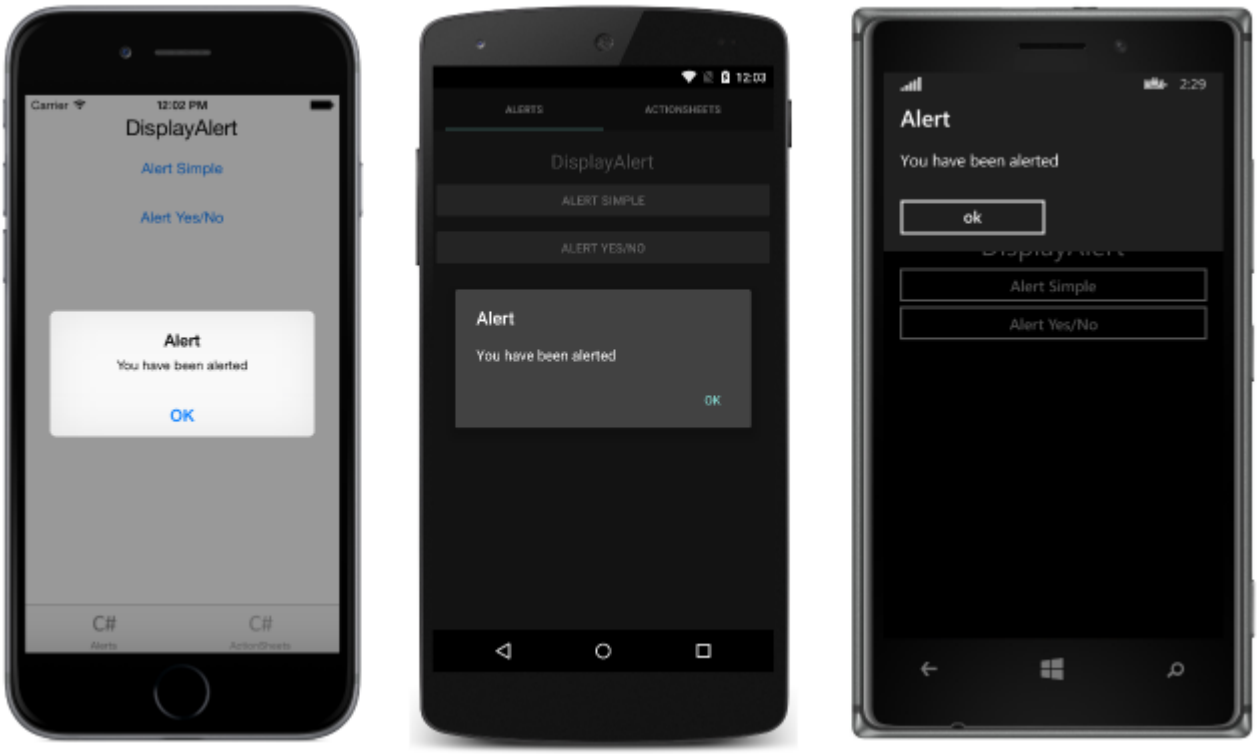
1. `public Task DisplayAlert (String title, String message, String cancel)`

Esta anulación presenta un cuadro de diálogo de alerta para el usuario de la aplicación con un solo botón de cancelación. La alerta se muestra de manera modal y, una vez descartada, el usuario continúa interactuando con la aplicación.

Ejemplo:

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

El fragmento anterior presentará una implementación nativa de Alertas en cada plataforma (`AlertDialog` en Android, `UIAlertView` en iOS, `MessageDialog` en Windows) como se muestra a continuación.



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

Esta anulación presenta un cuadro de diálogo de alerta para el usuario de la aplicación con los

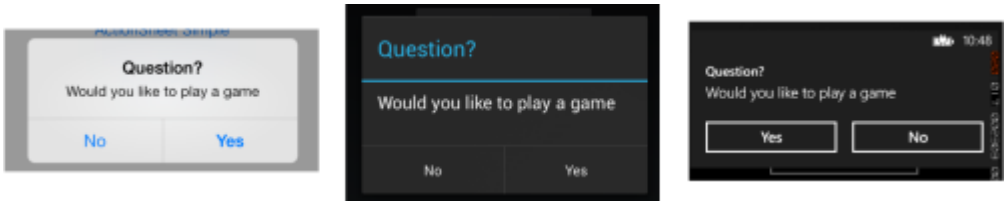
botones Aceptar y Cancelar. Captura la respuesta de un usuario presentando dos botones y devolviendo un valor `boolean` . Para obtener una respuesta de una alerta, suministre texto para ambos botones y espere el método. Después de que el usuario seleccione una de las opciones, la respuesta se devolverá al código.

Ejemplo:

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

Ejemplo 2: (si la Condición es verdadera o falsa, verifique la alerta)

```
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
        }
     }
```



Ejemplo de alerta con un solo botón y acción

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

Aquí obtendremos la acción Ok clic.

Lea Mostrar alerta en línea: https://riptutorial.com/es/xamarin-forms/topic/4883/mostrar-alerta

Capítulo 29: Navegación en Xamarin. Formas

Examples

Flujo de navegación

```
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;
```

```
        }
    }
}
```

Navegación en la página de flujo con XAML

Archivo App.xaml.cs (el archivo App.xaml es predeterminado, así que se omite)

```
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;
        }
    }
}
```

Archivo FirstPage.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="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>
```

En algunos casos, necesita abrir la nueva página no en la navegación actual sino en la global. Por ejemplo, si su página actual contiene el menú inferior, será visible cuando presione la nueva página en la navegación actual. Si necesita que la página se abra sobre todo el contenido visible que oculta el menú inferior y el contenido de otra página actual, debe insertar la nueva página como modal en la navegación global. Consulte la propiedad `App.GlobalNavigation` y el ejemplo a

continuación.

Archivo FirstPage.xaml.cs

```
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);
        }
    }
}
```

Archivo SecondPage.xaml (el archivo xaml.cs es predeterminado, por lo que se omite)

```
<?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>
```

Navegación jerárquica con XAML

De forma predeterminada, el patrón de navegación funciona como una pila de páginas, y llama a las páginas más recientes sobre las páginas anteriores. Necesitará usar el objeto NavigationPage para esto.

Empujando nuevas páginas

```
...
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());
    }
}
...
```

Saltando páginas

Normalmente, el usuario usa el botón Atrás para regresar páginas, pero a veces necesita controlar esto programáticamente, por lo que debe llamar al método **NavigationPage.PopAsync ()** para regresar a la página anterior o **NavigationPage.PopToRootAsync ()** para regresar al principio, tales como ...

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();
    }
}
...
```

Navegación modal con XAML

Las páginas modales se pueden crear de tres maneras:

- Desde el objeto **NavigationPage** para páginas de pantalla completa

- Para Alertas y Notificaciones
- Para ActionSheets que son menús emergentes.

Modales de pantalla completa

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

Alertas / Confirmaciones y Notificaciones

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

Hojas de acción

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

Página raíz de detalles maestros

```
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;
    }
}
```

Detalle maestro de navegación

El siguiente código muestra cómo realizar una navegación asíncrona cuando la aplicación se encuentra en un contexto MasterDetailPage.

```
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;
    }
```

Lea Navegación en Xamarin. Formas en línea: https://riptutorial.com/es/xamarin-forms/topic/1571/navegacion-en-xamarin--formas

Capítulo 30: Navegación en Xamarin. Formas

Observaciones

La navegación en Xamarin.Forms se basa en dos patrones de navegación principales: jerárquico y modal.

El patrón jerárquico permite al usuario moverse hacia abajo en una pila de páginas y regresar presionando el botón "atrás" / "arriba".

El patrón modal es una página de interrupción que requiere una acción específica del usuario, pero normalmente se puede cancelar presionando el botón de cancelar. Algunos ejemplos son notificaciones, alertas, cuadros de diálogo y páginas de registro / edición.

Examples

Usando INavigation desde el modelo de vista

El primer paso es crear una interfaz de navegación que usaremos en el modelo de vista:

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

En el método de `Initialize` , uso mi asignador personalizado donde mantengo la colección de tipos de páginas con claves asociadas.

```
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;
    }
}
```

Enumerar con páginas:

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

Archivo `App.cs` :

```
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
}
```

En el mapeador asocié el tipo de alguna página con el valor enum.

Implementación de `IViewNavigationService` :

```
[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.");
            }
        }
    }
```

Obtengo el tipo de página en la que el usuario desea navegar y crear su instancia utilizando la reflexión.

Y luego podría usar el servicio de navegación en el modelo de vista:

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

Lea Navegación en Xamarin. Formas en línea: https://riptutorial.com/es/xamarin-forms/topic/2507/navegacion-en-xamarin--formas

Capítulo 31: Notificaciones push

Observaciones

No hay una manera uniforme de manejar las notificaciones push en Xamarin Forms, ya que la implementación depende en gran medida de las características y eventos específicos de la plataforma. Por lo tanto, siempre será necesario el código específico de la plataforma.

Sin embargo, al usar `DependencyService` puede compartir la mayor cantidad de código posible. También hay un complemento diseñado para esto por rdelrosario, que se puede encontrar en su GitHub .

El código y las capturas de pantalla se tomaron de una serie de blogs de Gerald Versluis que explica el proceso con más detalle.

Examples

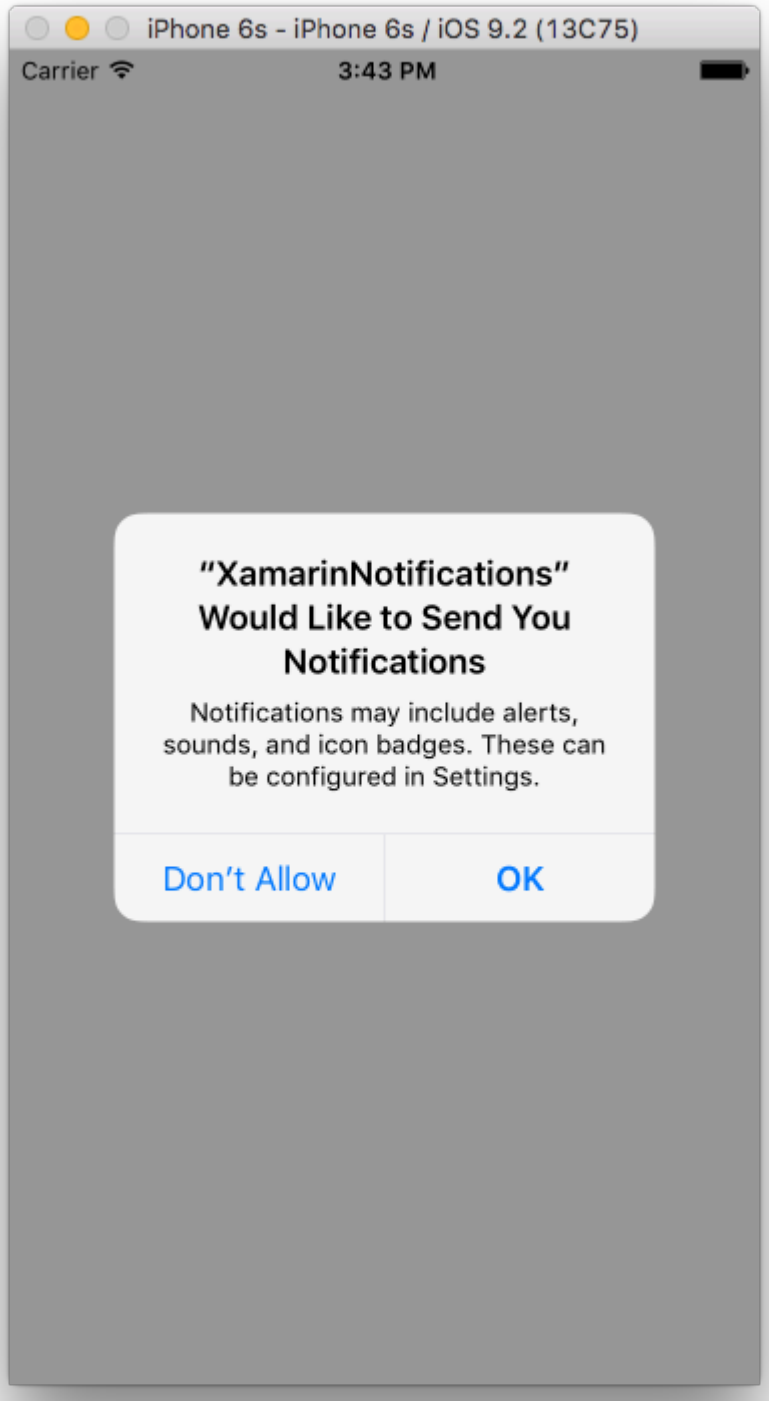
Notificaciones push para iOS con Azure

Para iniciar el registro de notificaciones push, debe ejecutar el siguiente código.

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

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

Este código puede ejecutarse directamente cuando la aplicación se inicia en `FinishedLaunching` en el archivo `AppDelegate.cs` . O puede hacerlo siempre que un usuario decida que desea habilitar las notificaciones push.

La ejecución de este código activará una alerta para avisar al usuario si acepta que la aplicación puede enviarles notificaciones. ¡Así también implementar un escenario donde el usuario lo niegue!

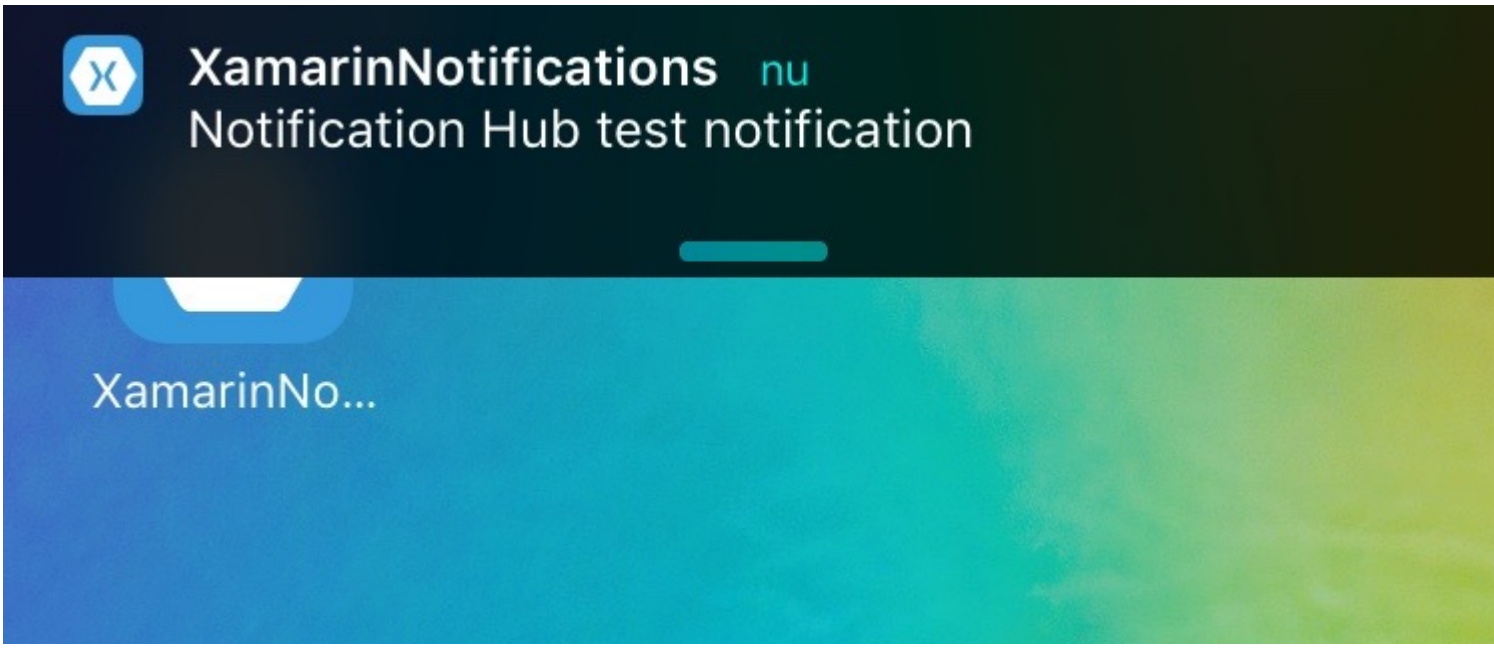


Estos son los eventos que necesitan implementación para implementar notificaciones push en iOS. Puedes encontrarlos en el archivo `AppDelegate.cs` .

```
// 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();
}
```

Cuando se recibe una notificación, esto es lo que parece.



