eBook Gratuit

APPRENEZ Xamarin.Forms

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#xamarin.fo

rms

Table des matières

À propos

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

Chapitre 1: Démarrer avec Xamarin.Forms

Remarques

Xamarin.Forms permet de créer des applications iOS, Android et Windows avec de grandes quantités de code partagé, notamment du code d'interface utilisateur ou du balisage d'interface utilisateur XAML. Les pages et les vues des applications sont mappées aux contrôles natifs de chaque plate-forme, mais peuvent être personnalisées pour fournir une interface utilisateur spécifique à la plate-forme ou pour accéder à des fonctionnalités spécifiques à la plate-forme.

Versions

Version	Date de sortie
2.3.1	2016-08-03
2.3.0-correctif1	2016-06-29
2.3.0	2016-06-16
2.2.0-correctif1	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

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

Examples

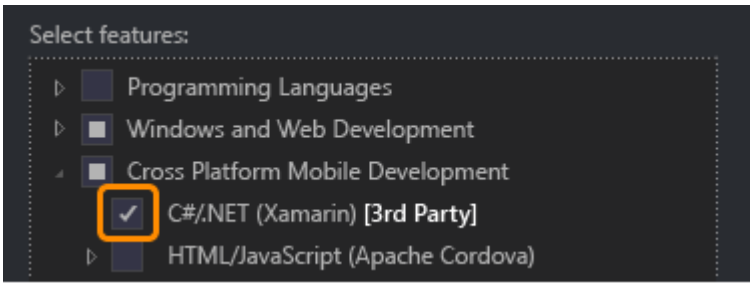
Installation (Visual Studio)

Xamarin.Forms est une abstraction d'interface utilisateur basée sur une plate-forme d'interface utilisateur native, qui permet aux développeurs de créer facilement des interfaces utilisateur pouvant être partagées entre Android, iOS, Windows et Windows Phone. Les interfaces utilisateur sont rendues à l'aide des contrôles natifs de la plate-forme cible, permettant aux applications Xamarin.Forms de conserver l'apparence appropriée pour chaque plate-forme.

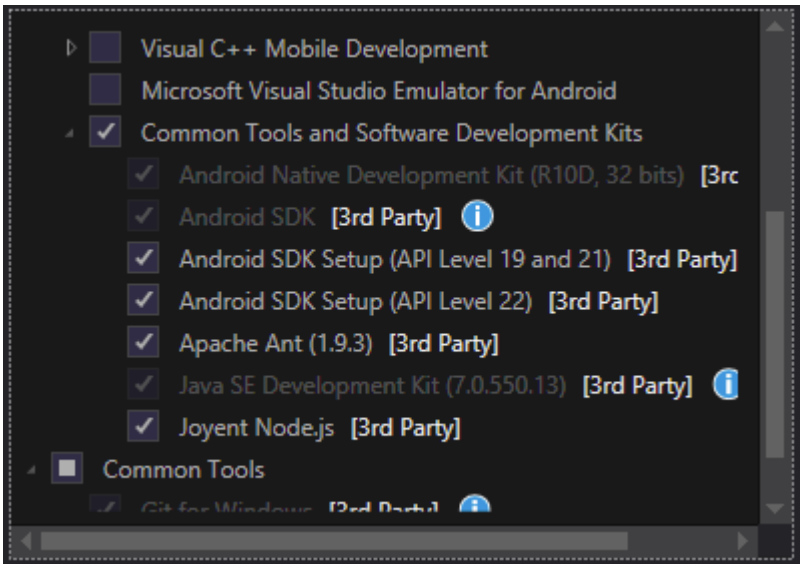
Xamarin Plugin pour Visual Studio

Pour commencer à utiliser Xamarin.Forms pour Visual Studio, vous devez avoir le plug-in Xamarin lui-même. La manière la plus simple de l’installer est de télécharger et d’installer la dernière version de Visual Studio.

Si vous avez déjà installé la dernière version de Visual Studio, accédez au Panneau de configuration> Programmes et fonctionnalités, cliquez avec le bouton droit sur Visual Studio, puis cliquez sur Modifier. Lorsque le programme d'installation s'ouvre, cliquez sur Modifier et sélectionnez les outils de développement mobiles multiplates-formes:



Vous pouvez également choisir d'installer le SDK Android:



Décochez-la si le SDK est déjà installé. Vous pourrez configurer Xamarin pour utiliser le SDK Android existant plus tard.

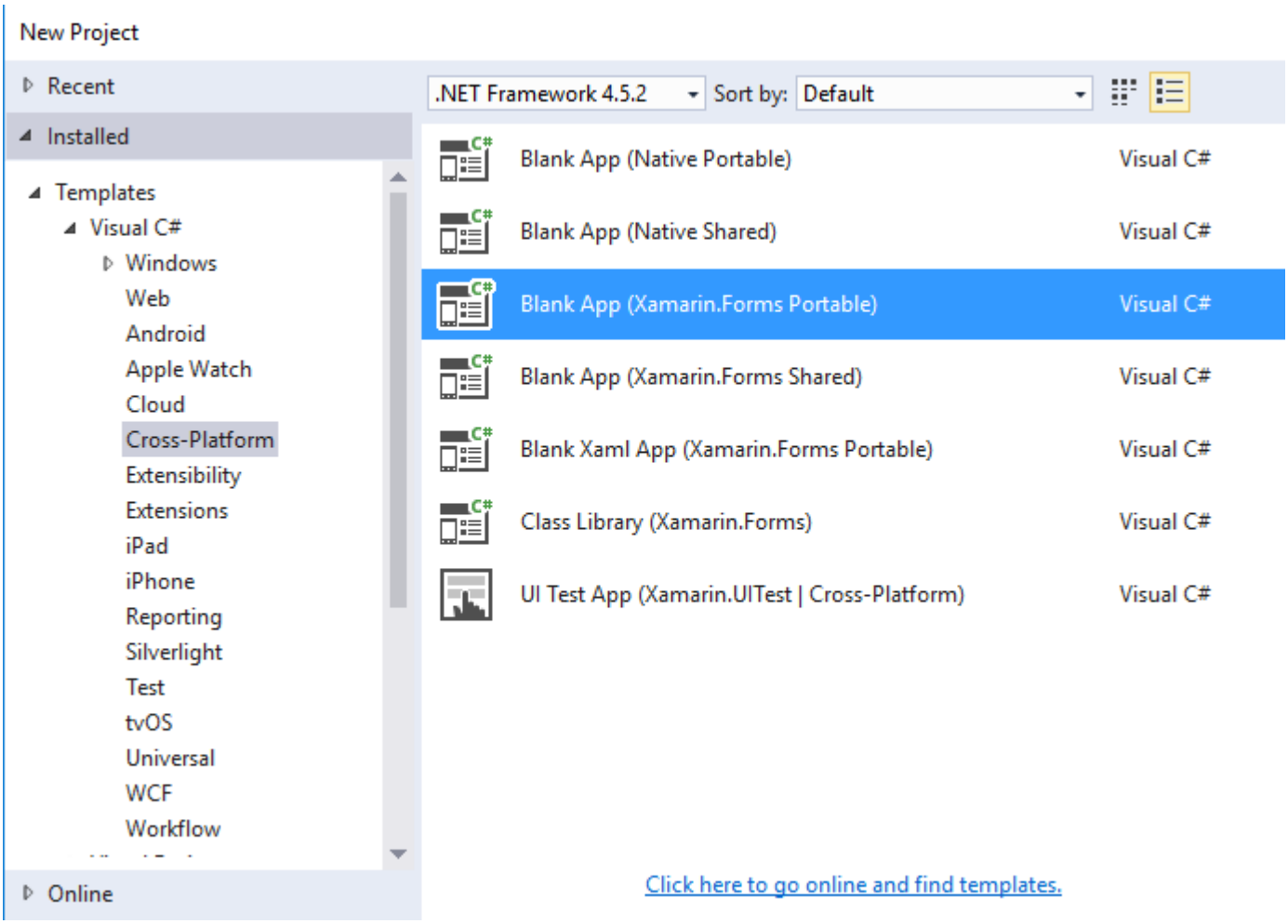
Xamarin.Forms

Xamarin.Forms est un ensemble de bibliothèques pour votre bibliothèque Portable Class et vos assemblys natifs. La bibliothèque Xamarin.Forms est disponible sous forme de package NuGet. Pour l'ajouter à votre projet, utilisez simplement la commande `Install-Package` standard de la console du gestionnaire de packages:

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

pour tous vos assemblages initiaux (par exemple MyProject, MyProject.Droid et MyProject.iOS).

La manière la plus simple de commencer avec Xamarin.Forms est de créer un projet vide dans Visual Studio:



Comme vous pouvez le voir, deux options sont disponibles pour créer l'application vide: Portable et Partagé. Je vous recommande de commencer avec Portable un parce que c'est le plus couramment utilisé dans le monde réel (différences et plus d'explications à ajouter).

Après avoir créé le projet, assurez-vous d'utiliser la dernière version de Xamarin.Forms car votre modèle initial peut contenir l'ancien. Utilisez l'option Console du gestionnaire de paquets ou Gérer les packages NuGet pour effectuer la mise à niveau vers la dernière version de Xamarin.Form (n'oubliez pas qu'il s'agit simplement d'un package NuGet).

Alors que les modèles Visual Studio Xamarin.Forms créeront un projet de plate-forme iOS pour vous, vous devrez connecter Xamarin à un hôte de génération Mac pour pouvoir exécuter ces projets sur le simulateur iOS ou les périphériques physiques.