Notificaciones push para Android con Azure

La implementación en Android es un poco más de trabajo y requiere la implementación de un `Service` específico.

Primero, verifiquemos si nuestro dispositivo es capaz de recibir notificaciones push, y si es así, regístrelo con Google. Esto se puede hacer con este código en nuestro archivo `MainActivity.cs` .

```
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());
}
```

Los ID de remitente se pueden encontrar en el código que se encuentra debajo y es el número de proyecto que obtiene del panel de desarrolladores de Google para poder enviar mensajes de inserción.

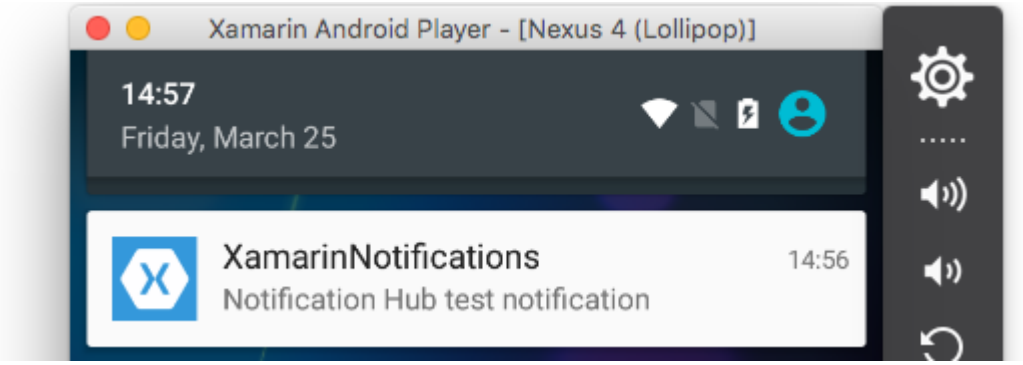
```
using Android.App;
using Android.Content;
using Gcm.Client;
using Java.Lang;
using System;
using WindowsAzure.Messaging;
using XamarinNotifications.Helpers;

// 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);
        }
    }
}
```

Una notificación de muestra en Android se ve así.

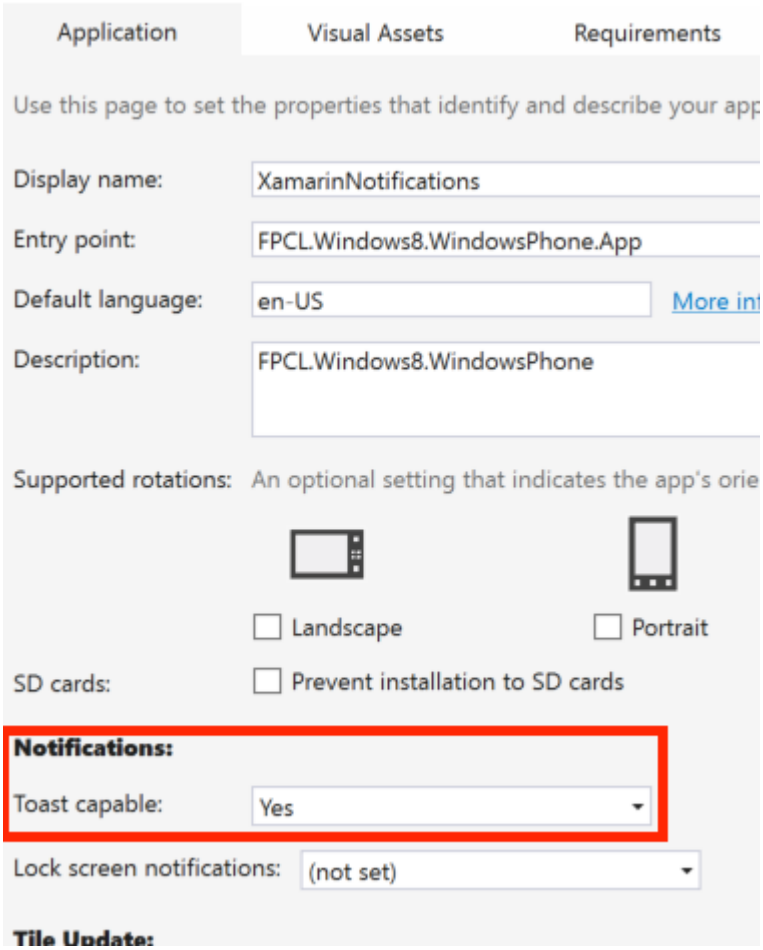


Notificaciones push para Windows Phone con Azure

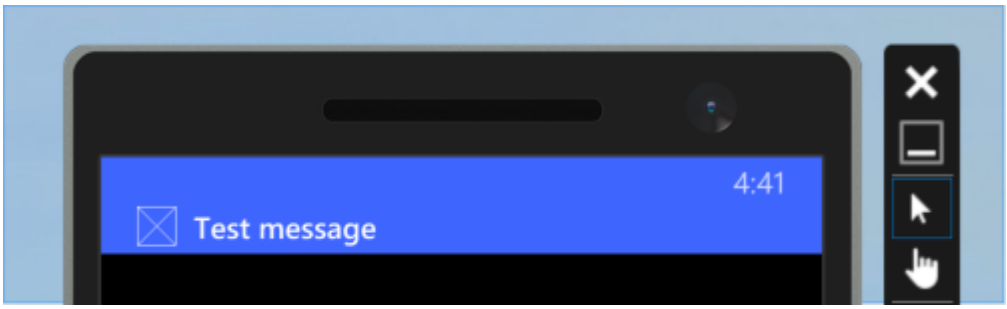
En Windows Phone, se debe implementar algo como el código que se encuentra debajo para comenzar a trabajar con las notificaciones push. Esto se puede encontrar en el archivo `App.xaml.cs`

```
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
}
```

Tampoco olvide habilitar las capacidades en el archivo `Package.appxmanifest` .



Una muestra de notificación de inserción puede verse así:



Lea Notificaciones push en línea: https://riptutorial.com/es/xamarin-forms/topic/5042/notificaciones-push

Capítulo 32: Notificaciones push

Observaciones

Servicio de notificación simple de AWS:

Punto final : el punto final puede ser un teléfono, una dirección de correo electrónico o lo que sea, es lo que AWS SNS puede devolver con una notificación.

Tema : esencialmente un grupo que contiene todos sus puntos finales

Suscribirse - Usted registra su teléfono / cliente para recibir notificaciones

Lingo genérico de notificaciones push:

APNS - Apple Push Notification Service. Apple es el único que puede enviar notificaciones push. Es por esto que proveemos nuestra aplicación con el certificado apropiado. Proporcionamos a AWS SNS el certificado que Apple nos proporciona para autorizar a SNS a enviar una notificación a APNS en nuestro nombre.

GCM : Google Cloud Messaging es muy similar a APNS. Google es el único que puede enviar directamente notificaciones push. Entonces, primero registramos nuestra aplicación en GCM y entregamos nuestro token a AWS SNS. SNS maneja todas las cosas complejas relacionadas con GCM y el envío de datos.

Examples

Ejemplo de iOS

1. Necesitarás un dispositivo de desarrollo.
2. Vaya a su cuenta de desarrollador de Apple y cree un perfil de aprovisionamiento con las notificaciones push habilitadas
3. Necesitará algún tipo de manera de notificar a su teléfono (AWS, Azure, etc.). **Usaremos AWS aquí.**

```
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"

            });

        }

    }
```

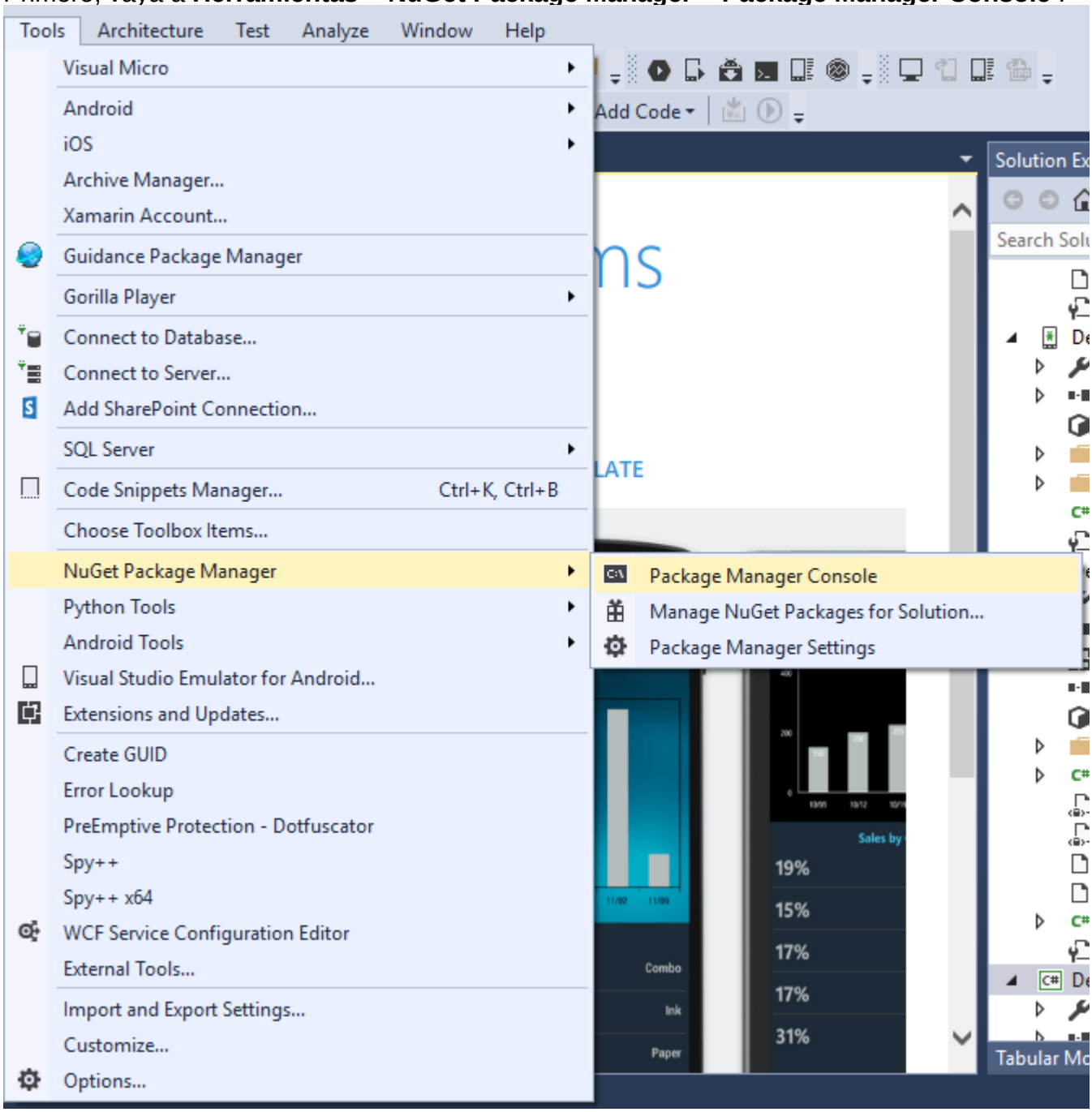
Lea Notificaciones push en línea: https://riptutorial.com/es/xamarin-forms/topic/5998/notificaciones-push

Capítulo 33: OAuth2

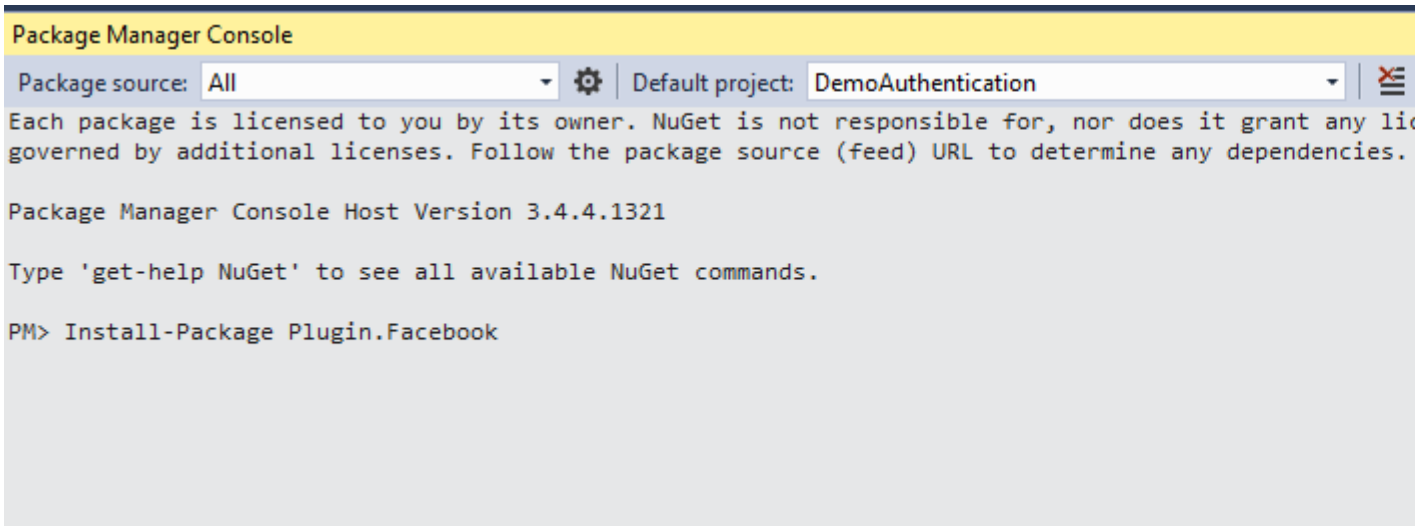
Examples

Autenticación utilizando el complemento

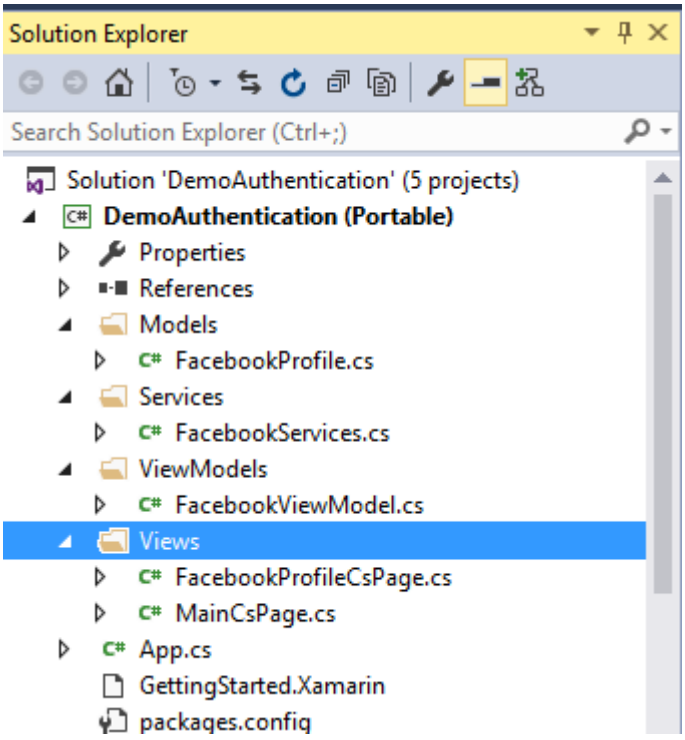
1. Primero, vaya a **Herramientas** > **NuGet Package Manager** > **Package Manager Console** .



2. Ingrese este comando " **Install-Package Plugin.Facebook** " en la Consola del **administrador** de paquetes.



3. Ahora todo el archivo se crea automáticamente.



Video : Ingresa con Facebook en Xamarin Forms

Otra autenticación mediante el uso de Plugin. Coloque el comando en la Consola del administrador de paquetes como se muestra en el Paso 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

Lea OAuth2 en línea: https://riptutorial.com/es/xamarin-forms/topic/8828/oauth2

Capítulo 34: Renderizadores personalizados

Examples

Procesador personalizado para ListView

Los procesadores personalizados permiten a los desarrolladores personalizar la apariencia y el comportamiento de los controles de Xamarin.Forms en cada plataforma. Los desarrolladores podrían usar las características de los controles nativos.

Por ejemplo, necesitamos desactivar el desplazamiento en `ListView` . En iOS, `ListView` es desplazable incluso si todos los elementos se colocan en la pantalla y el usuario no debería poder desplazarse por la lista. `Xamarin.Forms.ListView` no administra dicha configuración. En este caso, un renderer viene a ayudar.

En primer lugar, debemos crear un control personalizado en el proyecto PCL, que declarará alguna propiedad enlazable requerida:

```
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); }
    }
}
```

El siguiente paso será crear un renderizador para cada plataforma.

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;
```

```
        }
    }
}
```

Y Android (la lista de Android no tiene desplazamiento si todos los elementos se colocan en la pantalla, por lo que no deshabilitaremos el desplazamiento, pero aún podemos usar propiedades nativas):

```
[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` propiedad del `Element` del procesador es mi control `SuperListView` del proyecto PCL.

`Control` propiedad de `Control` del renderizador es control nativo. `Android.Widget.ListView` para Android y `UIKit.UITableView` para iOS.

Y como lo usaremos en `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>
```

Archivo `.cs` de la página:

```
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);
    }
}
```

Renderizador personalizado para BoxView

La ayuda personalizada del Renderizador permite agregar nuevas propiedades y representarlas de manera diferente en una plataforma nativa que, de otro modo, no se puede hacer a través del código compartido. En este ejemplo, agregaremos el radio y la sombra a una vista de caja.

En primer lugar, debemos crear un control personalizado en el proyecto PCL, que declarará alguna propiedad enlazable requerida:

```
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); }
        }
    }
}
```

El siguiente paso será crear un renderizador para cada plataforma.

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);
        }
    }

 }
}
```

Nuevamente puedes personalizar como quieras dentro del método de sorteo.

Y lo mismo para 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);
        }
}
```

}

El XAML:

Primero hacemos referencia a nuestro control con el espacio de nombres que definimos anteriormente.

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

Luego usamos el Control de la siguiente manera y usamos las propiedades definidas al principio:

```
<Controls:ExtendedBoxView
    x:Name="search_boxview"
```

```
    Color="#444"
    BorderRadius="5"
    HorizontalOptions="CenterAndExpand"
    />
```

Accediendo al renderizador desde un proyecto nativo

```
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
```

Etiqueta redondeada con un renderizador personalizado para Frame (partes PCL e iOS)

Primer paso: parte PCL

```
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); }
        }
    }
}
```

Segundo paso: parte de iOS

```
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;
            }
        }
    }
}
```

Tercer paso: código XAML para llamar a un ExtendedFrame

Si quieres usarlo en una parte XAML, no olvides escribir esto:

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

después

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

Ahora, puedes usar el ExtendedFrame así:

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

BoxView redondeado con color de fondo seleccionable

Primer paso: parte PCL

```
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);
        }
    }
}
```

Segundo paso: parte del droide

```
[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);
        }
    }
}
```

Tercer paso: parte de iOS

```
[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();
            }
        }
    }
}
```

Lea Renderizadores personalizados en línea: https://riptutorial.com/es/xamarin-forms/topic/2949/renderizadores-personalizados

Capítulo 35: Selector de contactos - Formas Xamarin (Android y iOS)

Observaciones

Contact Picker XF (Android y 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);
        }
    }
}
```

Lea Selector de contactos - Formas Xamarin (Android y iOS) en línea: https://riptutorial.com/es/xamarin-forms/topic/6659/selector-de-contactos---formas-xamarin--android-y-ios-

Capítulo 36: Servicios de dependencia

Observaciones

Acceda a la API específica de la plataforma desde PCL o proyecto compartido.

Examples

Acceso a la cámara y la galería.

(Código de acceso) [https://github.com/vDoers/vDoersCameraAccess]

Lea Servicios de dependencia en línea: https://riptutorial.com/es/xamarin-forms/topic/6127/servicios-de-dependencia

Capítulo 37: Trabajando con Mapas

Observaciones

Si va a ejecutar su proyecto en otra computadora, tendrá que generar una nueva clave API para él, porque las huellas dactilares SHA-1 no coincidirán con las diferentes máquinas de compilación.

Puede explorar el proyecto, que se describe en el ejemplo *Agregar un mapa en Xamarin.Forms* aquí

Examples

Añadiendo un mapa en Xamarin.Forms (Xamarin Studio)

Simplemente puede usar las API de mapas nativos en cada plataforma con Xamarin Forms. Todo lo que necesita es descargar el paquete *Xamarin.Forms.Maps* de nuget e instalarlo en cada proyecto (incluido el proyecto PCL).

Inicialización de mapas

En primer lugar, debe agregar este código a sus proyectos específicos de la plataforma. Para hacer esto, debe agregar la `Xamarin.FormsMaps.Init` método `Xamarin.FormsMaps.Init` , como en los ejemplos a continuación.

---

proyecto iOS

Archivo 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);
    }
}
```

Proyecto de android

Archivo 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());
    }
}
```

Configuración de la plataforma

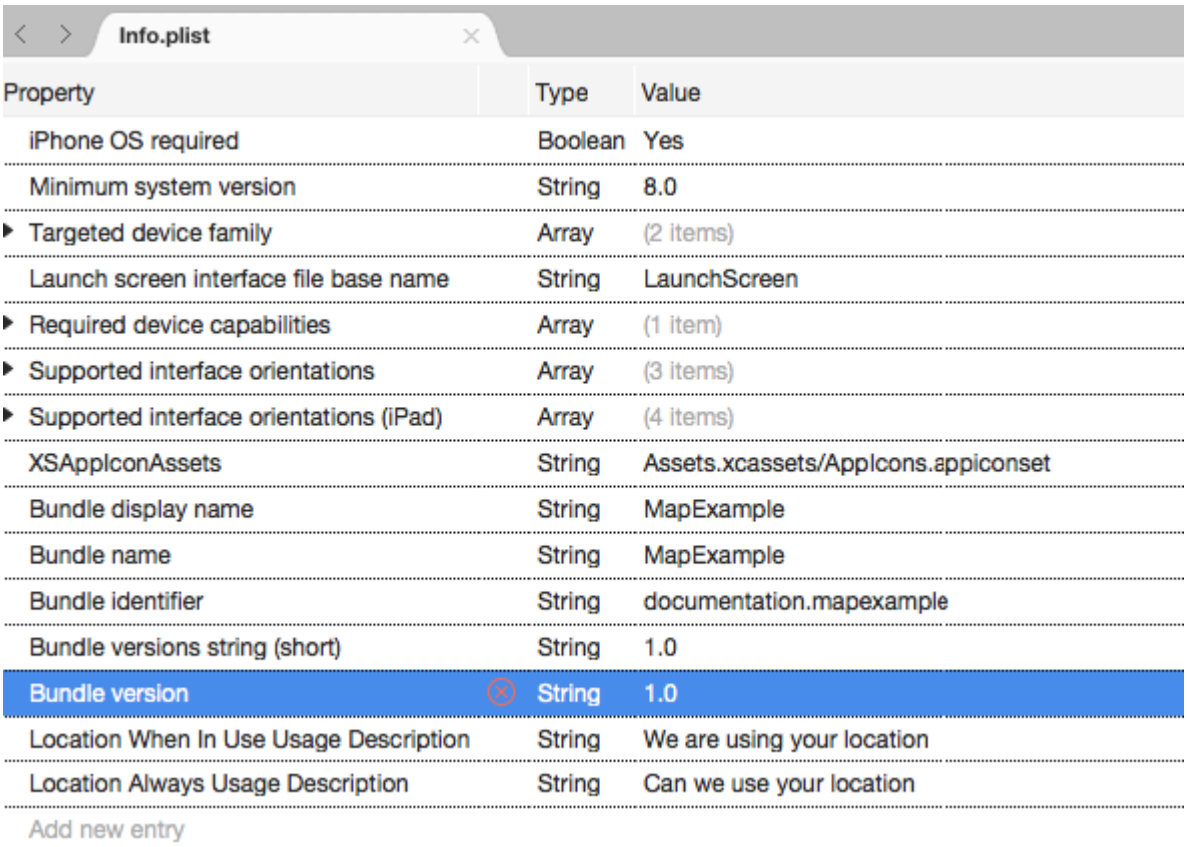
Se requieren pasos de configuración adicionales en algunas plataformas antes de que se muestre el mapa.

---

proyecto iOS

En el proyecto iOS, solo tiene que agregar 2 entradas a su archivo *Info.plist* :

- `NSLocationWhenInUseUsageDescription` *cadena* con valor `We are using your location`
- `NSLocationAlwaysUsageDescription` *cadena* con valor `Can we use your location`



Proyecto de android

Para usar Google Maps, debe generar una clave API y agregarla a su proyecto. Siga las instrucciones a continuación para obtener esta clave:

1. (Opcional) Encuentre la ubicación de la herramienta keytool (la `/System/Library/Frameworks/JavaVM.framework/Versions/Current/Commands` predeterminada es `/System/Library/Frameworks/JavaVM.framework/Versions/Current/Commands`)

2. (Opcional) Abra el terminal y vaya a la ubicación de su herramienta de claves:

   ```
   cd /System/Library/Frameworks/JavaVM.framework/Versions/Current/Commands
   ```

3. Ejecute el siguiente comando keytool:

   ```
   keytool -list -v -keystore "/Users/[USERNAME]/.local/share/Xamarin/Mono for
   Android/debug.keystore" -alias androiddebugkey -storepass android -keypass android
   ```

   Donde [NOMBRE DE USUARIO] es, obviamente, su carpeta de usuario actual. Debería obtener algo similar a esto en la salida:

   ```
   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
   ```

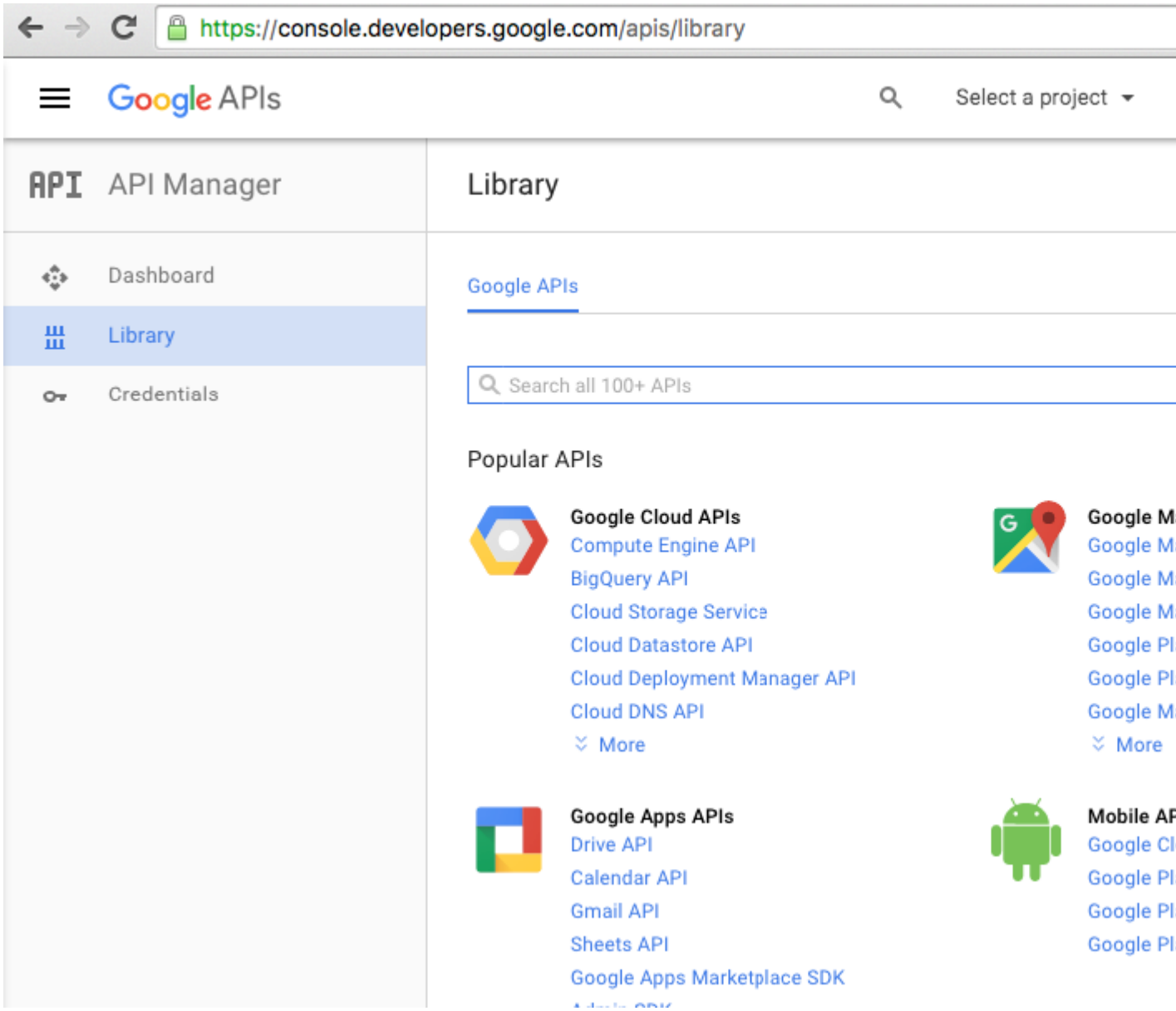
```
Serial number: 4b5ac934
Valid from: Thu Jun 30 10:22:00 EEST 2016 until: Sat Jun 23 10:22:00 EEST 2046
Certificate fingerprints:
         MD5:  4E:49:A7:14:99:D6:AB:9F:AA:C7:07:E2:6A:1A:1D:CA
         SHA1: 57:A1:E5:23:CE:49:2F:17:8D:8A:EA:87:65:44:C1:DD:1C:DA:51:95
         SHA256:
70:E1:F3:5B:95:69:36:4A:82:A9:62:F3:67:B6:73:A4:DD:92:95:51:44:E3:4C:3D:9E:ED:99:03:09:9F:90:3F

         Signature algorithm name: SHA256withRSA
         Version: 3
```