Hello World Xamarin Forms: Visual Studio

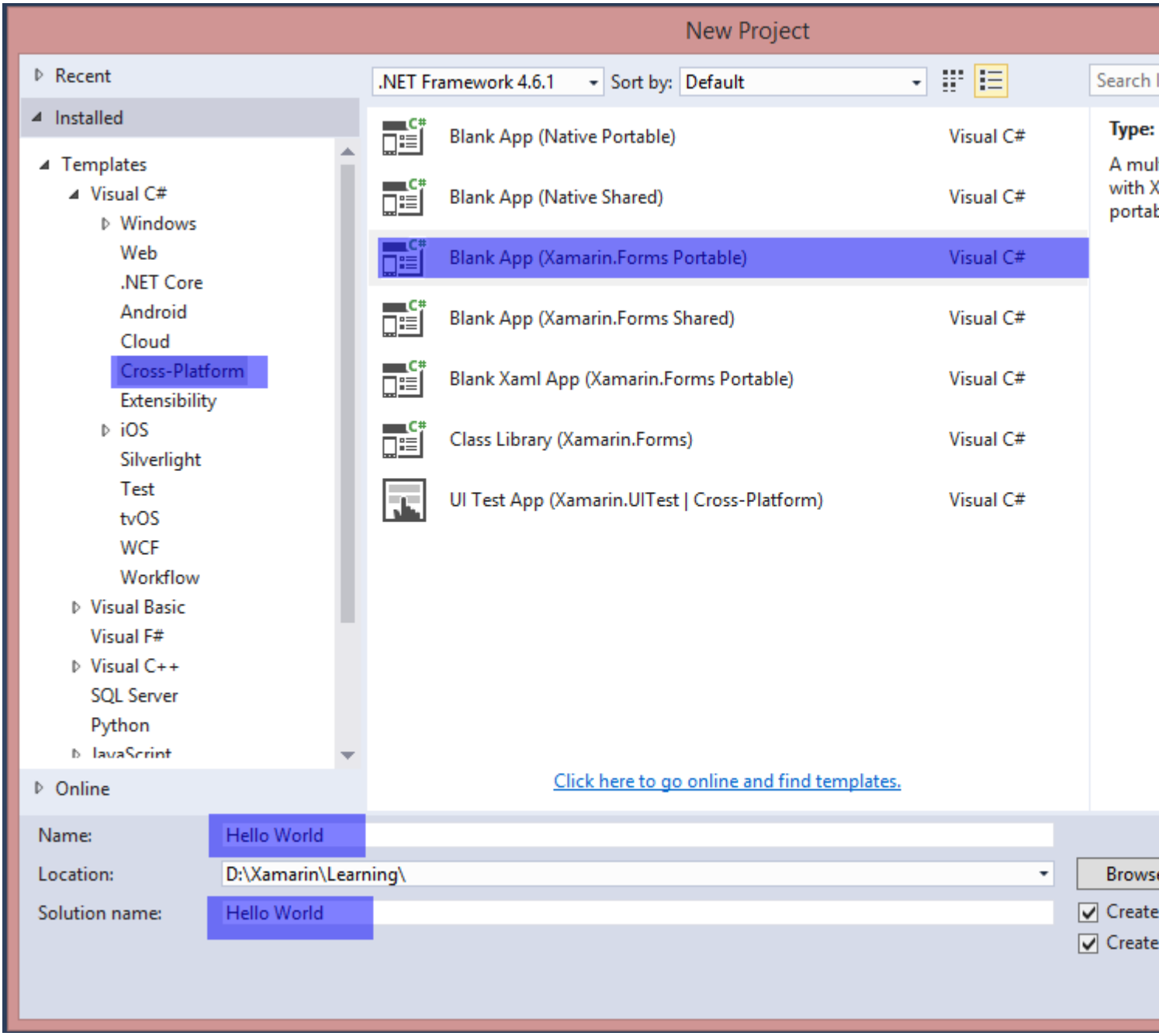
Après l'installation réussie de Xamarin comme décrit dans le premier exemple, il est temps de lancer le premier exemple d'application.

Étape 1: Création d'un nouveau projet.

Dans Visual Studio, choisissez Nouveau -> Projet -> Visual C # -> Plate-forme croisée -> Application vide (Xamarin.Forms Portable)

Nommez l'application "Hello World" et sélectionnez l'emplacement pour créer le projet et cliquez sur OK. Cela créera une solution pour vous qui contient trois projets:

1. HelloWorld (c'est là que se trouve votre logique et vos vues, c'est-à-dire le projet portable)
2. HelloWorld.Droid (le projet Android)
3. HelloWorld.iOS (le projet iOS)



Étape 2: Recherche de l'échantillon

Après avoir créé la solution, un exemple d'application sera prêt à être déployé. Ouvrez le `App.cs` situé à la racine du projet portable et `App.cs` le code. Comme on le voit ci-dessous, le `Content` s de l'échantillon est un `StackLayout` qui contient une `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
        }
    }
}
```

Étape 3: lancement de l'application

Maintenant, cliquez simplement sur le projet que vous souhaitez démarrer (`HelloWorld.Droid` ou `HelloWorld.iOS`) et cliquez sur `Set as StartUp Project` de démarrage. Ensuite, dans la barre d'outils Visual Studio, cliquez sur le bouton `Start` (le bouton triangulaire vert qui ressemble à un bouton Lecture) pour lancer l'application sur le simulateur / émulateur ciblé.

Lire Démarrer avec Xamarin.Forms en ligne: https://riptutorial.com/fr/xamarin-forms/topic/908/demarrer-avec-xamarin-forms

Chapitre 2: Accès aux fonctionnalités natives avec DependencyService

Remarques

Si vous ne voulez pas que votre code se brise lorsqu'aucune implémentation n'est trouvée, vérifiez d'abord le `DependencyService` s'il dispose d'une implémentation disponible.

Vous pouvez le faire par une simple vérification si elle n'est pas `null` .

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

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

ou, si votre IDE prend en charge C # 6, avec un opérateur conditionnel nul:

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

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

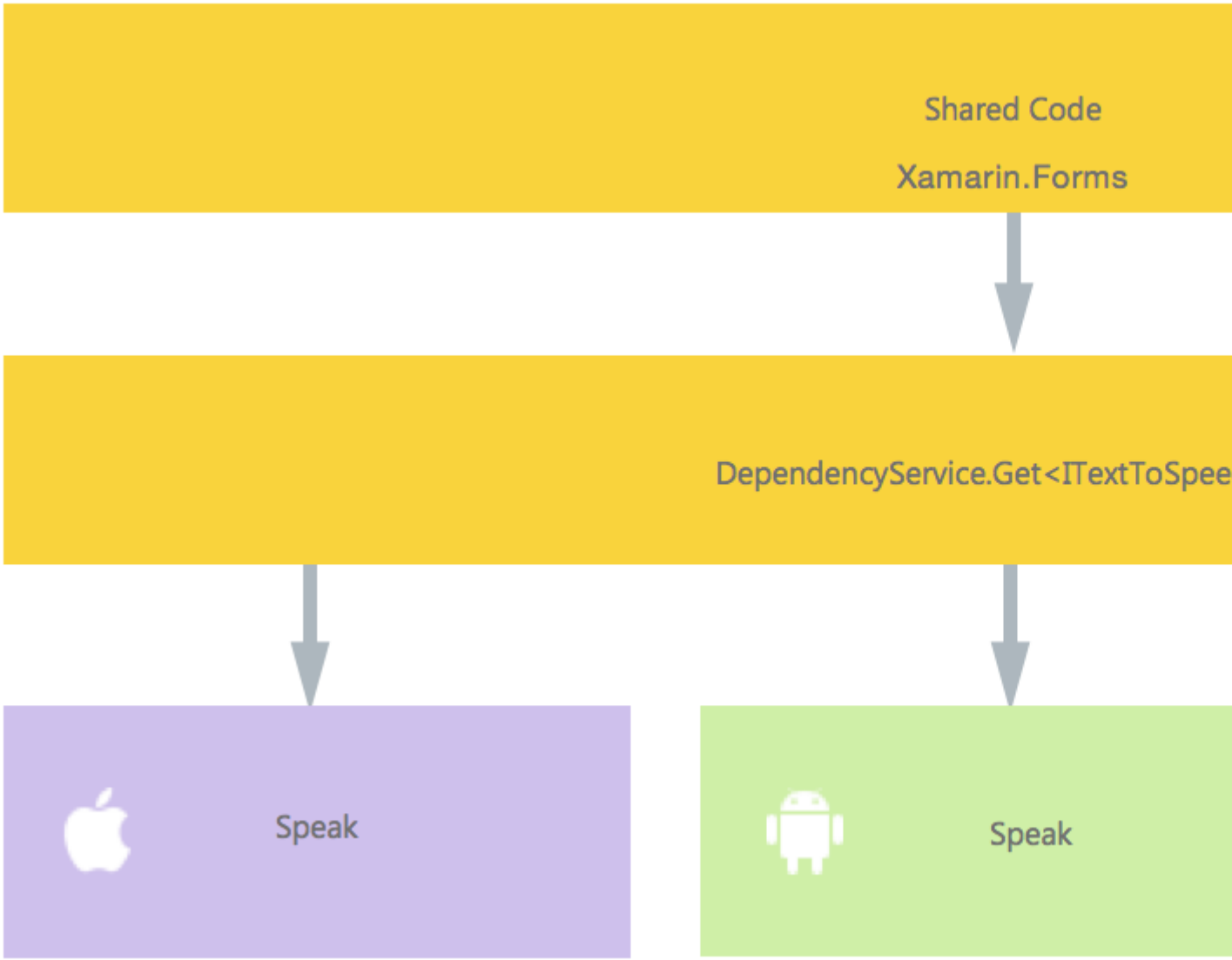
Si vous ne le faites pas et qu'aucune implémentation n'est trouvée à l'exécution, votre code générera une exception.

Examples

Mise en œuvre de la synthèse vocale

Un bon exemple de fonctionnalité qui demande un code spécifique à la plate-forme consiste à implémenter le text-to-speech (tts). Cet exemple suppose que vous travaillez avec du code partagé dans une bibliothèque PCL.

Un aperçu schématique de notre solution ressemblerait à l'image ci-dessous.



Dans notre code partagé, nous définissons une interface qui est enregistrée avec `DependencyService` . C'est là que nous ferons appel à nous. Définir une interface comme ci-dessous.

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

Maintenant, dans chaque plate-forme spécifique, nous devons créer une implémentation de cette interface. Commençons par l'implémentation iOS.

Mise en œuvre 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);
    }
}
```

Dans l'exemple de code ci-dessus, vous remarquez qu'il existe un code spécifique à iOS. Comme les types tels que `AVSpeechSynthesizer` . Celles-ci ne fonctionneraient pas dans le code partagé.

Pour enregistrer cette implémentation avec Xamarin `DependencyService` ajoutez cet attribut au-dessus de la déclaration d'espace de noms.

```
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
```

Maintenant, lorsque vous faites un appel comme celui-ci dans votre code partagé, la bonne implémentation de la plate-forme sur laquelle vous exécutez votre application est injectée.

`DependencyService.Get<ITextToSpeech>()` . Plus à ce sujet plus tard.

Implémentation Android

L'implémentation Android de ce code ressemblerait à celle ci-dessous.

```
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
}
```

Encore une fois, n'oubliez pas de l'enregistrer avec le `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
```

Implémentation Windows Phone

Enfin, pour Windows Phone, ce code peut être utilisé.

```
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);
    }
}
```

Et une fois de plus, n'oubliez pas de l'enregistrer.

```
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
```

Implémentation en code partagé

Maintenant, tout est en place pour que ça marche! Enfin, dans votre code partagé, vous pouvez maintenant appeler cette fonction en utilisant l'interface. Lors de l'exécution, l'implémentation sera injectée, ce qui correspond à la plate-forme actuelle sur laquelle elle s'exécute.

Dans ce code, vous verrez une page qui pourrait être dans un projet Xamarin Forms. Il crée un bouton qui appelle la méthode `Speak()` à l'aide de `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;
}
```

Le résultat sera que lorsque l'application est exécutée et que le bouton est cliqué, le texte fourni sera prononcé.

Tout cela sans avoir à faire des choses difficiles comme des conseils de compilation et autres. Vous disposez désormais d'un moyen unique d'accéder aux fonctionnalités spécifiques à la plateforme via un code indépendant de la plate-forme.

Obtenir les numéros de version du système d’exploitation et d’appareils - Android & iOS - PCL

L'exemple ci-dessous collectera le numéro de version du système d'exploitation de l'appareil et la version de l'application (définie dans les propriétés de chaque projet) entrée dans **Nom de la version** sur Android et **Version** sur iOS.

Faites d'abord une interface dans votre projet 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();
}
```

Maintenant, nous implémentons l'interface dans les projets Android et iOS.

Android:

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

namespace YourNamespace.Droid{
    public class NativeHelper_Android : INativeHelper {

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

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

iOS:

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

namespace YourNamespace.iOS {
    public class NativeHelper_iOS : INativeHelper {

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

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

```
}
```

Maintenant, utilisez le code dans une méthode:

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

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

Lire Accès aux fonctionnalités natives avec DependencyService en ligne: https://riptutorial.com/fr/xamarin-forms/topic/2409/acces-aux-fonctionnalites-natives-avec-dependencyservice

Chapitre 3: Ajustements visuels spécifiques à la plate-forme

Examples

Ajustements Idiom

Des ajustements spécifiques à l'idiome peuvent être effectués à partir du code C #, par exemple pour modifier l'orientation de la mise en page, que la vue soit affichée ou un téléphone ou une tablette.

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

Ces fonctionnalités sont également disponibles directement à partir du code 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>
```

Ajustements de la plateforme

Des ajustements peuvent être effectués pour des plates-formes spécifiques à partir du code C #, par exemple pour modifier le remplissage de toutes les plates-formes ciblées.