4. Todo lo que necesitamos en esta salida es la huella digital del certificado SHA1. En nuestro caso es igual a esto:

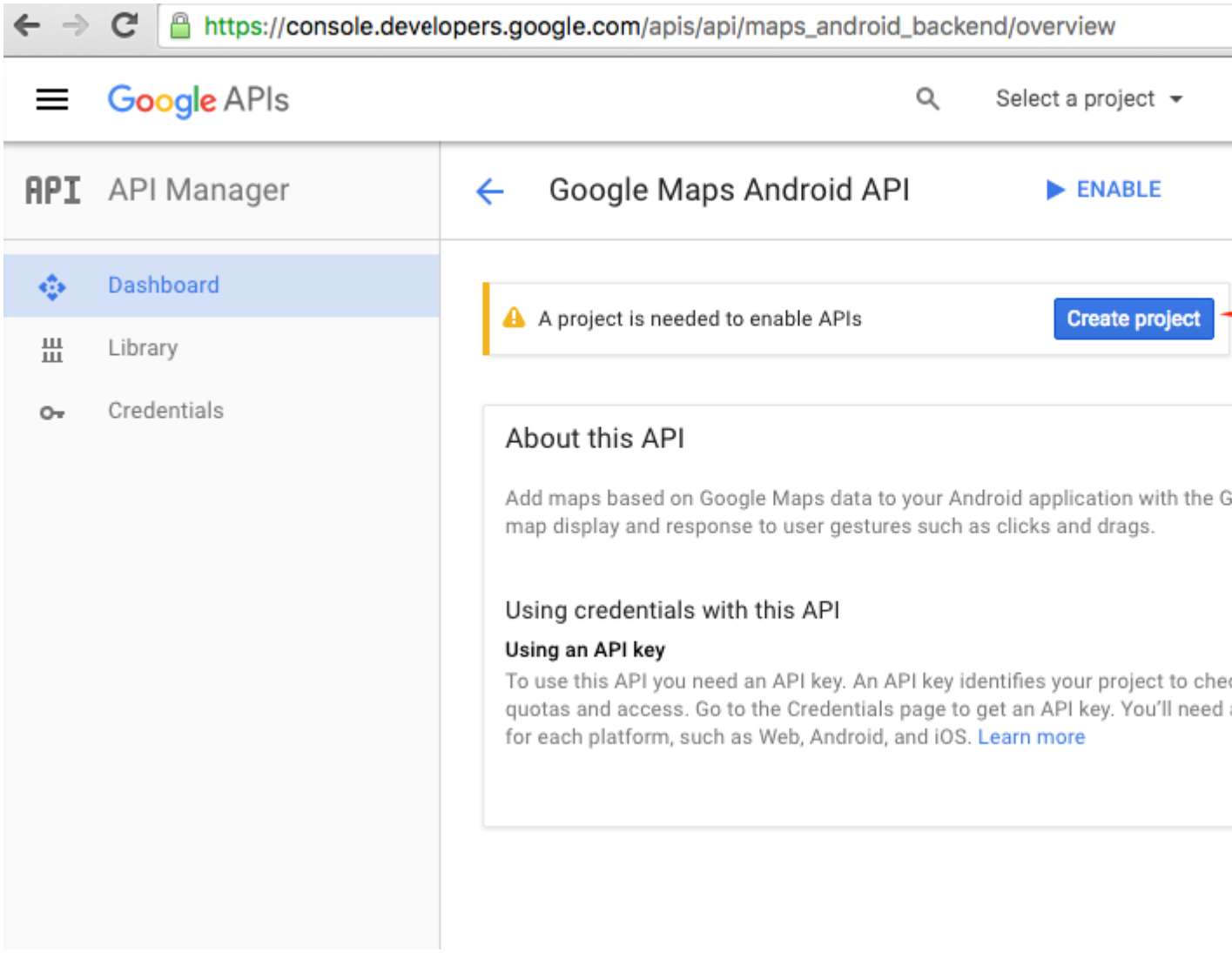
```
57:A1:E5:23:CE:49:2F:17:8D:8A:EA:87:65:44:C1:DD:1C:DA:51:95
```

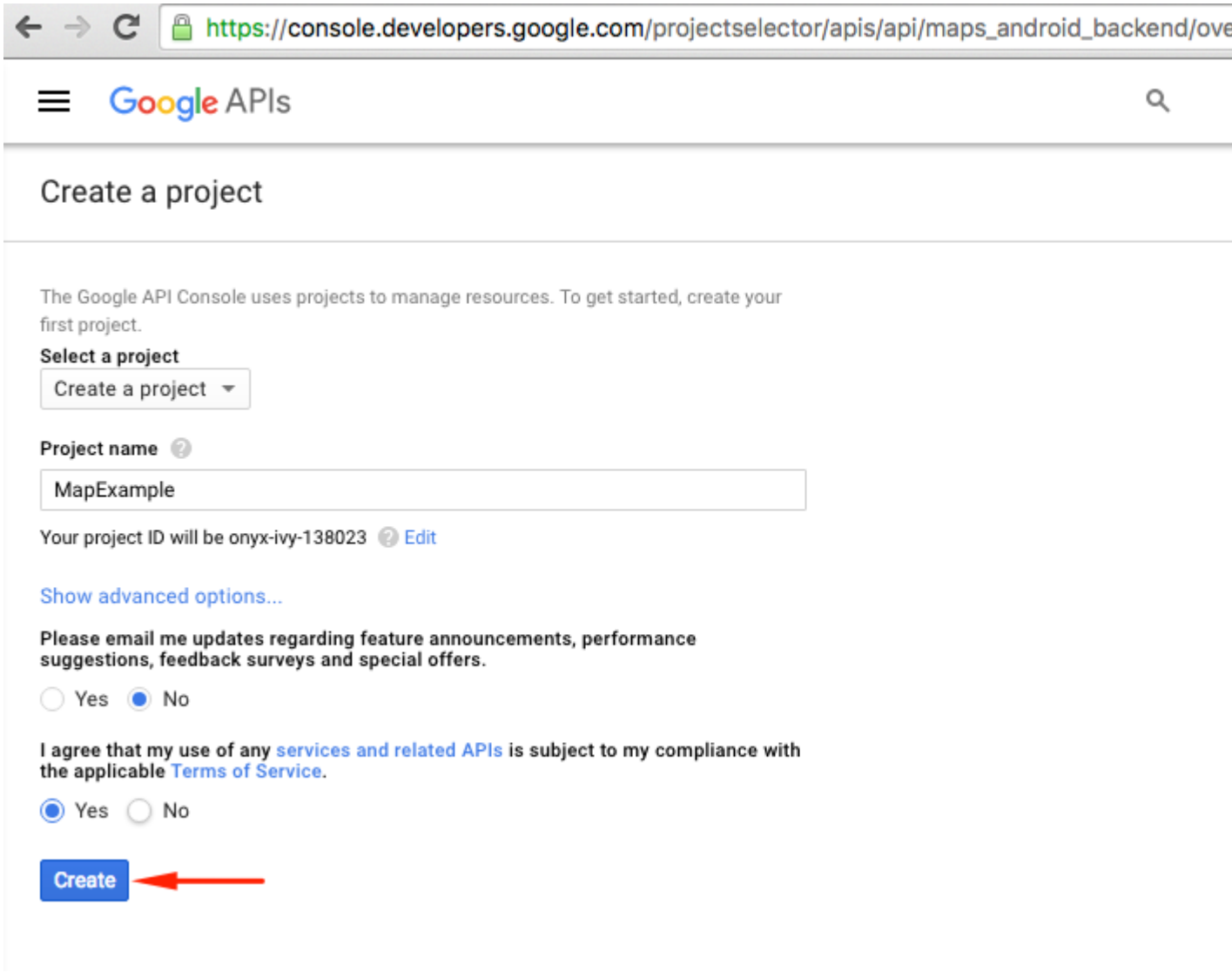
   Copia o guarda en alguna parte esta clave. Lo necesitaremos más adelante.

5. Vaya a la Consola de desarrolladores de Google , en nuestro caso debemos agregar la API de Android de Google Maps , así que elíjala

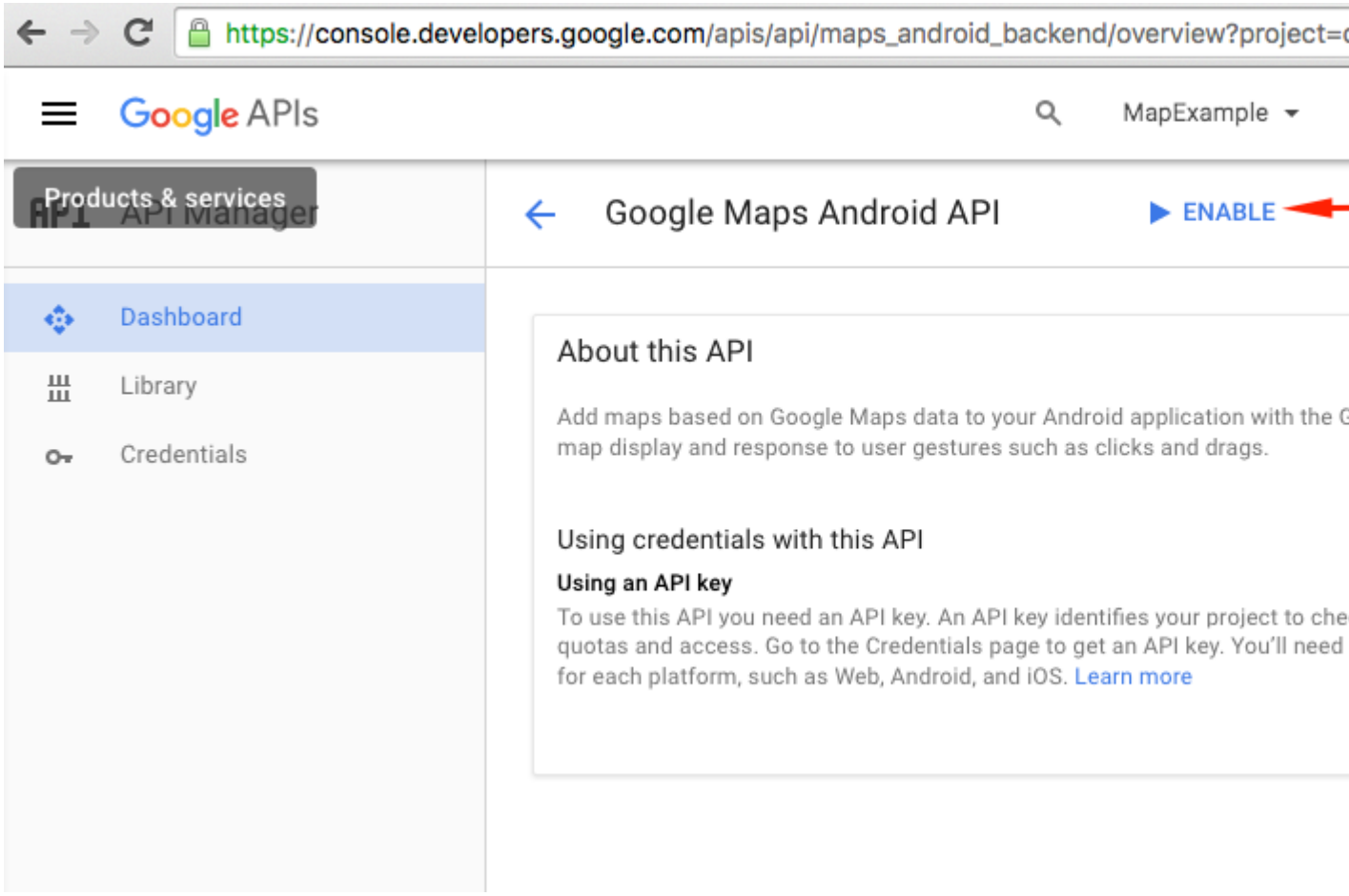


6. Google le pedirá que cree un proyecto para habilitar las API, siga este consejo y cree el proyecto:

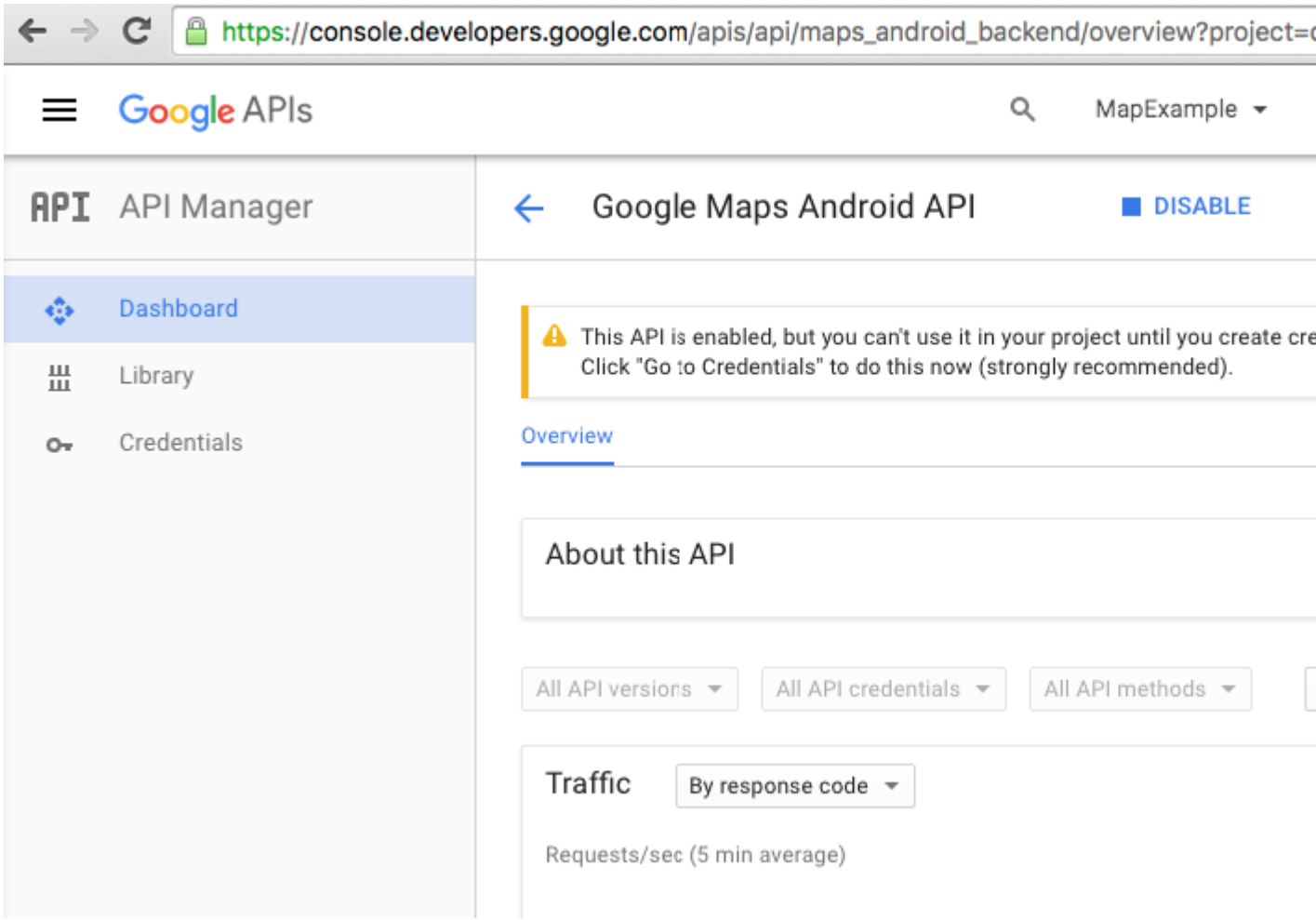




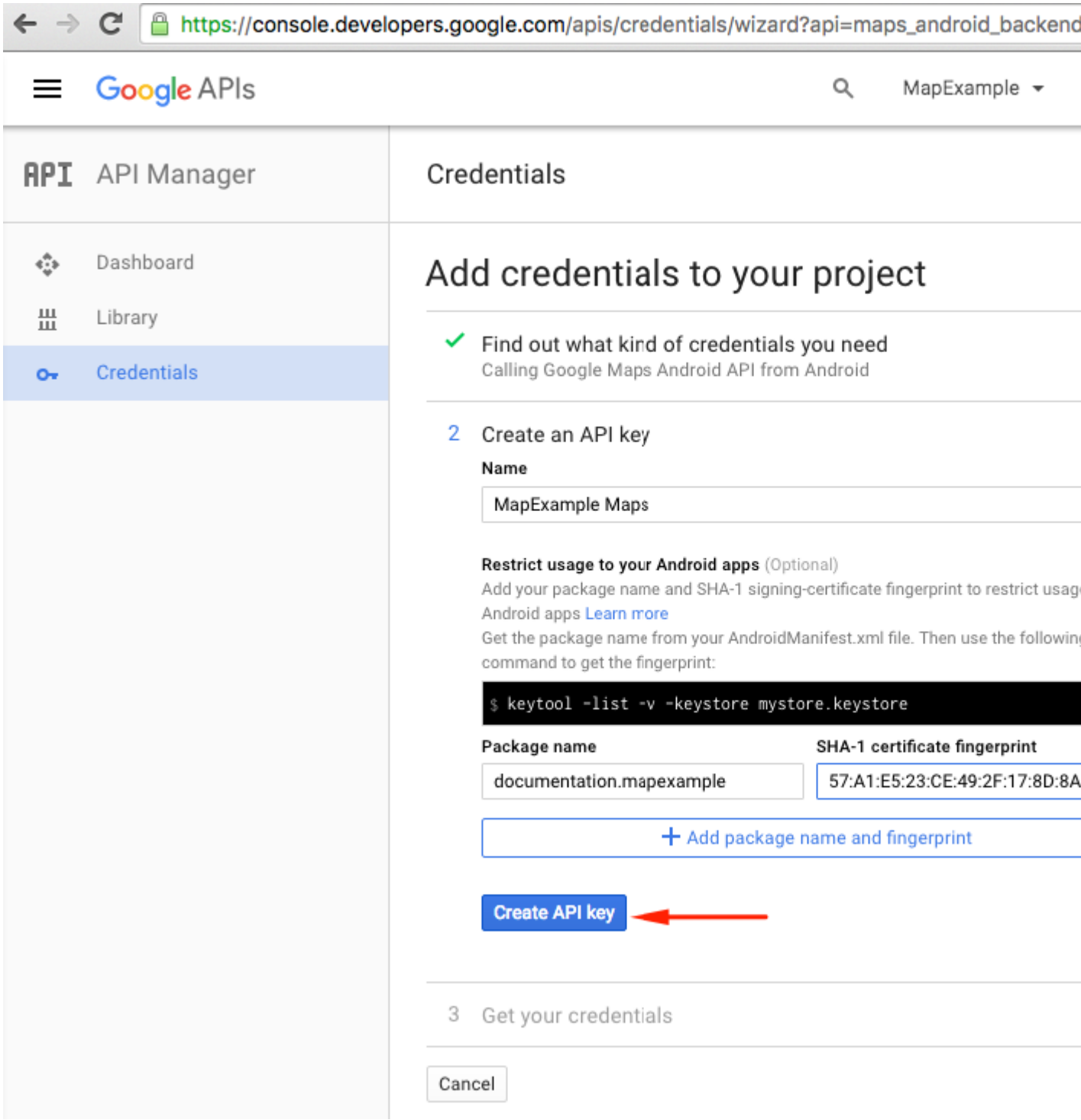
7. Habilite la API de Google Maps para su proyecto:



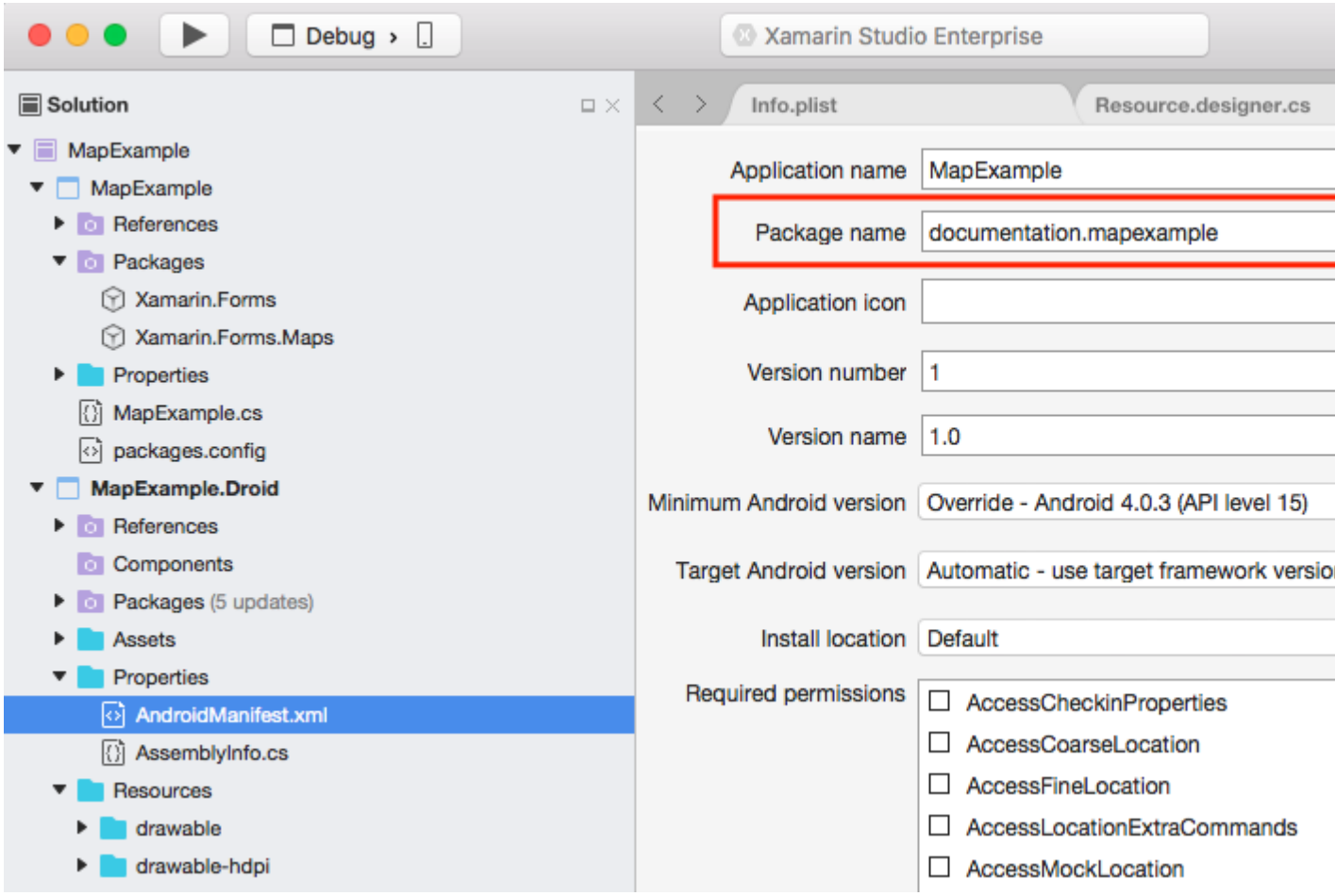
Después de habilitar la API, debe crear credenciales para su aplicación. Siga este consejo:



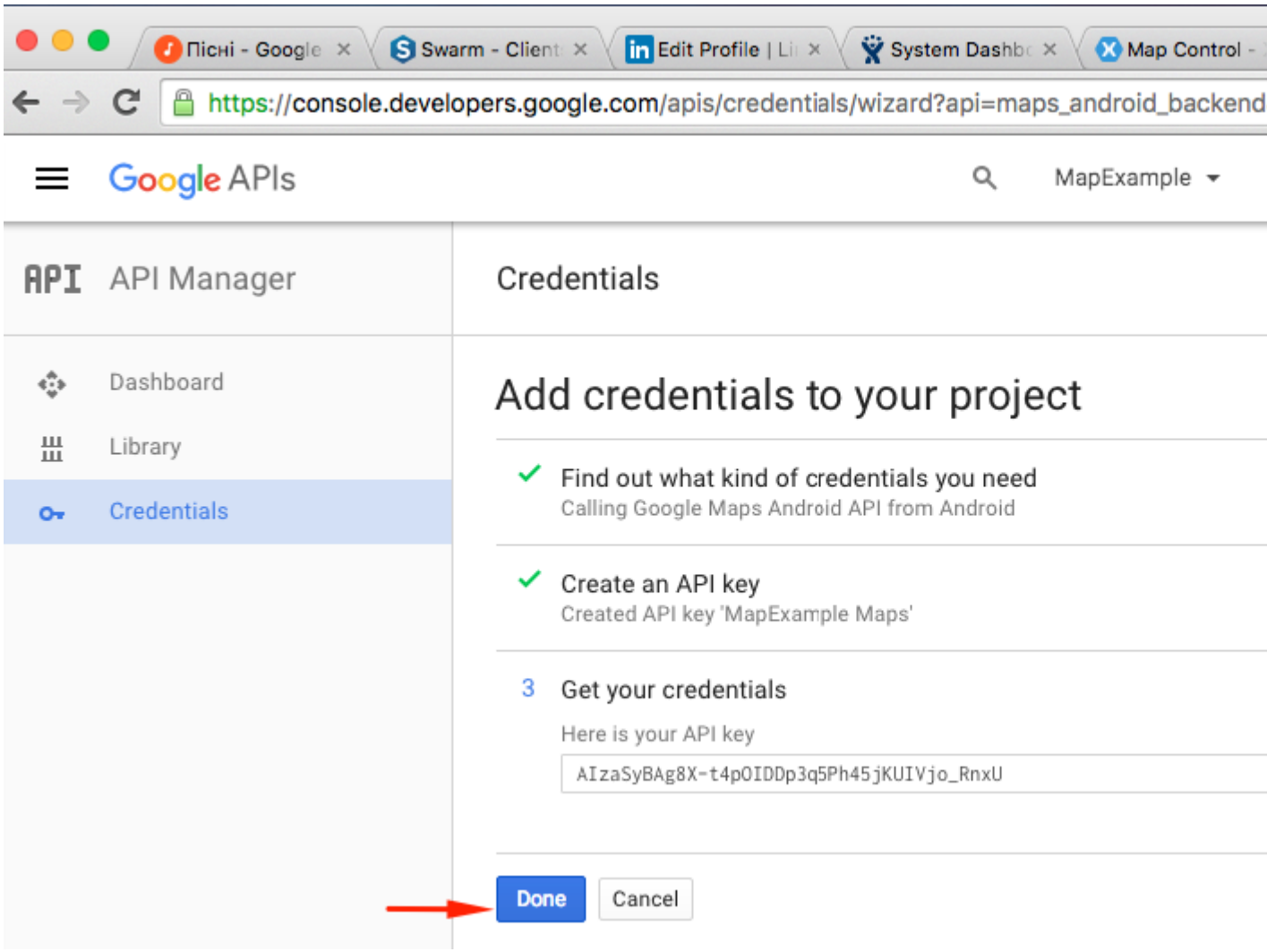
8. En la página siguiente, elija la plataforma Android, toque "¿Qué credenciales necesito?" botón, cree un nombre para su clave API, toque "Agregar nombre de paquete y huella digital", ingrese su nombre de paquete y su huella digital SHA1 en el paso 4 y finalmente cree una clave API:



Para encontrar el nombre de su paquete en Xamarin Studio, vaya a su solución .Droid -> AndroidManifest.xml:



9. Después de la creación, copie la nueva clave API (no olvide presionar el botón "Listo" después) y péguela en su archivo `AndroidManifest.xml` :



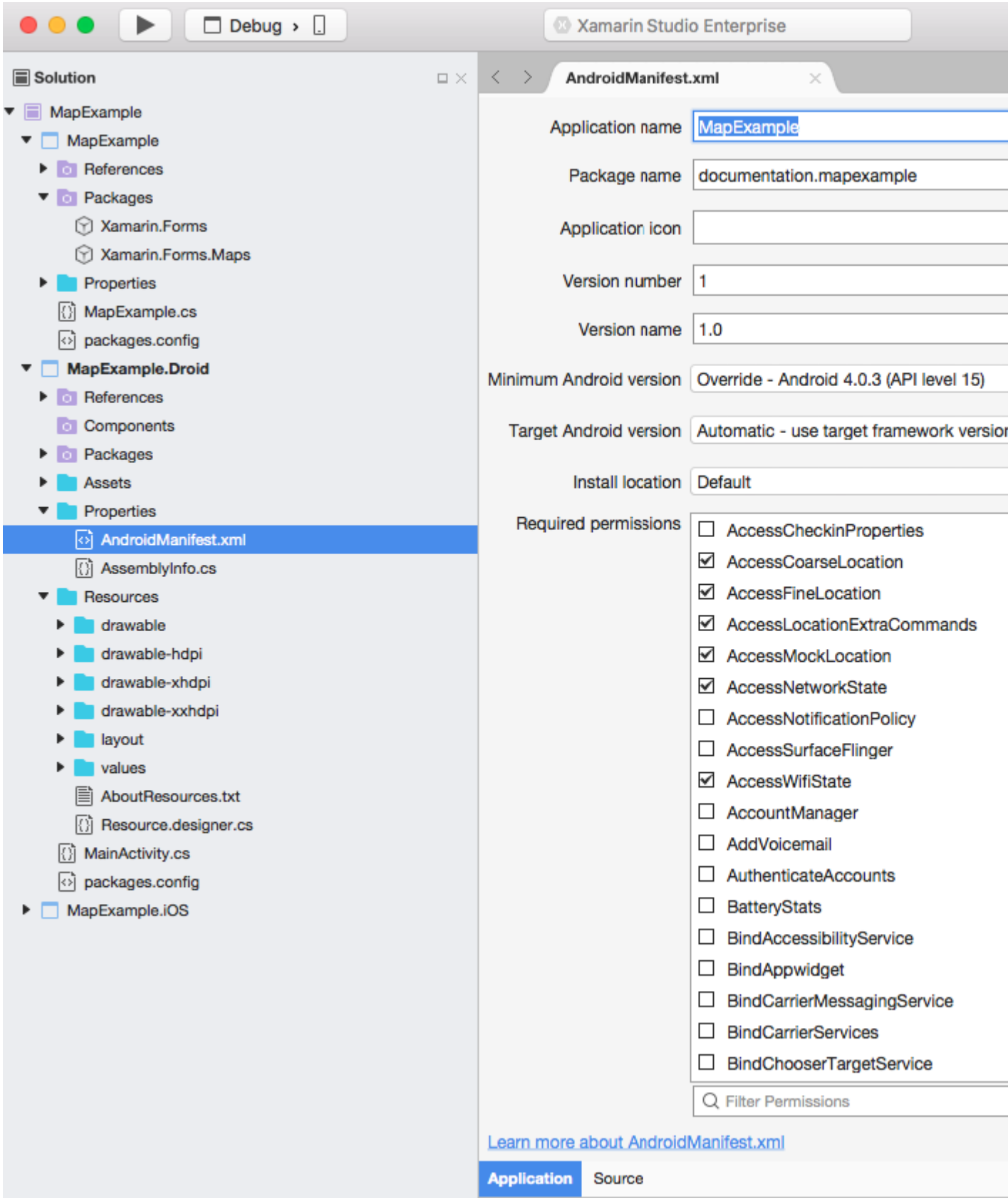
Archivo AndroidManifest.xml

```
<?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>
```

También necesitarás habilitar algunos permisos en tu manifiesto para habilitar algunas características adicionales:

- Acceso a la ubicación aproximada
- Acceso a Ubicación Fina

- Ubicación de acceso Comandos adicionales
- Acceso a la ubicación simulada
- Estado de la red de acceso
- Acceso Wifi Estado
- Internet



Aunque, los dos últimos permisos son necesarios para descargar datos de Maps. Lea

acerca de los permisos de Android para aprender más. Eso es todos los pasos para la configuración de Android.

> *Nota* : si desea ejecutar su aplicación en el simulador de Android, debe instalar los servicios de Google Play. Siga este tutorial para instalar Play Services en el reproductor de Android Xamarin. Si no puede encontrar la actualización de Google Play Services después de la instalación de Play Store, puede actualizarla directamente desde su aplicación, donde depende de los servicios de mapas.