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

Une méthode d'assistance est également disponible pour les déclarations C # raccourcies:

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

Ces fonctionnalités sont également disponibles directement à partir du code XAML:

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

Utiliser des styles

Lorsque vous travaillez avec XAML, l'utilisation d'un `Style` centralisé vous permet de mettre à jour un ensemble de vues stylisées à partir d'un seul endroit. Tous les réglages de langue et de plate-forme peuvent également être intégrés à vos styles.

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

Utiliser des vues personnalisées

Vous pouvez créer des vues personnalisées pouvant être intégrées à votre page grâce à ces outils de réglage.

Sélectionnez `File > New > File... > Forms > Forms ContentView (Xaml)` et créez une vue pour chaque présentation spécifique: `TabletHome.xaml` et `PhoneHome.xaml` .

Sélectionnez ensuite `File > New > File... > Forms > Forms ContentPage` et créez un `File > New > File... > Forms > Forms ContentPage HomePage.cs` contenant:

```
using Xamarin.Forms;

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

    }
}
```

Vous avez maintenant un `HomePage` qui crée une hiérarchie de vue différente pour `Phone` idiomes du `Phone` et de la `Tablet` .

Lire Ajustements visuels spécifiques à la plate-forme en ligne: https://riptutorial.com/fr/xamarin-forms/topic/5012/ajustements-visuels-specifiques-a-la-plate-forme

Chapitre 4: Alerte d'affichage

Examples

DisplayAlert

Une boîte d'alerte peut être `Xamarin.Forms Page` sur une `Xamarin.Forms Page` par la méthode `DisplayAlert` . Nous pouvons fournir un titre, un corps (texte à alerter) et un / deux boutons d'action. `Page` offre deux substitutions de la méthode `DisplayAlert` .

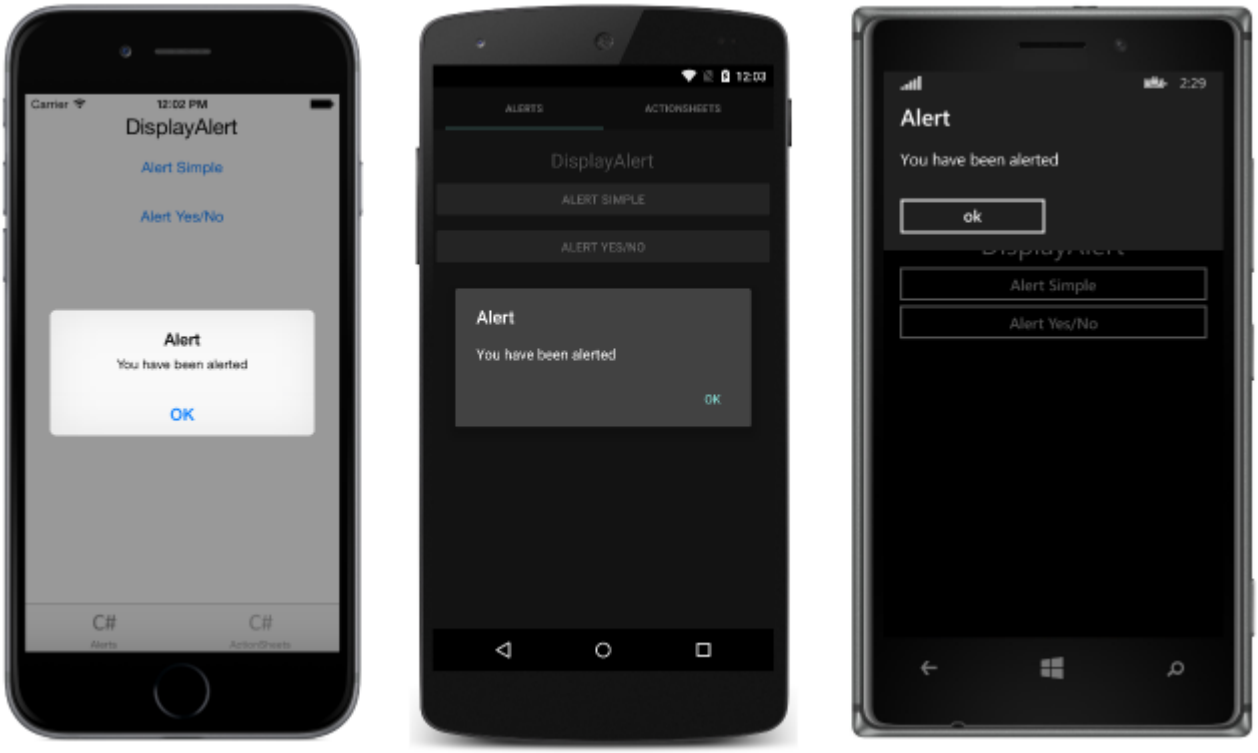
1. `public Task DisplayAlert (String title, String message, String cancel)`

Cette substitution présente une boîte de dialogue d'alerte à l'utilisateur de l'application avec un seul bouton d'annulation. L'alerte s'affiche de manière modale et une fois renvoyée, l'utilisateur continue d'interagir avec l'application.

Exemple :

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

L'extrait ci-dessus présentera une implémentation native des alertes dans chaque plate-forme (`AlertDialog` dans Android, `UIAlertView` dans iOS, `MessageDialog` dans Windows) comme ci-dessous.



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

Cette substitution présente une boîte de dialogue d'alerte à l'utilisateur de l'application avec un bouton d'acceptation et un bouton d'annulation. Il capture la réponse d'un utilisateur en présentant

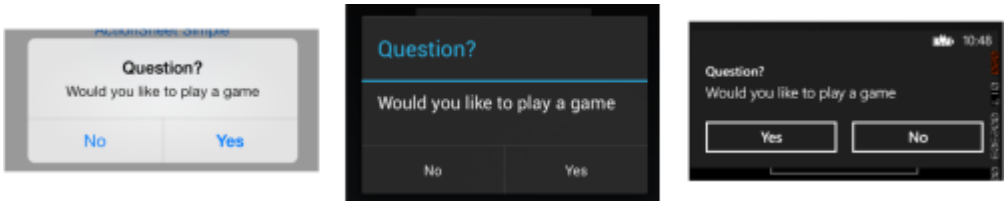
deux boutons et en retournant un `boolean` . Pour obtenir une réponse d'une alerte, fournissez le texte pour les deux boutons et attendez la méthode. Une fois que l'utilisateur a sélectionné l'une des options, la réponse sera renvoyée au code.

Exemple :

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

Exemple 2: (si condition true ou false check pour alerter continuer)

```
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
        }
     }
```



Exemple d'alerte avec un seul bouton et action

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

Ici, nous allons obtenir une action de clic OK.

Lire Alerte d'affichage en ligne: https://riptutorial.com/fr/xamarin-forms/topic/4883/alerte-d-affichage

Chapitre 5: AppSettings Reader dans Xamarin.Forms

Examples

Lecture du fichier app.config dans un projet Xamarin.Forms Xaml

Bien que chaque plate-forme mobile offre ses propres api de gestion des paramètres, il n'existe aucune méthode intégrée pour lire les paramètres à partir d'un bon fichier app.config xml de style .net; Cela est dû à un bon nombre de bonnes raisons, notamment l'api de gestion de la configuration du framework .net, qui est plutôt lourde, et chaque plate-forme ayant son propre API de système de fichiers.

Nous avons donc construit une bibliothèque PCLAppConfig simple, joliment empaquetée pour votre consommation immédiate.

Cette bibliothèque utilise la belle bibliothèque PCLStorage

Cet exemple suppose que vous développez un projet Xamarin.Forms Xaml, où vous devez accéder aux paramètres de votre modèle de vue partagé.

1. Initialisez ConfigurationManager.AppSettings sur chacun de vos projets de plate-forme, juste après l'instruction 'Xamarin.Forms.Forms.Init', comme indiqué ci-dessous:

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. Ajoutez un fichier app.config à votre projet PCL partagé et ajoutez vos entrées appSettings, comme vous le feriez avec n'importe quel fichier app.config

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

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

3. Ajoutez ce fichier app.config PCL en **tant que fichier lié** sur tous vos projets de plate-forme. Pour Android, assurez-vous de définir l'action de **génération** sur **'AndroidAsset'** , car UWP définit l'action de génération sur **'Contenu'**

4. Accédez à votre paramètre: `ConfigurationManager.AppSettings["config.text"];`

Lire AppSettings Reader dans Xamarin.Forms en ligne: https://riptutorial.com/fr/xamarin-forms/topic/5911/appsettings-reader-dans-xamarin-forms

Chapitre 6: Base de données SQL et API dans les formulaires Xamarin.

Remarques

Créez votre propre API avec la base de données Microsoft SQL et implémentez-les dans l'application de formulaires Xamarin.

Examples

Créer une API à l'aide d'une base de données SQL et implémenter des formulaires Xamarin,

Blog du code source

Lire Base de données SQL et API dans les formulaires Xamarin. en ligne: https://riptutorial.com/fr/xamarin-forms/topic/6513/base-de-donnees-sql-et-api-dans-les-formulaires-xamarin-

Chapitre 7: CarouselView - Version préliminaire

Remarques

CarouselView est un contrôle Xamarin qui peut contenir tout type de vue. Ce contrôle préalable à la publication ne peut être utilisé que dans les projets Xamarin Forms.