Añadiendo un mapa

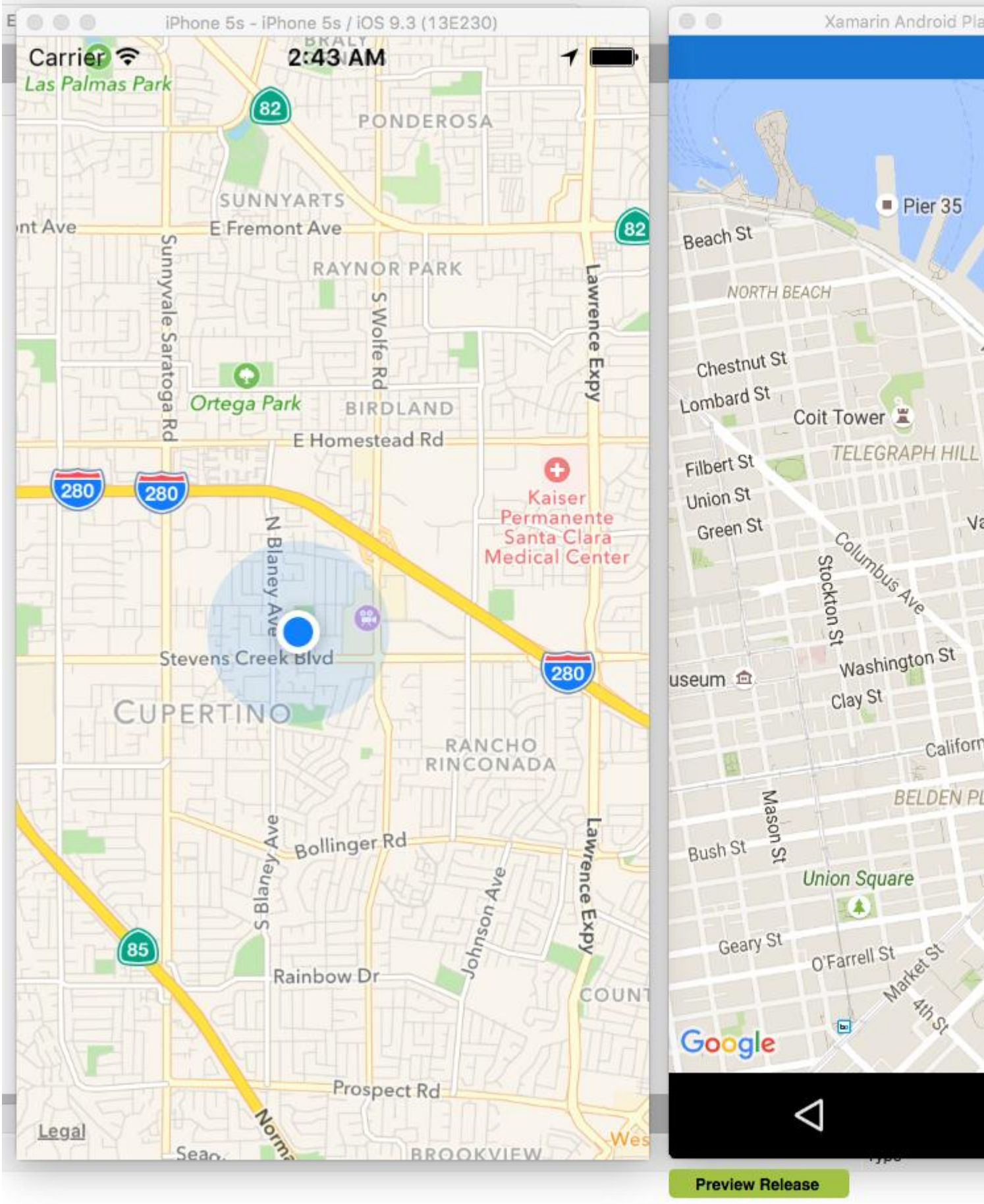
Agregar una vista de mapa a su proyecto multiplataforma es bastante simple. Aquí hay un ejemplo de cómo puede hacerlo (estoy usando el proyecto PCL sin XAML).

Proyecto PCL

Archivo MapExample.cs

```
public class App : Application
{
    public App()
    {
        var map = new Map();
        map.IsShowingUser = true;

        var rootPage = new ContentPage();
        rootPage.Content = map;

        MainPage = rootPage;
    }
}
```

Eso es todo. Ahora, si ejecuta su aplicación en iOS o Android, le mostrará la vista del mapa:



Lea Trabajando con Mapas en línea: https://riptutorial.com/es/xamarin-forms/topic/3917/trabajando-con-mapas

Capítulo 38: Trabajar con bases de datos locales

Examples

Usando SQLite.NET en un proyecto compartido

SQLite.NET es una biblioteca de código abierto que permite agregar soporte de bases de datos locales utilizando `SQLite` versión 3 en un proyecto `Xamarin.Forms` .

Los pasos a continuación `Xamarin.Forms` cómo incluir este componente en un `Xamarin.Forms` compartido de `Xamarin.Forms` :

1. Descargue la última versión de la clase SQLite.cs y agréguela al Proyecto Compartido.

2. Cada tabla que se incluirá en la base de datos debe modelarse como una clase en el Proyecto Compartido. Una tabla se define agregando al menos dos atributos en la clase: `Table` (para la clase) y `PrimaryKey` (para una propiedad).

Para este ejemplo, se agrega una nueva clase llamada `Song` al proyecto compartido, que se define de la siguiente manera:

```
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. A continuación, agregue una nueva clase llamada `Database` , que hereda de la clase `SQLiteConnection` (incluida en SQLite.cs). En esta nueva clase, se define el código para el acceso a la base de datos, la creación de tablas y las operaciones de CRUD para cada tabla. El código de muestra se muestra a continuación:

```
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. Como pudo ver en el paso anterior, el constructor de nuestra clase de `Database` de `Database` incluye un parámetro de `path` , que representa la ubicación del archivo que almacena el archivo de base de datos SQLite. Un objeto de `Database` estática se puede declarar en `App.cs` El `path` es específico de la plataforma:

```
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. Ahora simplemente llame al objeto `DB` través de la clase de `App` cualquier momento que necesite realizar una operación CRUD a la tabla de `Songs` . Por ejemplo, para insertar una nueva `Song` después de que el usuario haya hecho clic en un botón, puede usar el siguiente código:

```
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);
}
```

Trabajar con bases de datos locales utilizando xamarin.forms en visual studio 2015

Ejemplo de SQlite Paso a paso Explicación

1. Los pasos a continuación muestran cómo incluir este componente en un proyecto compartido de Xamarin.Forms: para agregar paquetes en (pcl, Andriod, Windows, Ios) Agregar referencias Haga clic en **Administrar paquetes de Nuget** -> haga clic en Examinar para instalar **SQLite.Net.Core- PCL** , **SQLite Net Extensions una** vez completada la instalación, verifique una vez en las referencias y luego

2. Para agregar Class **Employee.cs** debajo del código

```
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. Para agregar una interfaz ISQLite

```
using SQLite.Net;
           //using SQLite.Net;
           namespace DatabaseEmployeeCreation.SqlLite.ViewModel
           {
               public interface ISQLite
               {
                   SQLiteConnection GetConnection();
               }
           }
```

4. Cree una clase para lógica de base de datos y siga los métodos a continuación.

utilizando SQLite.Net; utilizando System.Collections.Generic; utilizando System.Linq; utilizando Xamarin.Forms; espacio de nombres DatabaseEmployeeCreation.SqlLite.ViewModel {clase pública DatabaseLogic {static object locker = new object (); Base de datos SQLiteConnection;

```
   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. crear un 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. para mostrar EmployeeDetails debajo del código detrás

```
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. Implementar método en Android y ios GetConnection ().

```
    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();
            }
        }
    }
```

Espero que este ejemplo de arriba sea muy fácil de explicar.

Lea Trabajar con bases de datos locales en línea: https://riptutorial.com/es/xamarin-forms/topic/5997/trabajar-con-bases-de-datos-locales

Capítulo 39: Unidad de Pruebas BDD en Xamarin. Formas

Observaciones

- El contenedor / resolución DI que utilizamos internamente en esta biblioteca es Autofac.
- El marco de prueba es NUnit 3x.
- Debería poder usar esta biblioteca con cualquier framework Xamarin.Forms
- Fuente y proyecto de ejemplo disponible aquí.

Examples

Sencillo flujo de especificaciones para probar los comandos y la navegación con NUnit Test Runner

¿Porqué necesitamos esto?

La forma actual de realizar pruebas unitarias en Xamarin.Forms es a través de un corredor de plataforma, por lo que su prueba tendrá que ejecutarse en un entorno ios, android, windows o mac UI: Ejecución de pruebas en el IDE

Xamarin también proporciona pruebas de IU impresionantes con la oferta de Xamarin.TestCloud , pero cuando se desean implementar las prácticas de desarrollo de BDD, y tienen la capacidad de probar ViewModels y Comandos, mientras se ejecutan a bajo costo en una unidad de prueba de corredores localmente o en un servidor de compilación, construido de manera

Desarrollé una biblioteca que permite utilizar Specflow con Xamarin.Forms para implementar fácilmente sus funciones desde las definiciones de Scenarios hasta ViewModel, independientemente de cualquier marco MVVM utilizado para la aplicación (como XLabs , MVVMCross , Prism)

Si eres nuevo en BDD, revisa Specflow out.

Uso:

- Si aún no lo tiene, instale la extensión de estudio visual de flujo de especificaciones desde aquí (o desde su IDE de estudio visual): https://visualstudiogallery.msdn.microsoft.com/c74211e7-cb6e-4dfa-855d-df0ad4a37dd6
- Agregue una biblioteca de clases a su proyecto Xamarin.Forms. Ese es tu proyecto de prueba.

- Agregue el paquete SpecFlow.Xamarin.Forms de nuget a sus proyectos de prueba.
- Agregue una clase a su proyecto de prueba que herede 'TestApp', y registre sus pares de vistas / modelos de vista, así como agregue cualquier registro DI, como se muestra a continuación:

```
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();
    }
}
```

- Agregue una clase de SetupHook a su proyecto de prueba, para agregar los enlaces de Specflow. Deberá iniciar la aplicación de prueba como se indica a continuación, proporcionando la clase que creó anteriormente y el modelo de vista inicial de su aplicación:

```
[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>();
    }
}
```

- Necesitará agregar un bloque catch a sus vistas xamarin.forms codebehind para ignorar el marco de xamarin.forms forzándolo a ejecutar la aplicación ui (algo que no queremos hacer):

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

```
            }
        }
```

- Agregue una función de flujo de especificaciones a su proyecto (usando las plantillas de flujo de especificaciones vs entregadas con la extensión de flujo de especificaciones vs)
- Cree / genere una clase de paso que herede TestStepBase, pasando el parámetro sceneryContext a la base.
- Utilice los servicios de navegación y los ayudantes para navegar, ejecutar comandos y probar sus modelos de vista:

```
  [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);
        }
    }
```

Uso avanzado para MVVM

Para agregar al primer ejemplo, para probar las declaraciones de navegación que se producen dentro de la aplicación, debemos proporcionar un ViewModel con un gancho a la navegación. Lograr esto:

- Agregue el paquete SpecFlow.Xamarin.Forms.IViewModel de nuget a su proyecto PCL Xamarin.Forms
- Implemente la interfaz IViewModel en su ViewModel. Esto simplemente expondrá la propiedad de investigación de Xamarin.Forms:
- `public class MainViewModel : INotifyPropertyChanged, IViewModel.IViewModel { public`

```
INavigation Navigation { get; set; }
```

- El marco de prueba recogerá y gestionará la navegación interna.
- Puede usar cualquier marco MVVM para su aplicación (como XLabs , MVVMCross , Prism , por nombrar algunos. Mientras la interfaz IViewModel esté implementada en su ViewModel, el marco lo recogerá.

Lea Unidad de Pruebas BDD en Xamarin. Formas en línea: https://riptutorial.com/es/xamarin-forms/topic/6172/unidad-de-pruebas-bdd-en-xamarin--formas

Capítulo 40: Usando ListViews

Introducción

Esta documentación detalla cómo usar los diferentes componentes de Xamarin Forms ListView

Examples

Tirar para actualizar en XAML y codificar detrás

Para habilitar Pull to Refresh en un `ListView` en Xamarin, primero debe especificar que `PullToRefresh` está habilitado y luego especificar el nombre del comando que desea invocar cuando se `PullToRefresh ListView` :

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

Lo mismo se puede lograr en el código detrás:

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

Luego, debe especificar qué hace el Comando de `Refresh` en su código detrás:

```
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;
    }
}
```

Lea Usando ListViews en línea: https://riptutorial.com/es/xamarin-forms/topic/9487/usando-listviews

Capítulo 41: Xamarin.Formas Células

Examples

EntryCell

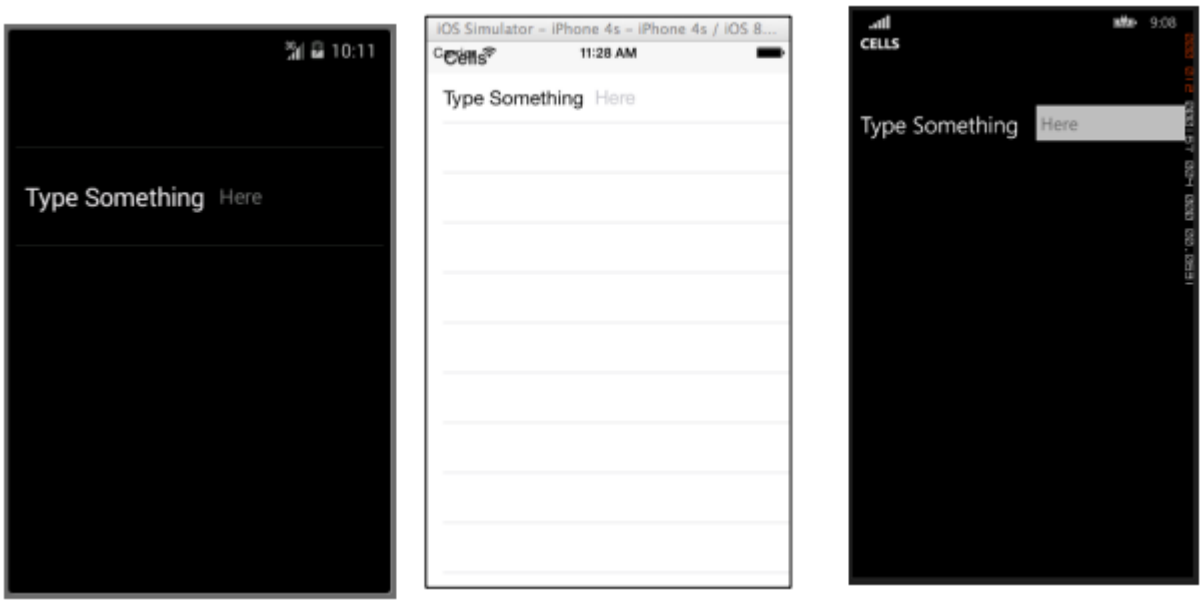
Un EntryCell es una celda que combina las capacidades de una etiqueta y una entrada. El EntryCell puede ser útil en escenarios cuando se construye alguna funcionalidad dentro de su aplicación para recopilar datos del usuario. Se pueden colocar fácilmente en un TableView y ser tratados como un formulario simple.

XAML

```
<EntryCell Label="Type Something"
Placeholder="Here"/>
```

Código

```
var entryCell = new EntryCell {
Label = "Type Something",
Placeholder = "Here"
};
```



SwitchCell

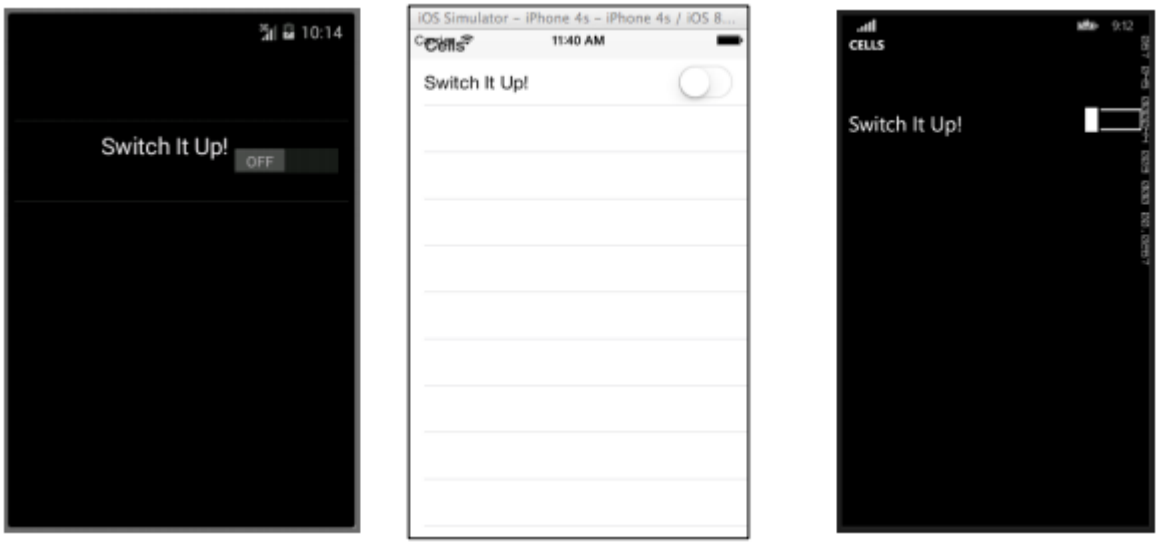
Un SwitchCell es una celda que combina las capacidades de una etiqueta y un interruptor de encendido y apagado. Un SwitchCell puede ser útil para activar y desactivar la funcionalidad, o incluso las preferencias de usuario u opciones de configuración.

XAML

```
<SwitchCell Text="Switch It Up!" />
```

Código

```
var switchCell = new SwitchCell {
Text = "Switch It Up!"
};
```



TextCell

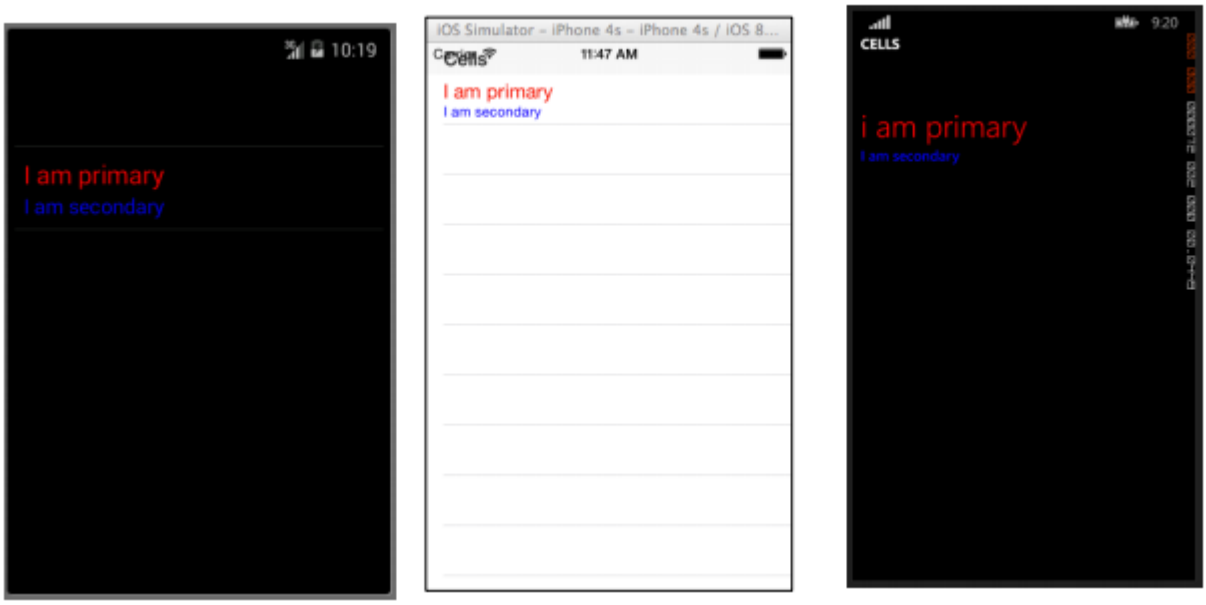
Un TextCell es una celda que tiene dos áreas de texto separadas para mostrar datos. Un TextCell se usa normalmente con fines informativos en los controles TableView y ListView. Las dos áreas de texto están alineadas verticalmente para maximizar el espacio dentro de la celda. Este tipo de celda también se usa comúnmente para mostrar datos jerárquicos, por lo que cuando el usuario toca esta celda, navegará a otra página.

XAML

```
<TextCell Text="I am primary"
TextColor="Red"
Detail="I am secondary"
DetailColor="Blue"/>
```

Código

```
var textCell = new TextCell {
Text = "I am primary",
TextColor = Color.Red,
Detail = "I am secondary",
DetailColor = Color.Blue
};
```



ImageCell

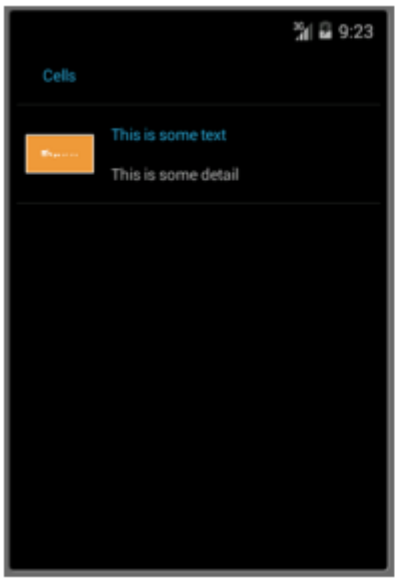
Un ImageCell es exactamente lo que suena. Es una simple celda que contiene solo una imagen. Este control funciona de manera muy similar a un control de imagen normal, pero con menos campanas y silbidos.

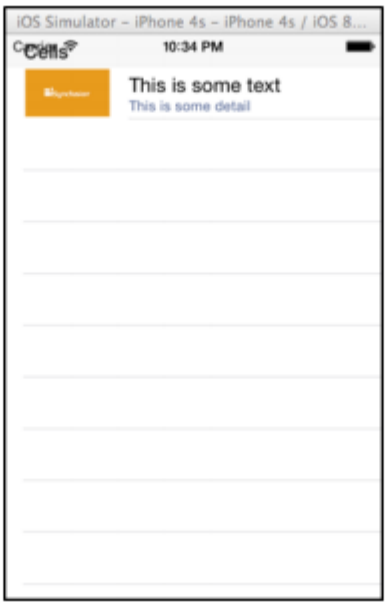
XAML

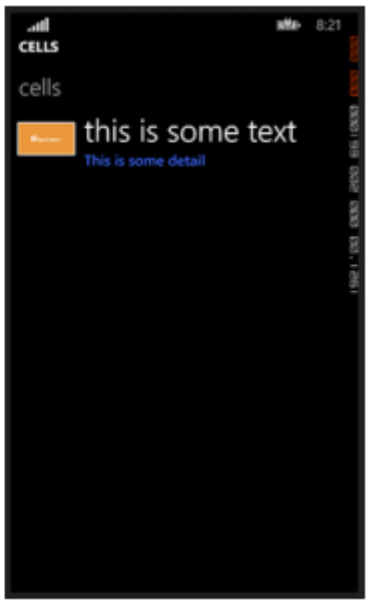
```
<ImageCell ImageSource="http://d2g29cya9iq7ip.cloudfront.net/content/imag
es/company/aboutus-video-bg.png?v=25072014072745")),
 Text="This is some text"
 Detail="This is some detail" />
```

Código

```
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

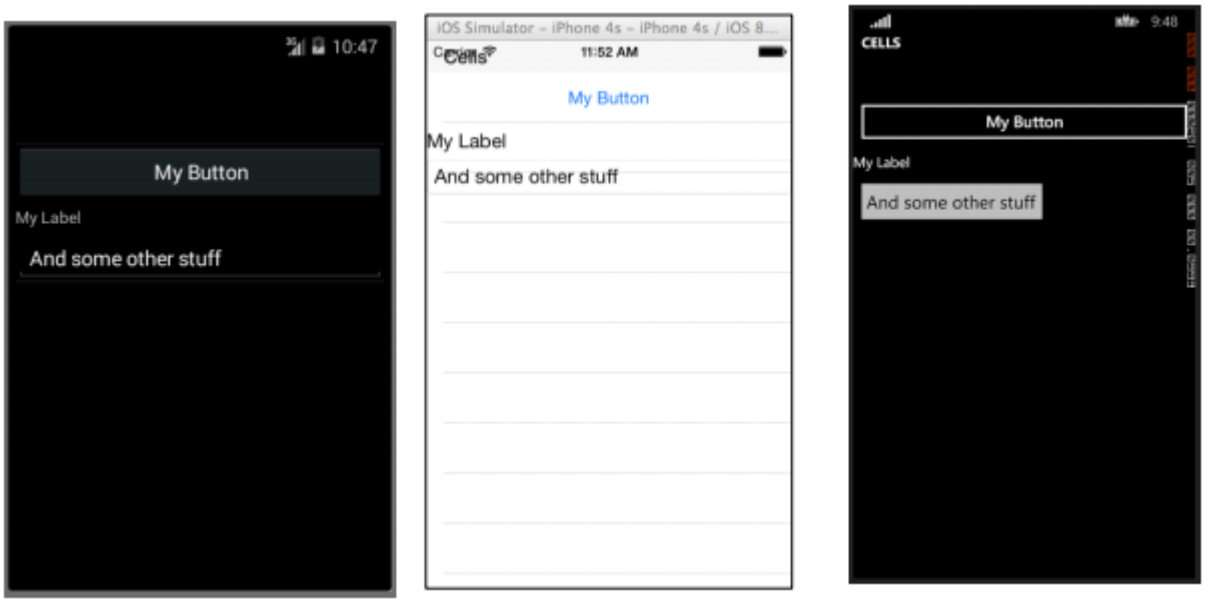
Usted puede considerar a un ViewCell una pizarra en blanco. Es tu lienzo personal crear una celda que se vea exactamente como la quieres. Incluso puede componerlo de instancias de otros objetos de Vista combinados con controles de Diseño. Sólo estás limitado por tu imaginación. Y tal vez el tamaño de la pantalla.

XAML

```
<ViewCell>
<ViewCell.View>
<StackLayout>
<Button Text="My Button"/>

<Label Text="My Label"/>
<Entry Text="And some other stuff"/>
</StackLayout>
</ViewCell.View>
</ViewCell>
```

Código

```
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 }
}
};
```



Lea Xamarin.Formas Células en línea: https://riptutorial.com/es/xamarin-forms/topic/7370/xamarin-formas-celulas

Capítulo 42: Xamarin.Formas vistas

Examples

Botón

El **botón** es probablemente el control más común no solo en aplicaciones móviles, sino en cualquier aplicación que tenga una interfaz de usuario. El concepto de un botón tiene demasiados propósitos para enumerar aquí. Sin embargo, en términos generales, utilizará un botón para permitir a los usuarios iniciar algún tipo de acción u operación dentro de su aplicación. Esta operación podría incluir cualquier cosa, desde la navegación básica dentro de su aplicación, hasta el envío de datos a un servicio web en algún lugar de Internet.

XAML

```
<Button
    x:Name="MyButton"
    Text="Click Me!"
    TextColor="Red"
    BorderColor="Blue"
    VerticalOptions="Center"
    HorizontalOptions="Center"
    Clicked="Button_Clicked"/>
```

Código XAML detrás

```
public void Button_Clicked( object sender, EventArgs args )
{
    MyButton.Text = "I've been clicked!";
}
```

Código

```
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!";
};
```

Selector de fechas

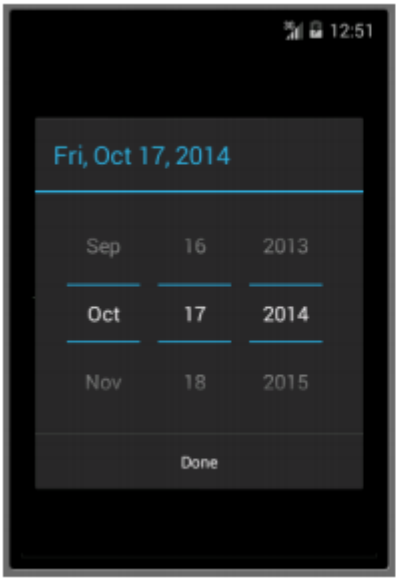
Muy a menudo dentro de las aplicaciones móviles, habrá una razón para tratar con las fechas. Cuando trabaje con fechas, probablemente necesitará algún tipo de información del usuario para seleccionar una fecha. Esto podría ocurrir cuando se trabaja con una aplicación de calendario o calendario. En este caso, es mejor proporcionar a los usuarios un control especializado que les permita elegir una fecha de manera interactiva, en lugar de requerir que los usuarios escriban una fecha manualmente. Aquí es donde el control DatePicker es realmente útil.

XAML

```
<DatePicker Date="09/12/2014" Format="d" />
```

Código

```
var datePicker = new DatePicker{
Date = DateTime.Now,
Format = "d"
};
```



Entrada

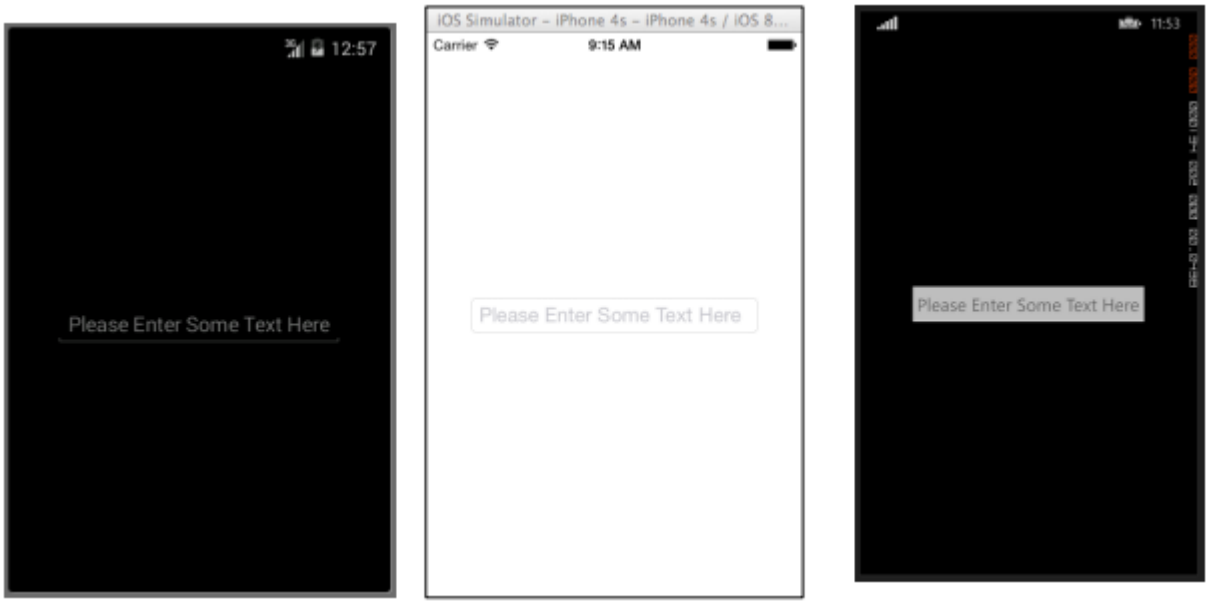
La Vista de entrada se utiliza para permitir a los usuarios escribir una sola línea de texto. Esta única línea de texto se puede usar para múltiples propósitos, incluyendo el ingreso de notas básicas, credenciales, URL y más. Esta vista es una vista multipropósito, lo que significa que si necesita escribir texto regular o quiere ocultar una contraseña, todo se hace a través de este único control.

XAML

```
<Entry Placeholder="Please Enter Some Text Here"
HorizontalOptions="Center"
VerticalOptions="Center"
Keyboard="Email"/>
```

Código

```
var entry = new Entry {
Placeholder = "Please Enter Some Text Here",
HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center,
Keyboard = Keyboard.Email
};
```



Editor

El Editor es muy similar a la Entrada, ya que permite a los usuarios ingresar texto de forma libre. La diferencia es que el Editor permite la entrada multilínea, mientras que la entrada solo se utiliza para la entrada de una sola línea. La Entrada también proporciona algunas propiedades más que el Editor para permitir una mayor personalización de la Vista.

XAML

```
<Editor HorizontalOptions="Fill"
VerticalOptions="Fill"
 Keyboard="Chat"/>
```

Código