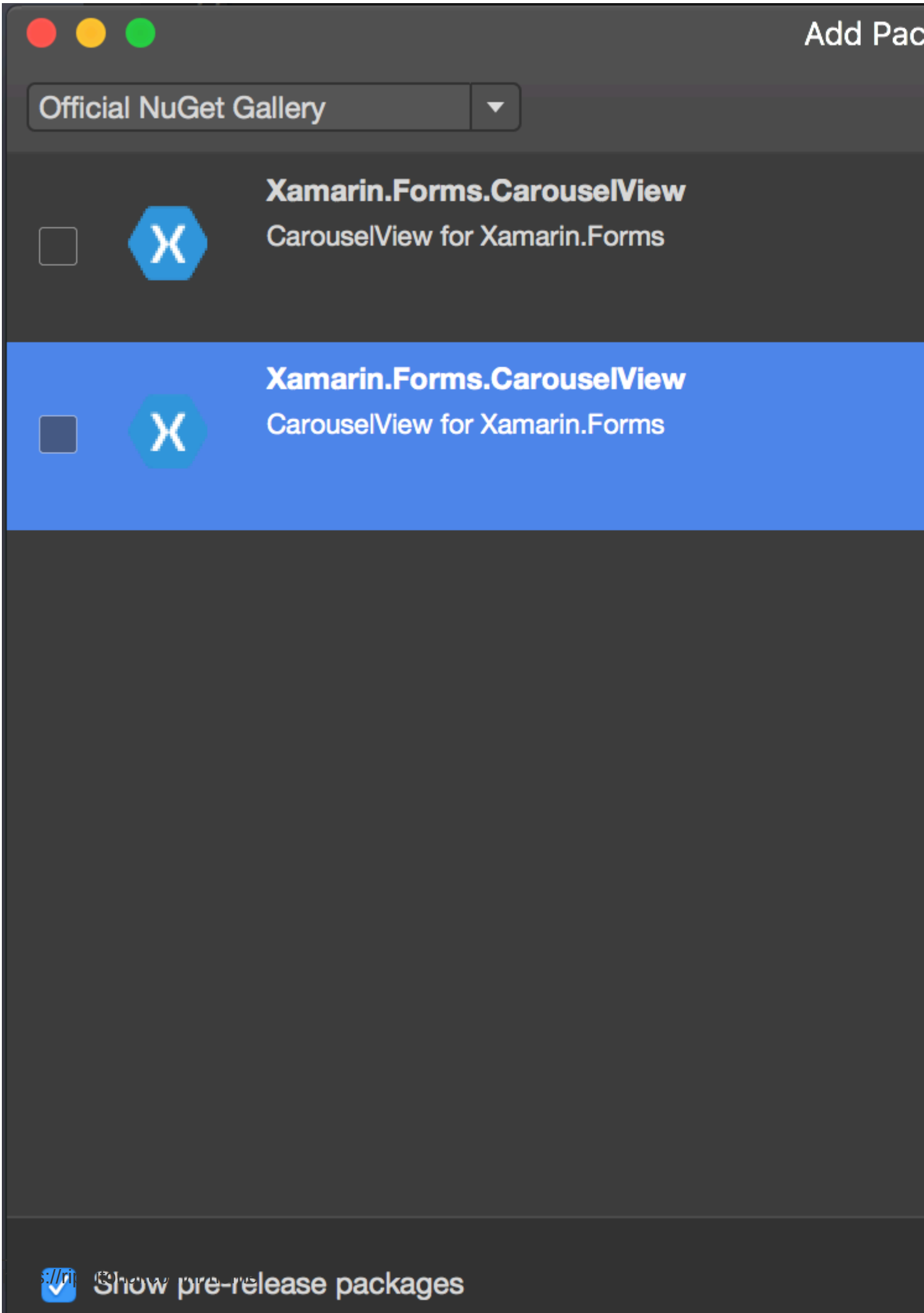
Dans l'exemple fourni par James Montemagno , sur le blog de Xamarin, CarouselView est utilisé pour afficher des images.

CarouselView n'est actuellement pas intégré à Xamarin.Forms. Pour utiliser ceci dans vos projets, vous devrez ajouter le NuGet-Package (voir exemple ci-dessus).

Examples

Importer CarouselView

La méthode la plus simple pour importer CarouselView consiste à utiliser NuGet-Packages Manager dans 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/fr/xamarin-forms/topic/6094/carouselview---version-preliminaire

Chapitre 8: Cellules Xamarin.Forms

Examples

EntryCell

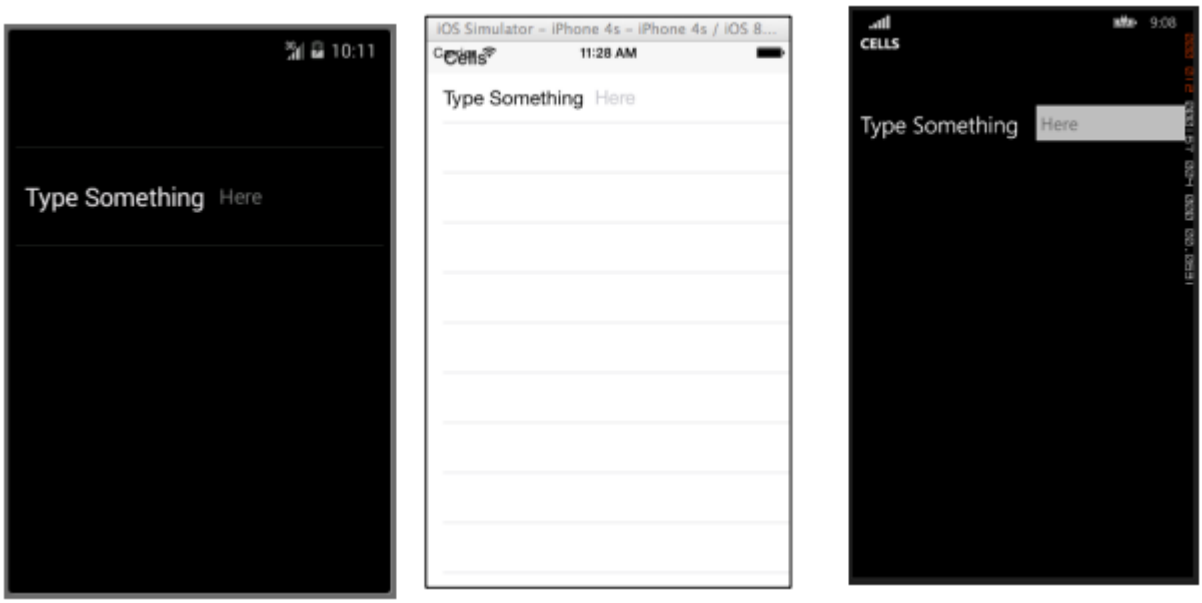
Un EntryCell est une cellule qui combine les capacités d'une étiquette et d'une entrée. EntryCell peut être utile dans des scénarios lors de la création de fonctionnalités au sein de votre application pour collecter des données auprès de l'utilisateur. Ils peuvent facilement être placés dans un tableau et être traités comme un simple formulaire.

XAML

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

Code

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



SwitchCell

Une SwitchCell est une cellule qui combine les capacités d'une étiquette et d'un commutateur on-off. Une SwitchCell peut être utile pour activer ou désactiver la fonctionnalité, ou même les préférences utilisateur ou les options de configuration.

XAML

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

Code

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

TextCell

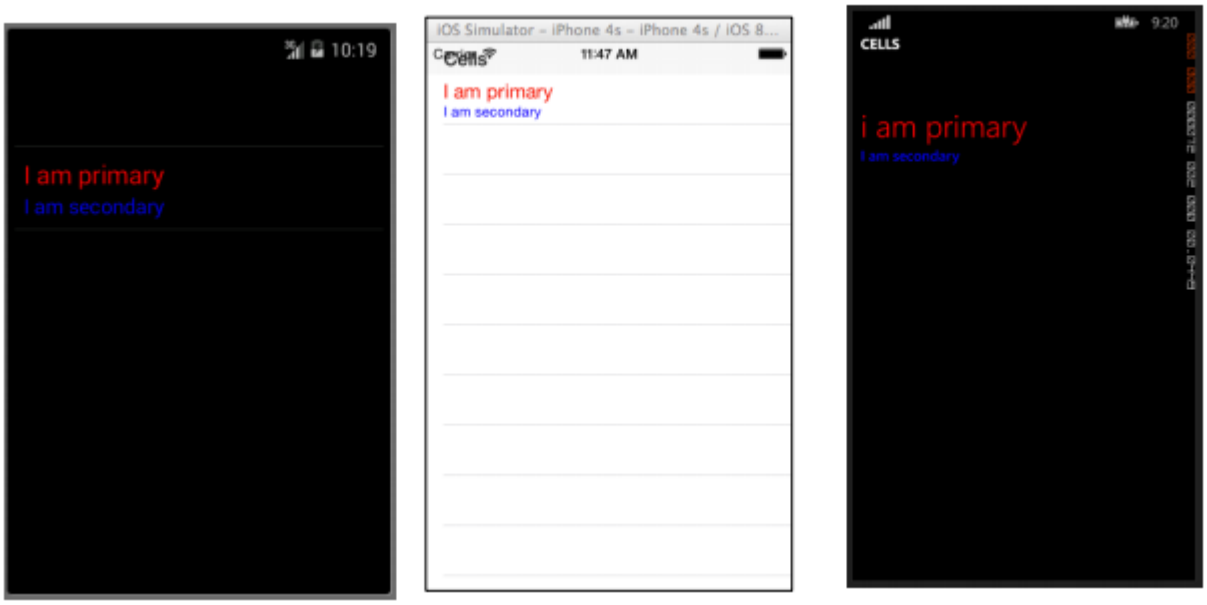
Un objet TextCell est une cellule comportant deux zones de texte distinctes pour l'affichage des données. Un objet TextCell est généralement utilisé à des fins d'information dans les contrôles TableView et ListView. Les deux zones de texte sont alignées verticalement pour maximiser l'espace dans la cellule. Ce type de cellule est également couramment utilisé pour afficher des données hiérarchiques. Ainsi, lorsque l'utilisateur appuie sur cette cellule, il accède à une autre page.

XAML

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

Code

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



ImageCell

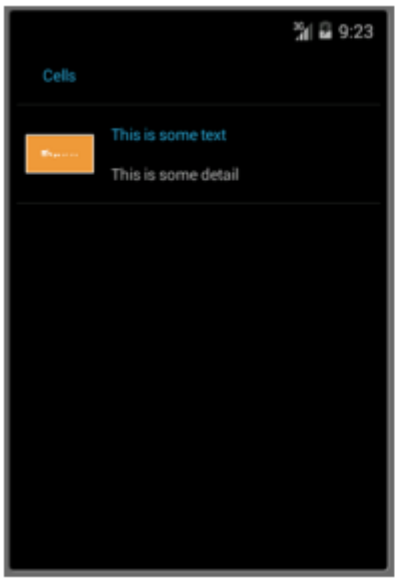
Un ImageCell est exactement ce que cela ressemble. C'est une cellule simple qui ne contient qu'une image. Ce contrôle fonctionne de manière très similaire à un contrôle d'image normal, mais avec beaucoup moins de cloches et de sifflets.

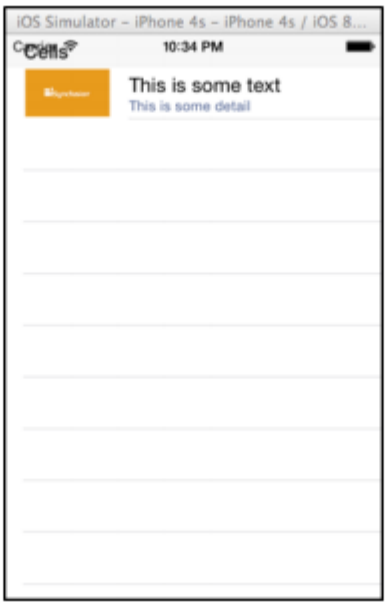
XAML

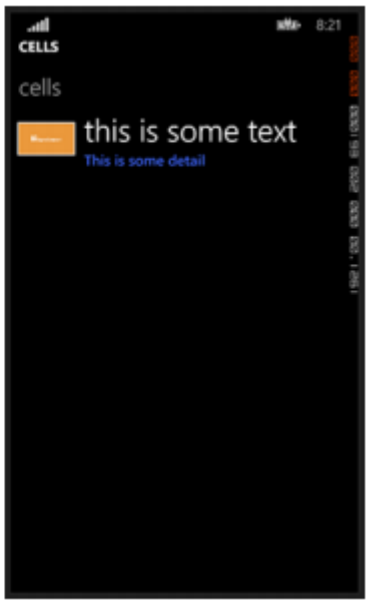
```
<ImageCell ImageSource="http://d2g29cya9iq7ip.cloudfront.net/content/imag
es/company/aboutus-video-bg.png?v=25072014072745")),
 Text="This is some text"
 Detail="This is some detail" />
```

Code

```
var imageCell = new ImageCell {
ImageSource = ImageSource.FromUri(new Uri("http://d2g29cya9iq7ip.clou
109
dfront.net/content/images/company/aboutus-videobg.png?v=25072014072745")),
 Text = "This is some text",
 Detail = "This is some detail"
};
```







ViewCell

Vous pouvez considérer un ViewCell comme une ardoise vierge. C'est votre toile personnelle pour créer une cellule qui vous ressemble exactement. Vous pouvez même le composer d'instances de plusieurs autres objets View associées aux contrôles Layout. Tu es seulement limité par ton imagination. Et peut-être la taille de l'écran.