```
var editor = new Editor {
HorizontalOptions = LayoutOptions.Fill,
VerticalOptions = LayoutOptions.Fill,
Keyboard = Keyboard.Chat
};
```

Imagen

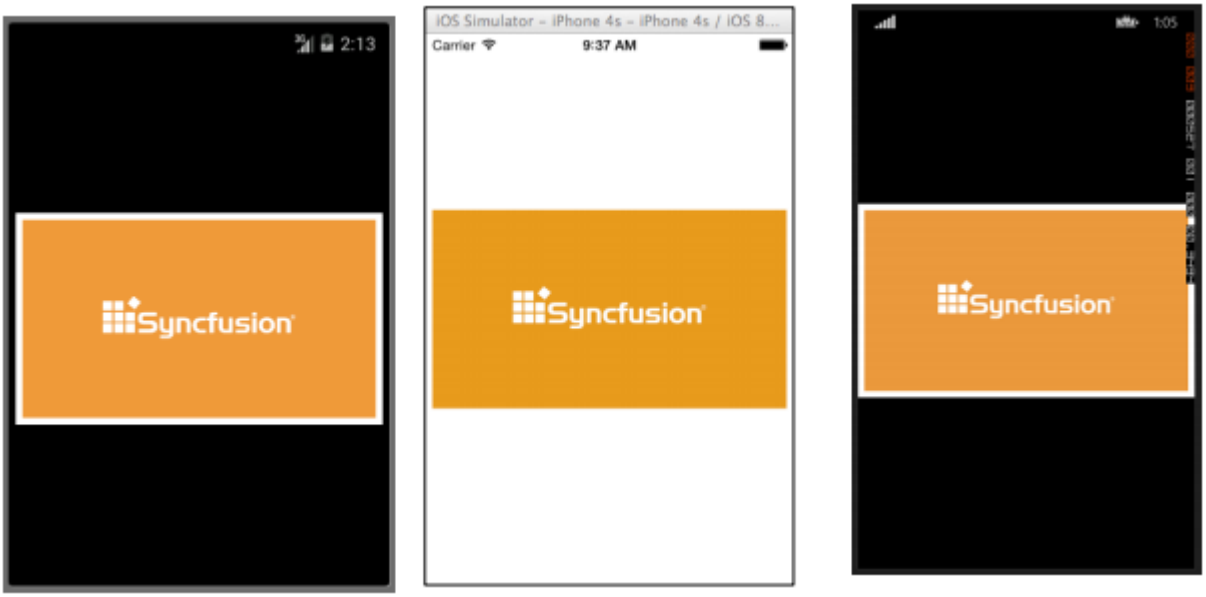
Las imágenes son partes muy importantes de cualquier aplicación. Brindan la oportunidad de inyectar elementos visuales adicionales, así como de marca en su aplicación. Sin mencionar que las imágenes suelen ser más interesantes de ver que el texto o los botones. Puede usar una imagen como elemento independiente dentro de su aplicación, pero un elemento de imagen también se puede agregar a otros elementos de vista, como un botón.

XAML

```
<Image Aspect="AspectFit" Source="http://d2g29cya9iq7ip.cloudfront.net/co
ntent/images/company/aboutus-video-bg.png?v=25072014072745"/>
```

Código

```
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"))
};
```



Etiqueta

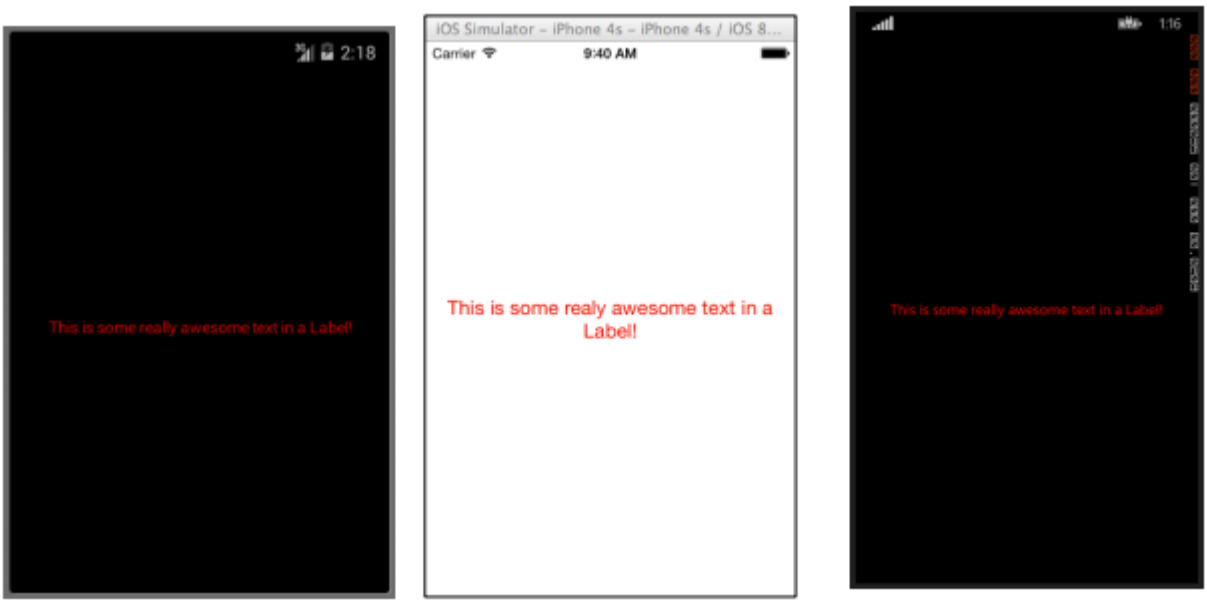
Lo creas o no, el Label es una de las clases de View más importantes y menos apreciadas, no solo en Xamarin.Forms, sino en el desarrollo de IU en general. Se ve como una línea de texto bastante aburrida, pero sin esa línea de texto sería muy difícil transmitir ciertas ideas al usuario. Los controles de etiqueta se pueden usar para describir lo que el usuario debe ingresar en un editor o control de entrada. Pueden describir una sección de la interfaz de usuario y darle contexto. Se pueden utilizar para mostrar el total en una aplicación de calculadora. Sí, la etiqueta es realmente el control más versátil en su bolsa de herramientas que puede no siempre despertar mucha atención, pero es la primera que se observa si no está allí.

XAML

```
<Label Text="This is some really awesome text in a Label!"
TextColor="Red"
XAlign="Center"
YAlign="Center"/>
```

Código

```
var label = new Label {
Text = "This is some really awesome text in a Label!",
TextColor = Color.Red,
XAlign = TextAlignment.Center,
YAlign = TextAlignment.Center
};
```



Lea Xamarin.Formas vistas en línea: https://riptutorial.com/es/xamarin-forms/topic/7369/xamarin-formas-vistas

Capítulo 43: Xamarin.Forms Page

Examples

TabbedPage

Una TabbedPage es similar a una NavigationPage ya que permite y gestiona la navegación simple entre varios objetos secundarios de la página. La diferencia es que, en términos generales, cada plataforma muestra algún tipo de barra en la parte superior o inferior de la pantalla que muestra la mayoría, si no todos, de los objetos de la página secundaria disponibles. En las aplicaciones de Xamarin.Forms, TabbedPage es generalmente útil cuando tiene un pequeño número predefinido de páginas entre las que los usuarios pueden navegar, como un menú o un asistente simple que se puede colocar en la parte superior o inferior de la pantalla.

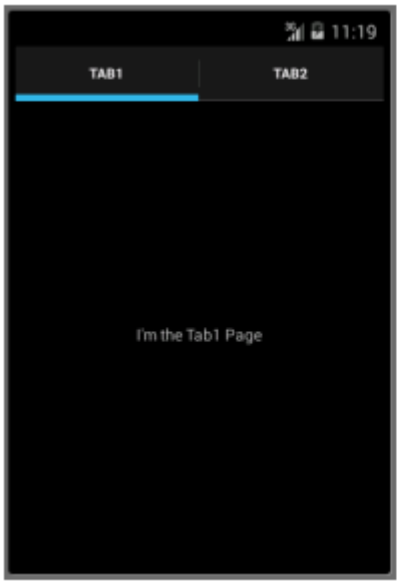
XAML

```
<?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>
```

Código

```
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 }
};
```



Pagina de contenido

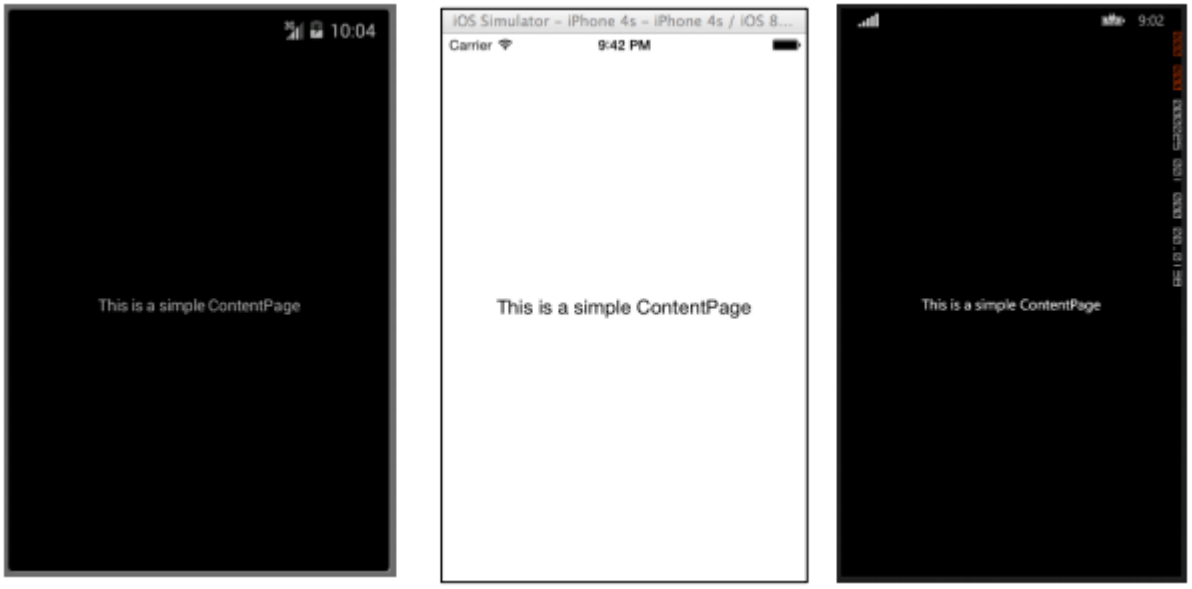
Página de contenido: muestra una vista única.

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>
```

Código

```
var label = new Label {
Text = "This is a simple ContentPage",
HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center
};
var contentPage = new ContentPage {
Content = label
};
```



MasterDetailPage

MasterDetailPage: administra dos páginas separadas (paneles) de información.

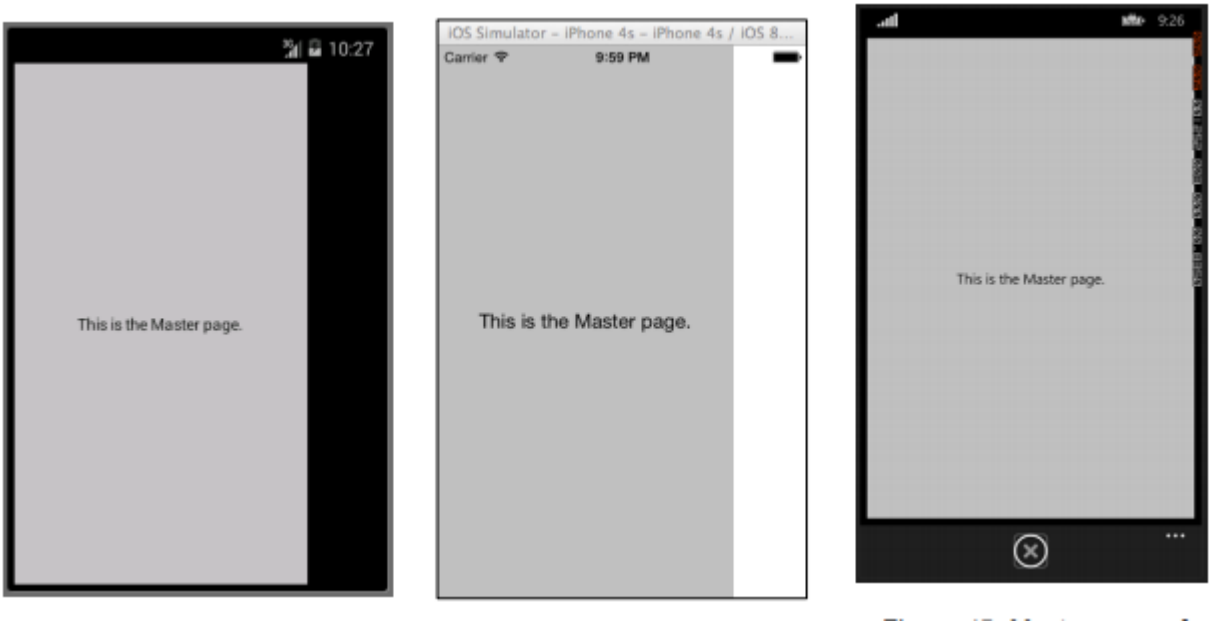
XAML

```
<?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>
```

Código

```
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
}
}
};
```



Lea Xamarin.Forms Page en línea: https://riptutorial.com/es/xamarin-forms/topic/7018/xamarin-forms-page

Creditos

S. No	Capítulos	Contributors
1	Empezando con Xamarin.Forms	Akshay Kulkarni, chrisntr, Community, Demitrian, hankide, jdstaerk, Manohar, patridge, Sergey Metlov, spaceplane
2	¿Ciclo de vida de la aplicación Xamarin.Forms genérico? Dependiente de la plataforma!	Zverev Eugene
3	¿Por qué Xamarin se forma y cuándo usar Xamarin?	Daniel Krzyczkowski, mike
4	Acceso a funciones nativas con DependencyService	Gerald Versluis, hankide, hvaughan3, Sergey Metlov
5	Ajustes visuales específicos de la plataforma.	Alois, GalaxiaGuy, Paul
6	Almacenamiento en caché	Sergey Metlov
7	AppSettings Reader en Xamarin.Forms	Ben Ishiyama-Levy, GvSharma
8	Base de datos SQL y API en Xamarin Forms.	RIYAZ
9	CarouselView - Versión de prelanzamiento	dpserge
10	Complemento Xamarin	Eng Soon Cheah
11	Comportamiento específico de la plataforma	Ege Aydın

12	Creando controles personalizados	hvaughan3, spaceplane, Yehor Hromadskyi
13	DependenciaServicio	Steven Thewissen
14	Diseños de formas de Xamarin	Eng Soon Cheah, Gerald Versluis, Lucas Moura Veloso
15	Disparadores y comportamientos	hamalaiv, hvaughan3
16	Disposición relativa de Xamarin	Ege Aydın
17	Efectos	Swaminathan Vetri
18	El enlace de datos	Andrew, Matthew, Yehor Hromadskyi
19	Examen de la unidad	jerone, Sergey Metlov
20	Fuentes personalizadas en estilos	Roma Rudyak
21	Gesto de Xamarin	Joehl
22	Gestos	doerig, Gerald Versluis, Michael Rumpler
23	Manejo de excepciones	Yehor Hromadskyi
24	MessagingCenter	Gerald Versluis
25	Mostrar alerta	aboozz pallikara, GvSharma, Sreeraj, Yehor Hromadskyi
26	Navegación en Xamarin. Formas	Fernando Arreguín, jimmgarr, Lucas Moura Veloso, Paul, Sergey Metlov, Taras Shevchuk, Willian D. Andrade
27	Notificaciones push	Gerald Versluis, user1568891
28	OAuth2	Eng Soon Cheah
29	Renderizadores personalizados	Bonelol, hankide, Nicolas Bodin-Ripert, Nicolas Bodin-Ripert, nishantvodoo, Yehor Hromadskyi, Zverev Eugene
30	Selector de contactos - Formas Xamarin (Android y iOS)	Pucho Eric
31	Servicios de	RIYAZ

	dependencia	
32	Trabajando con Mapas	Taras Shevchuk
33	Trabajar con bases de datos locales	Luis Beltran, Manohar
34	Unidad de Pruebas BDD en Xamarin. Formas	Ben Ishiyama-Levy
35	Usando ListViews	cvanbeek
36	Xamarin.Formas Células	Eng Soon Cheah
37	Xamarin.Formas vistas	Aaron Thompson, Eng Soon Cheah
38	Xamarin.Forms Page	Eng Soon Cheah