XAML

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

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

Code

```
var button = new Button { Text = "My Button" };
var label = new Label { Text = "My Label" };
var entry = new Entry { Text ="And some other stuff" };
var viewCell = new ViewCell {
View = new StackLayout {
Children = { button, label, entry }
}
};
```

Lire Cellules Xamarin.Forms en ligne: https://riptutorial.com/fr/xamarin-forms/topic/7370/cellules-xamarin-forms

Chapitre 9: Comportement spécifique à la plate-forme

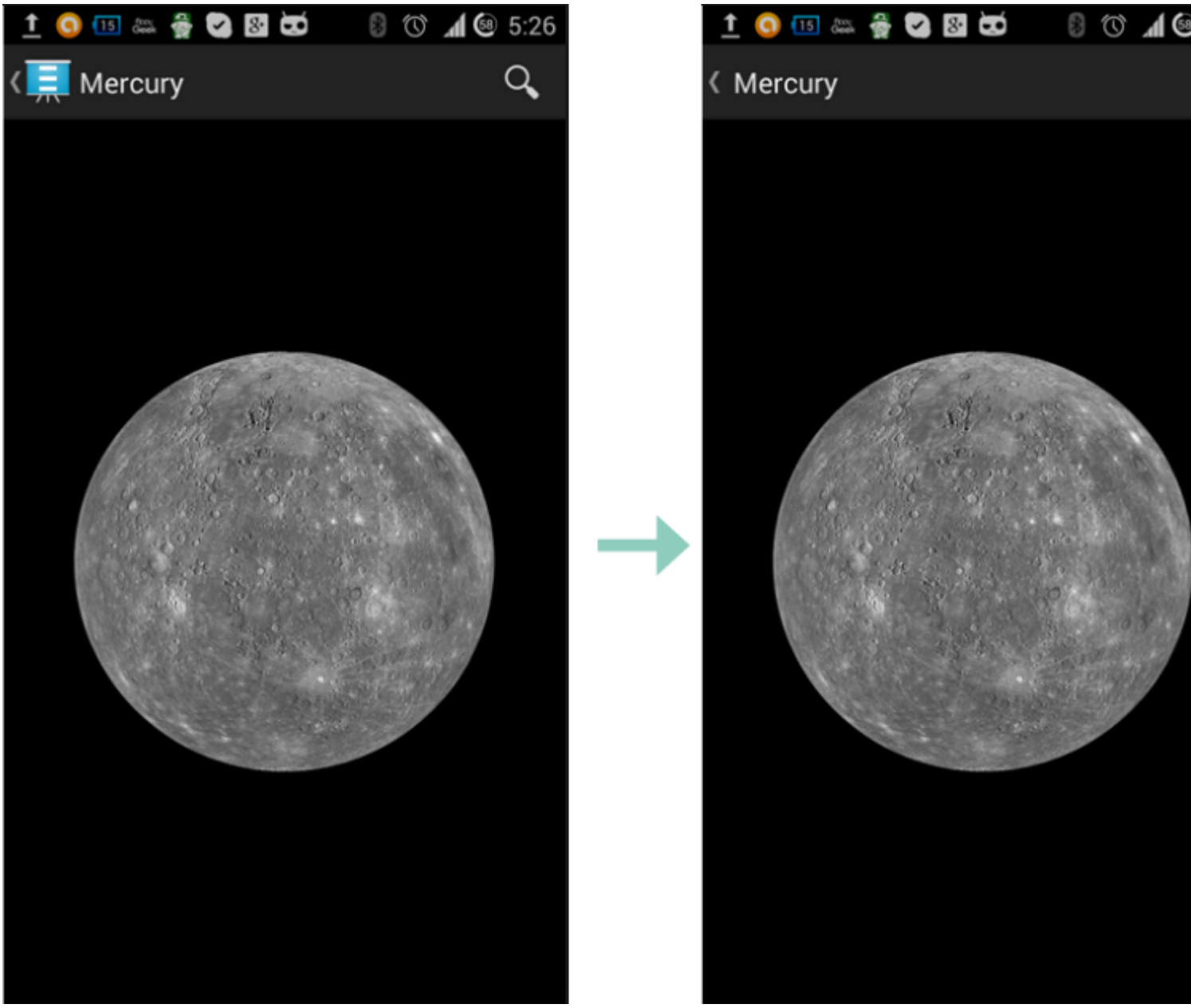
Remarques

Plateformes cibles

```
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

Supprimer l'icône dans l'en-tête de navigation dans Anroid



Utiliser une petite image transparente appelée empty.png

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

Rend la taille de la police de l'étiquette plus petite dans iOS

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

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

Lire Comportement spécifique à la plate-forme en ligne: https://riptutorial.com/fr/xamarin-forms/topic/6636/comportement-specifique-a-la-plate-forme

Chapitre 10: Contact Picker - Formulaires Xamarin (Android et iOS)

Remarques

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

Lire Contact Picker - Formulaires Xamarin (Android et iOS) en ligne: https://riptutorial.com/fr/xamarin-forms/topic/6659/contact-picker---formulaires-xamarin--android-et-ios-

Chapitre 11: Création de contrôles personnalisés

Examples

Créez un contrôle d'entrée personnalisé Xamarin Forms (aucun élément natif requis)

Vous trouverez ci-dessous un exemple de contrôle personnalisé Xamarin Forms pur. Aucun rendu personnalisé n'est fait pour cela, mais pourrait facilement être implémenté, en fait, dans mon propre code, j'utilise ce même contrôle avec un moteur de rendu personnalisé pour le `Label` et l' `Entry` .

Le contrôle personnalisé est un `ContentView` avec une `Label` , une `Entry` et un `BoxView` , qui est maintenu en place à l'aide de 2 `StackLayout` . Nous définissons également plusieurs propriétés pouvant être liées, ainsi qu'un événement `TextChanged` .

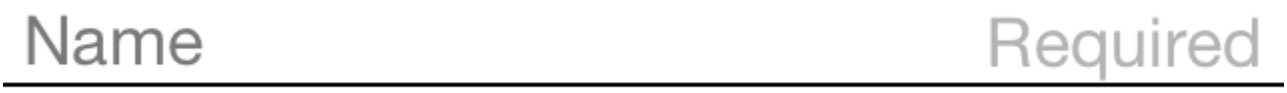
Les propriétés personnalisables personnalisables sont définies comme elles sont ci-dessous et les éléments du contrôle (dans ce cas, une `Label` et une `Entry`) sont liés aux propriétés personnalisables. Quelques-unes des propriétés pouvant être liées doivent également implémenter un `BindingPropertyChangedDelegate` afin que les éléments liés changent leurs valeurs.

```
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;
    }
}
```

Et voici une image du produit final sur iOS (l'image montre à quoi elle ressemble lors de l'utilisation d'un moteur de rendu personnalisé pour l' `Label` et l' `Entry` utilisées pour supprimer la bordure sur iOS et pour spécifier une police personnalisée pour les deux éléments):



Un problème que j'ai rencontré `BoxView.BackgroundColor` à faire changer `BoxView.BackgroundColor` lorsque `UnderlineColor` changeait. Même après la liaison du `BoxView` de `BackgroundColor` propriété, cela ne changerait pas jusqu'à ce que j'ajouté le `UnderlineColorChanged` délégué.

Étiquette avec une collection pouvant être liée de Spans

J'ai créé une étiquette personnalisée avec un wrapper autour de la propriété `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;
    }
}
```

J'ai ajouté une collection de `TextComponent` personnalisés:

```
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); }
    }
}
```

Et lorsque la collection de composants de texte change ou que la propriété `Text` d'un composant distinct change, je reconstruit la propriété `FormattedText` de l' `Label` de base.

Et comment je l'ai utilisé dans `XAML` :

```
<ContentPage x:Name="Page"
         xmlns="http://xamarin.com/schemas/2014/forms"
         xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
         xmlns:controls="clr-namespace:SuperForms.Controls;assembly=SuperForms.Controls"
         x:Class="SuperForms.Samples.MultiComponentLabelPage">
<controls:MultiComponentLabel Margin="0,20,0,0">
    <controls:MultiComponentLabel.Components>
      <controls:TextComponent Text="Time"/>
      <controls:TextComponent Text=": "/>
      <controls:TextComponent Text="{Binding CurrentTime, Source={x:Reference Page}}"/>
    </controls:MultiComponentLabel.Components>
  </controls:MultiComponentLabel>
</ContentPage>
```

Codebehind de la page:

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

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

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

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

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

Création d'un contrôle d'entrée personnalisé avec une propriété MaxLength

Le contrôle Xamarin Forms `Entry` ne possède pas de propriété `MaxLength` . Pour ce faire, vous pouvez étendre `Entry` comme ci-dessous, en ajoutant une propriété Bindable `MaxLength` . Ensuite, il vous suffit de vous inscrire à l'événement `TextChanged` sur `Entry` et de valider la longueur du `Text` à l'appel suivant:

```
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;
    }
}
```

Utilisation dans 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>
```

Lire Création de contrôles personnalisés en ligne: https://riptutorial.com/fr/xamarin-forms/topic/3913/creation-de-controles-personnalises

Chapitre 12: Création de contrôles personnalisés

Introduction

Chaque vue `Xamarin.Forms` est accompagnée d'un rendu pour chaque plate-forme qui crée une instance d'un contrôle natif. Lorsqu'une vue est rendue sur la plate-forme spécifique, la classe `ViewRenderer` est instanciée.

Le processus pour ce faire est le suivant:

Créez un contrôle personnalisé Xamarin.Forms.

Consommez le contrôle personnalisé de Xamarin.Forms.

Créez le moteur de rendu personnalisé pour le contrôle sur chaque plate-forme.

Examples

Implémenter un contrôle CheckBox

Dans cet exemple, nous allons implémenter une case à cocher personnalisée pour Android et iOS.

Création du contrôle personnalisé

```
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);
        }
    }
}
```

Nous commencerons par le moteur de rendu personnalisé Android en créant une nouvelle classe (`CheckboxCustomRenderer`) dans la partie `Android` de notre solution.

Quelques détails importants à noter:

- Nous devons marquer le haut de notre classe avec l'attribut ExportRenderer pour que le rendu soit enregistré avec `Xamarin.Forms` . De cette façon, `Xamarin.Forms` utilisera ce moteur de rendu lorsqu'il `Xamarin.Forms` de créer notre objet `Checkbox` sur `Android` .
- Nous faisons la plupart de notre travail dans la méthode `OnElementChanged` , où nous instancions et configurons notre contrôle natif.

Consommer le contrôle personnalisé

```
<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>
```

Création du moteur de rendu personnalisé sur chaque plate-forme

Le processus de création de la classe de rendu personnalisé est le suivant:

1. Créez une sous-classe de la classe `ViewRenderer<T1,T2>` qui affiche le contrôle personnalisé. Le premier argument de type doit être le contrôle personnalisé utilisé par le moteur de rendu, dans ce cas, `CheckBox` . Le deuxième argument de type doit être le contrôle natif qui implémentera le contrôle personnalisé.

2. Remplacez la méthode `OnElementChanged` qui `OnElementChanged` le contrôle personnalisé et écrit la logique pour le personnaliser. Cette méthode est appelée lorsque le contrôle `Xamarin.Forms` correspondant est créé.
3. Ajoutez un attribut `ExportRenderer` à la classe de rendu personnalisée pour indiquer qu'elle sera utilisée pour effectuer le rendu du contrôle personnalisé `Xamarin.Forms` . Cet attribut est utilisé pour enregistrer le moteur de rendu personnalisé avec `Xamarin.Forms` .

Création du moteur de rendu personnalisé pour 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;
            }
        }
    }
}
```

Création du moteur de rendu personnalisé pour iOS

Comme dans iOS, la case à cocher n'est pas intégrée, nous allons d'abord créer un `CheckBoxView` puis créer un moteur de rendu pour notre case à cocher Xamarin.Forms.

Le `CheckBoxView` est basé sur deux images, checked_checkbox.png et unchecked_checkbox.png, afin que la propriété `Color` soit ignorée.

La vue 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);
            }
        }
```

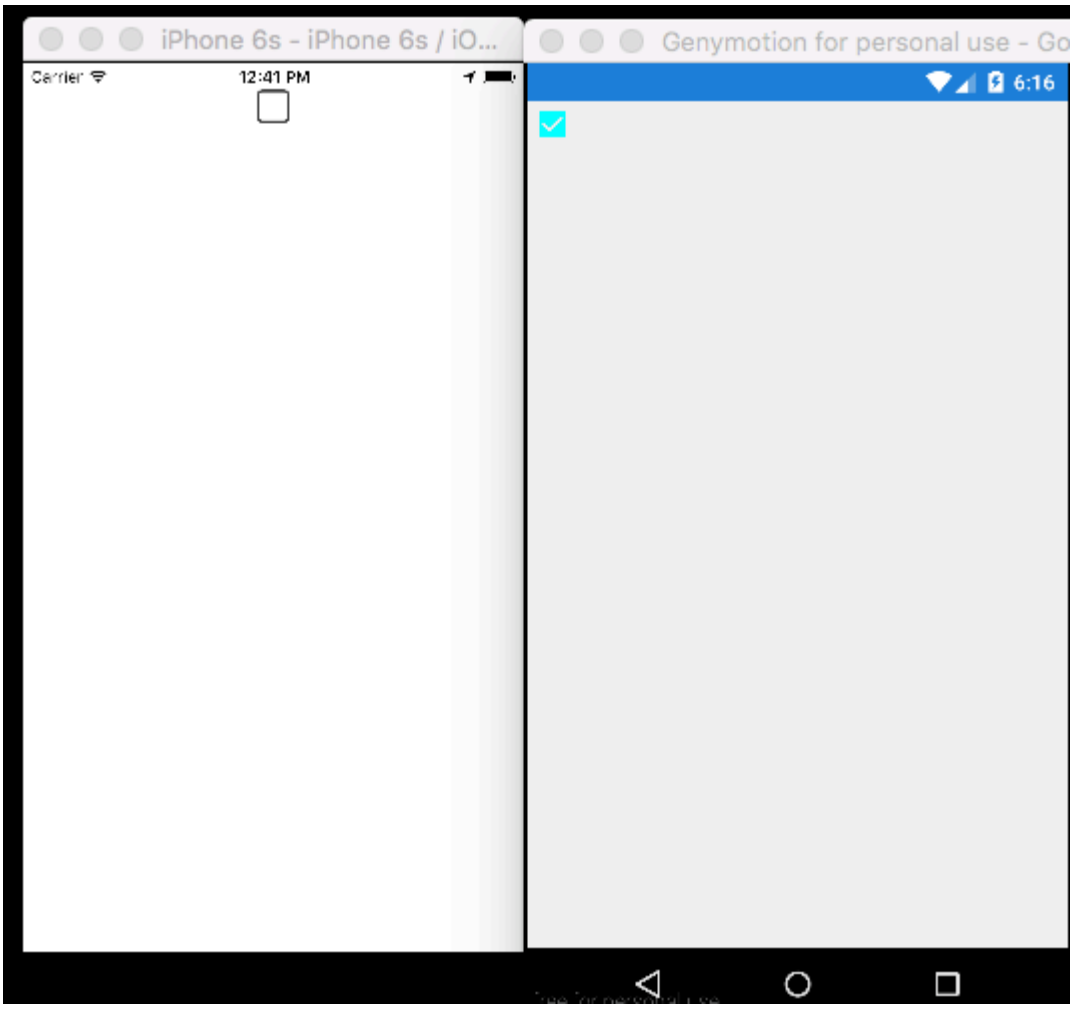
```
        public string UncheckedTitle
        {
            set
            {
                SetTitle(value, UIControlState.Normal);
            }
        }

        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);
        }
    }
}
```

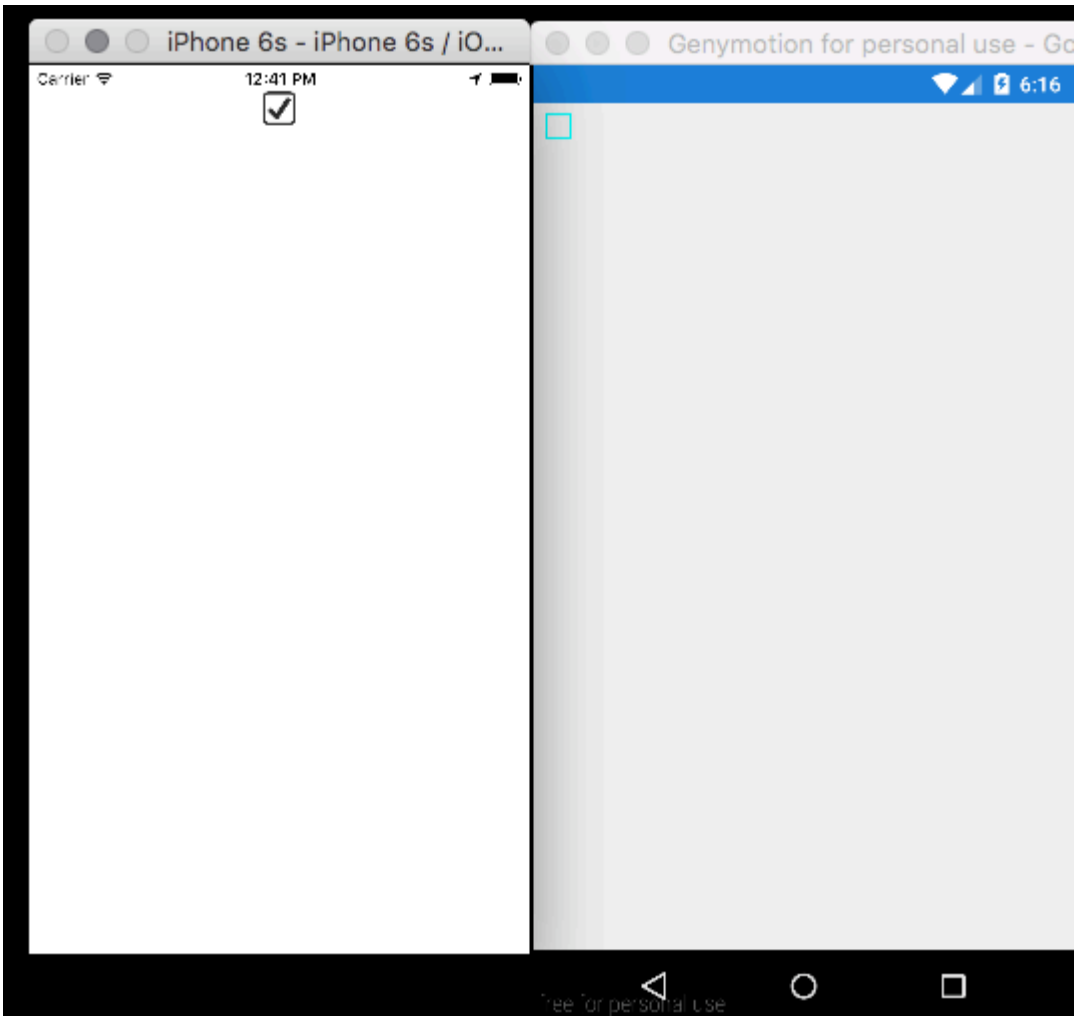
Le moteur de rendu personnalisé CheckBox:

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

Résultat:


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



Lire Création de contrôles personnalisés en ligne: https://riptutorial.com/fr/xamarin-forms/topic/5975/creation-de-controles-personnalises

Chapitre 13: Création de contrôles personnalisés

Examples

Création d'un bouton personnalisé

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

Utilisation dans 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>
```

Maintenant, vous pouvez utiliser vos propriétés dans c #:

```
exampleControl.IntData
```

Notez que vous devez spécifier vous-même où votre classe TurboButton est placée dans votre projet. Je l'ai fait dans cette ligne:

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

Vous pouvez librement modifier "customControls" sous un autre nom. C'est à vous de voir comment vous allez l'appeler.

Lire Création de contrôles personnalisés en ligne: https://riptutorial.com/fr/xamarin-forms/topic/6592/creation-de-controles-personnalises

Chapitre 14: Cycle de vie de l'application Xamarin.Forms générique? Plate-forme dépendante!

Examples

Le cycle de vie de Xamarin.Forms n'est pas le cycle de vie réel d'une application, mais une représentation multiplateforme de celui-ci.

Jetons un coup d'œil aux méthodes de cycle de vie des applications natives pour différentes plates-formes.

Android.

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

iOS

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

Les fenêtres.

```
//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;
```

Et maintenant, les méthodes du cycle de vie de l'application **Xamarin.Forms** :

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

Ce que vous pouvez facilement constater en observant simplement les listes, la perspective du cycle de vie des applications multiplates-formes de Xamarin.Forms est grandement simplifiée. Il vous donne une idée générique sur l'état de votre application, mais dans la plupart des cas, vous devrez créer une logique dépendant de la plate-forme.

Lire Cycle de vie de l'application Xamarin.Forms générique? Plate-forme dépendante! en ligne: https://riptutorial.com/fr/xamarin-forms/topic/8329/cycle-de-vie-de-l-application-xamarin-forms-generique--plate-forme-dependante-

Chapitre 15: Déclencheurs et comportements

Examples

Exemple de déclencheur de formulaires Xamarin

`Trigger` sont un moyen simple d'ajouter une certaine réactivité à votre application. Un moyen facile de le faire est d'ajouter un `Trigger` qui modifie une `Label` de `TextColor` selon que son connexe `Entry` est le texte entré dans ou non.

L'utilisation d'un `Trigger` pour cela permet à `Label.TextColor` de passer du gris (quand aucun texte n'est entré) au noir (dès que l'utilisateur entre du texte):

Convertisseur (chaque convertisseur reçoit une variable d' `Instance` utilisée dans la liaison pour qu'une nouvelle instance de la classe ne soit pas créée à chaque utilisation):

```
/// <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 (le code XAML utilise le nom `x:Name` de l' `Entry` à comprendre dans la propriété `Entry.Text` est long de plus de 3 caractères):

```
<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 déclencheurs

MultiTrigger n'est pas nécessaire fréquemment mais il y a certaines situations où il est très pratique. MultiTrigger se comporte de la même manière que Trigger ou DataTrigger, mais il a plusieurs conditions. Toutes les conditions doivent être remplies pour qu'un Setters tire. Voici un exemple 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>
```

L'exemple a deux entrées différentes, téléphone et email, et l'une d'entre elles doit être remplie.

Le MultiTrigger désactive le bouton de soumission lorsque les deux champs sont vides.

Lire Déclencheurs et comportements en ligne: https://riptutorial.com/fr/xamarin-forms/topic/6012/declencheurs-et-comportements

Chapitre 16: DependencyService

Remarques

Lorsque vous utilisez `DependencyService` vous avez généralement besoin de trois parties:

- **Interface** - Définit les fonctions que vous souhaitez abstraire.
- **Implémentation de la plate-forme** - Classe de chaque projet spécifique à la plate-forme qui implémente l'interface définie précédemment.
- **Registration** - Chaque classe d'implémentation spécifique à la plate-forme doit être enregistrée avec `DependencyService` via un attribut de métadonnées. Cela permet à `DependencyService` de trouver votre implémentation au moment de l'exécution.

Lorsque vous utilisez `DependencyService` vous devez fournir une implémentation pour chaque plate-forme que vous ciblez. Lorsqu'une implémentation n'est pas fournie, l'application échouera au moment de l'exécution.

Examples

Interface

L'interface définit le comportement que vous souhaitez exposer via le service `DependencyService` . Un exemple d'utilisation d'un `DependencyService` est un service Text-To-Speech. Il n'y a actuellement aucune abstraction pour cette fonctionnalité dans Xamarin.Forms, vous devez donc créer les vôtres. Commencez par définir une interface pour le comportement:

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

Parce que nous définissons notre interface, nous pouvons le coder à partir de notre code partagé.

Remarque: Les classes qui implémentent l'interface doivent disposer d'un constructeur sans paramètre pour fonctionner avec `DependencyService` .

mise en œuvre iOS

L'interface que vous avez définie doit être implémentée dans chaque plate-forme ciblée. Pour iOS, cela se fait via le framework `AVFoundation` . L'implémentation suivante de l'interface `ITextToSpeech` gère la `ITextToSpeech` charge d'un texte donné en anglais.

```
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);
    }
}
```

Lorsque vous avez créé votre classe, vous devez activer `DependencyService` pour le découvrir au moment de l'exécution. Cela se fait en ajoutant un attribut `[assembly]` au-dessus de la définition de classe et en dehors de toute définition d'espace de noms.

```
using AVFoundation;
using DependencyServiceSample.iOS;

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

Cet attribut enregistre la classe avec `DependencyService` afin de pouvoir l'utiliser lorsqu'une instance de l'interface `ITextToSpeech` est requise.

Code partagé

Une fois que vous avez créé et enregistré vos classes spécifiques à la plate-forme, vous pouvez les connecter à votre code partagé. La page suivante contient un bouton qui déclenche la fonctionnalité de synthèse vocale à l'aide d'une phrase prédéfinie. Il utilise `DependencyService` pour récupérer une implémentation spécifique à la plate-forme de `ITextToSpeech` au moment de l'exécution en utilisant les SDK natifs.

```
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;
}
```

Lorsque vous exécutez cette application sur un appareil iOS ou Android et que vous appuyez sur le bouton, vous entendrez l'application parler la phrase donnée.

Implémentation Android

L'implémentation spécifique à Android est un peu plus complexe car elle vous oblige à hériter d'un `Java.Lang.Object` natif et vous oblige à implémenter l'interface `IOnInitListener` . Android exige que vous fournissiez un contexte Android valide pour un grand nombre des méthodes SDK qu'il expose. Xamarin.Forms expose un objet `Forms.Context` qui vous fournit un contexte Android que vous pouvez utiliser dans de tels cas.

```
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
}
```

Lorsque vous avez créé votre classe, vous devez activer `DependencyService` pour le découvrir au moment de l'exécution. Cela se fait en ajoutant un attribut `[assembly]` au-dessus de la définition de classe et en dehors de toute définition d'espace de noms.

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


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

Cet attribut enregistre la classe avec `DependencyService` afin de pouvoir l'utiliser lorsqu'une instance de l'interface `ITextToSpeech` est requise.

Lire DependencyService en ligne: https://riptutorial.com/fr/xamarin-forms/topic/2508/dependencyservice

Chapitre 17: Effets

Introduction

Effects simplifie les personnalisations spécifiques à la plateforme. Lorsqu'il est nécessaire de modifier les propriétés d'un Xamarin Forms Control, vous pouvez utiliser Effects. Lorsqu'il est nécessaire de remplacer les méthodes de Xamarin Forms Control, des moteurs de rendu personnalisés peuvent être utilisés

Examples

Ajout d'un effet spécifique à la plate-forme pour un contrôle d'entrée

1. Créez une nouvelle application Xamarin Forms à l'aide de PCL File -> New Solution -> Multiplatform App -> Xamarin Forms -> Forms App; Nommez le projet comme `EffectsDemo`
2. Sous le projet iOS, ajoutez une nouvelle classe `Effect` qui hérite de la classe `PlatformEffect` et remplace les méthodes `OnAttached` , `OnDetached` et `OnElementPropertyChanged` Notez les deux attributs `ResolutionGroupName` et `ExportEffect` , nécessaires pour `ExportEffect` cet effet du projet PCL / shared.

- `OnAttached` est la méthode où la logique de la personnalisation va dans
- `OnDetached` est la méthode par laquelle le nettoyage et le `OnDetached` se produisent
- `OnElementPropertyChanged` est la méthode qui est déclenchée lors des modifications de propriété de différents éléments. Pour identifier la bonne propriété, vérifiez la modification exacte de la propriété et ajoutez votre logique. Dans cet exemple, `OnFocus` donnera la couleur `Blue` et `OutofFocus` donnera `Red` couleur `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);
     }
 }
```

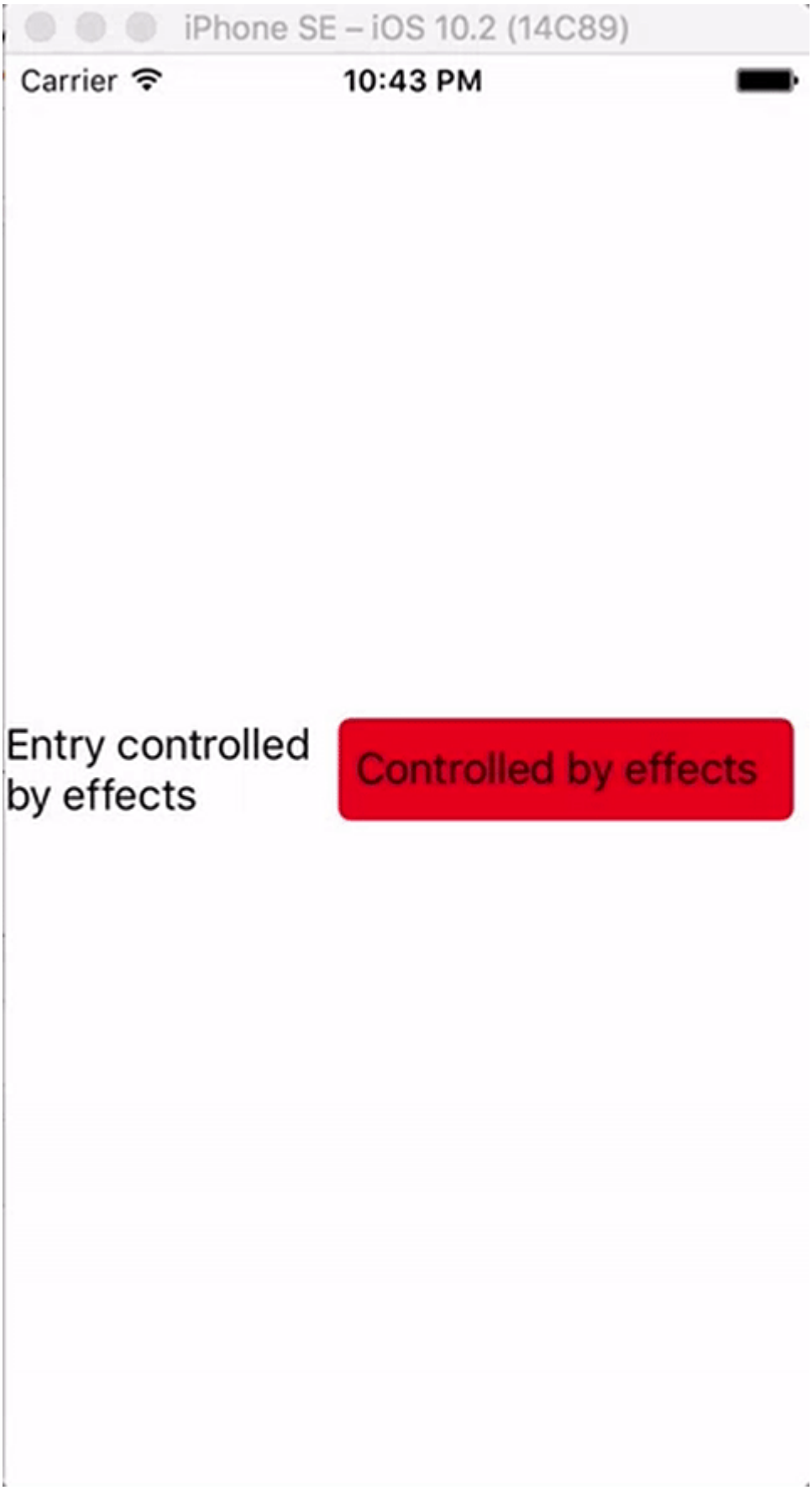
}}

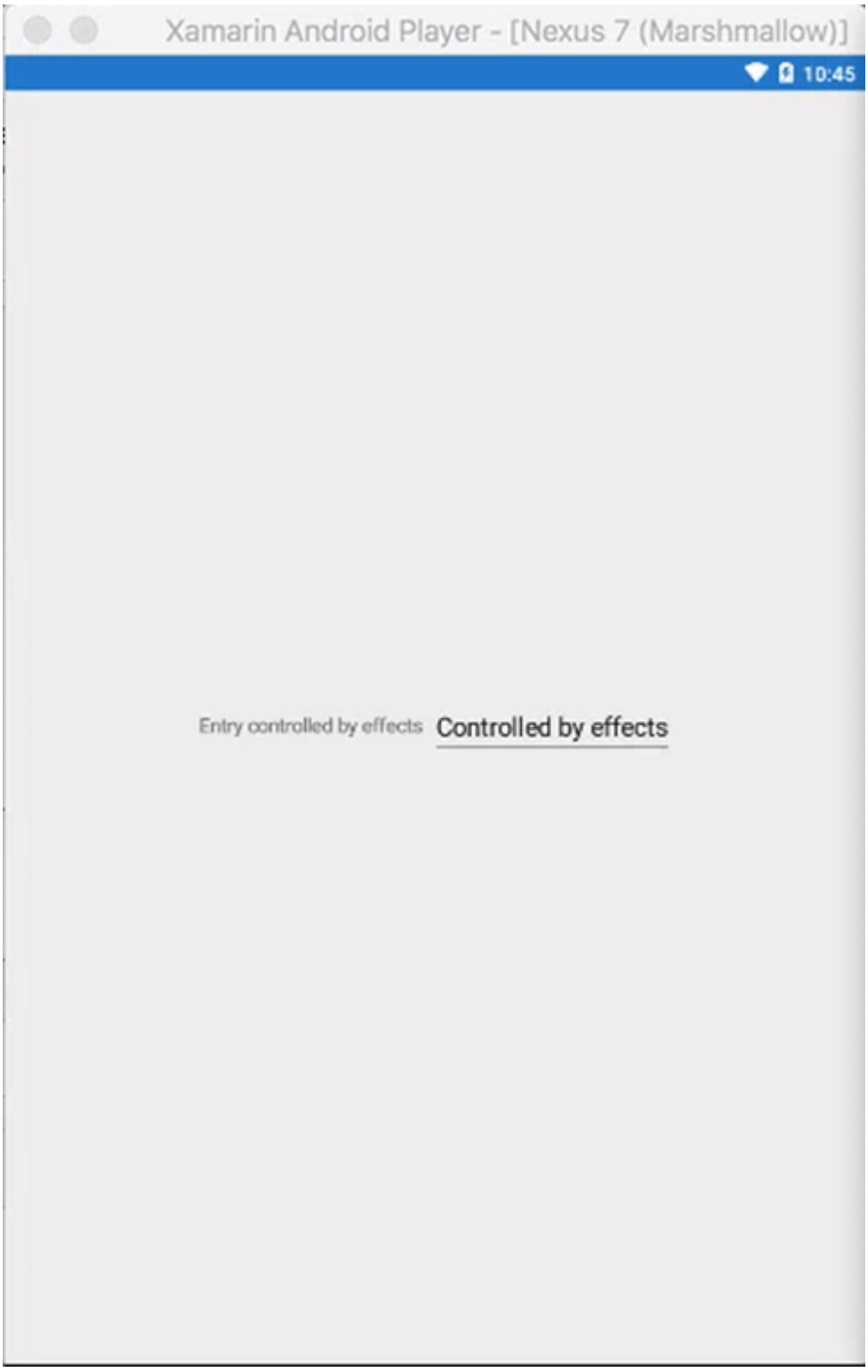
3. Pour consommer cet effet dans l'application, sous le projet `PCL` , créez une nouvelle classe nommée `FocusEffect` qui hérite de `RoutingEffect` . Ceci est essentiel pour que le PCL instancie la mise en œuvre spécifique de la plate-forme de l'effet. Exemple de code ci-dessous:

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

4. Ajouter l'effet au contrôle d' `Entry` dans le 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



Étant donné que l'effet a été implémenté uniquement dans la version iOS, lorsque l'application s'exécute dans `iOS Simulator` en concentrant les changements de couleur d'arrière-plan `Entry` et que rien ne se produit dans `Android Emulator` car l' `Effect` n'a pas été créé dans le projet `Droid`

Lire Effets en ligne: https://riptutorial.com/fr/xamarin-forms/topic/9252/effets

Chapitre 18: Geste Xamarin

Examples

Touchez le geste

Avec le geste tap, vous pouvez faire en sorte que tout élément de l'interface utilisateur soit cliquable (images, boutons, StackLayouts, ...):

(1) En code, en utilisant l'événement:

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

(2) En code, en utilisant `ICommand` (avec MVVM-Pattern , par exemple):

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

(3) Ou en Xaml (avec event et `ICommand` , un seul est nécessaire):

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

Lire Geste Xamarin en ligne: https://riptutorial.com/fr/xamarin-forms/topic/7994/geste-xamarin

Chapitre 19: Geste Xamarin

Examples

Événement gestuel

Lorsque nous mettons le contrôle de Label, le Label ne fournit aucun événement. <Label x: Name = "lblSignUp Text =" Vous n'avez pas de compte? "/> Comme indiqué dans le but d'affichage uniquement Label.

Lorsque l'utilisateur souhaite remplacer Button par une étiquette, nous donnons l'événement pour Label. Comme indiqué ci-dessous:

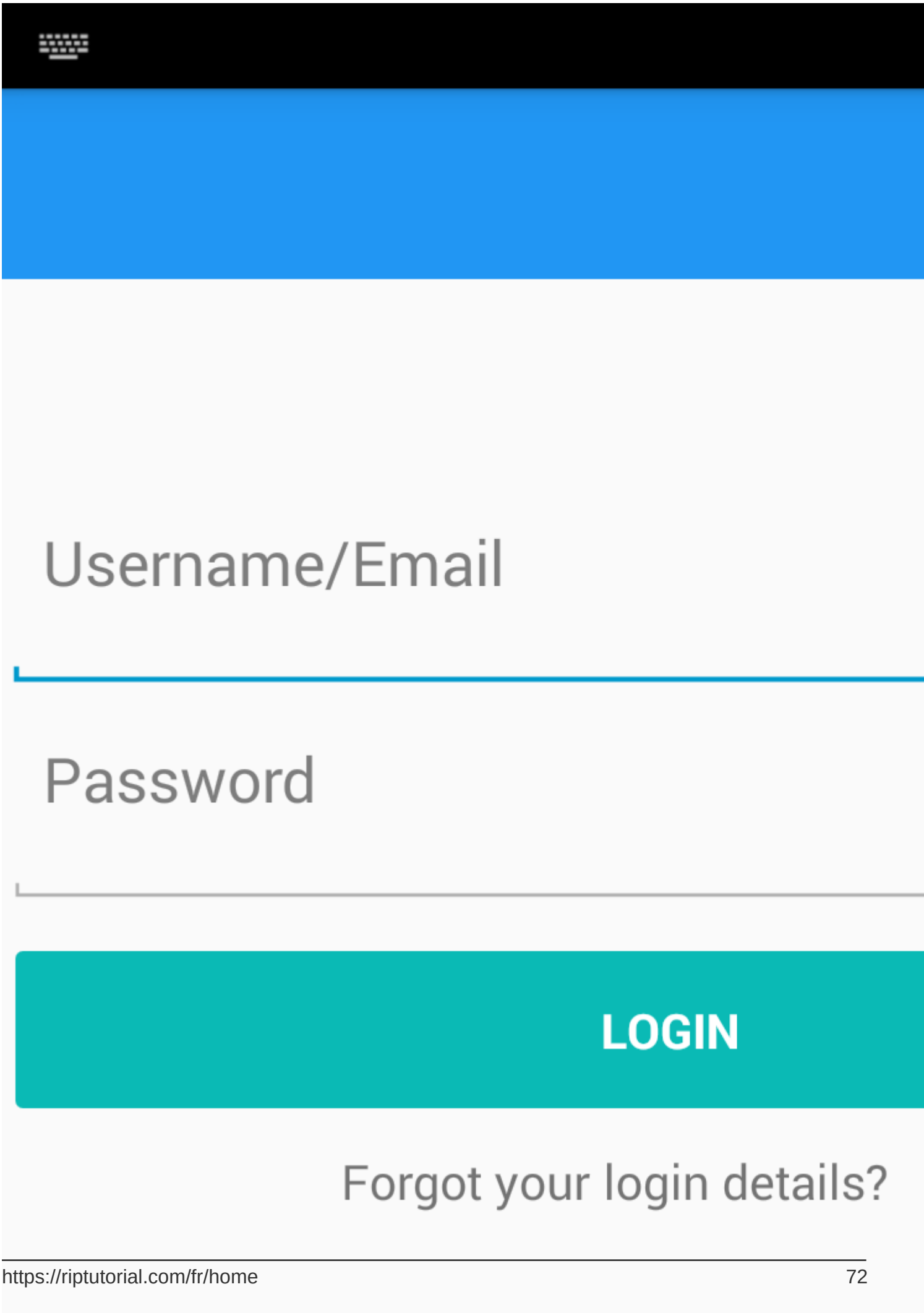
XAML

```
<Label x:Name="lblSignUp" Text="Don't have an account?" Grid.Row="8" Grid.Column="1"
Grid.ColumnSpan="2">
  <Label.GestureRecognizers>
    <TapGestureRecognizer
   Tapped="lblSignUp_Tapped"/>
  </Label.GestureRecognizers>
```

C #

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

L'écran ci-dessous montre l'événement Label. Écran 1: L'étiquette "Vous n'avez pas de compte?" comme indiqué en bas.


Username/Email
Password
LOGIN
Forgot your login details?

Chapitre 20: Gestes

Examples

Faire une image tappable en ajoutant un TapGestureRecognizer

Il existe deux types de reconnaissance par défaut disponibles dans Xamarin.Forms, l'un d'entre eux est le `TapGestureRecognizer` .

Vous pouvez les ajouter à pratiquement n'importe quel élément visuel. Jetez un oeil à une implémentation simple qui se lie à une `Image` . Voici comment le faire en code.

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

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

Ou en XAML:

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

Ici, la commande est définie à l'aide de la liaison de données. Comme vous pouvez le voir, vous pouvez également définir le `NumberOfTapsRequired` pour l'activer pour plus de taps avant d'agir. La valeur par défaut est 1 tapotement.

Les autres gestes sont Pinch et Pan.

Zoomer une image avec le geste de pincement

Afin de rendre une `Image` (ou tout autre élément visuel) zoomable, nous devons lui ajouter un `PinchGestureRecognizer` . Voici comment le faire en code:

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

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

Mais cela peut aussi être fait à partir de XAML:

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

Dans le gestionnaire d'événement accompagné, vous devez fournir le code pour zoomer sur votre image. Bien entendu, d'autres utilisations peuvent également être mises en œuvre.

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

Les autres gestes sont Tap et Pan.

Afficher tout le contenu de l'image zoomée avec le PanGestureRecognizer

Lorsque vous avez une `Image` agrandie (ou un autre contenu), vous pouvez faire glisser l' `Image` pour afficher tout son contenu dans l'état agrandi.

Cela peut être réalisé en implémentant le PanGestureRecognizer. Du code cela ressemble à ça:

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

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

Cela peut également être fait à partir de XAML:

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

Dans l'événement code-behind, vous pouvez maintenant gérer le panoramique en conséquence. Utilisez cette signature de méthode pour le gérer:

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

Placez une épingle à l'endroit où l'utilisateur a touché l'écran avec MR.Gestures

Les détecteurs de gestes intégrés Xamarins ne fournissent qu'une manipulation tactile très simple. Par exemple, il n'y a aucun moyen d'obtenir la position d'un doigt touchant. MR.Gestures est un

composant qui ajoute 14 événements de traitement tactile différents. La position des doigts qui se touchent fait partie de `EventArgs` transmis à tous les événements MR.Gestures.

Si vous souhaitez placer une épingle n'importe où sur l'écran, le plus simple est d'utiliser un `MR.Gestures.AbsoluteLayout` qui gère l'événement `Tapping` .

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

Comme vous pouvez le voir, `Tapping="OnTapping"` ressemble plus à la syntaxe .NET que Xamarins avec les `GestureRecognizers` imbriqués. Cette syntaxe a été copiée depuis iOS et ça sent un peu pour les développeurs .NET.

Dans votre code, vous pouvez ajouter le gestionnaire `OnTapping` comme `OnTapping` :

```
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);
    }
}
```

Au lieu de l'événement `Tapping` , vous pouvez également utiliser la commande `TappingCommand` et la lier à votre ViewModel, mais cela compliquerait les choses dans cet exemple simple.

Vous trouverez d'autres exemples pour MR.Gestures dans l' application GestureSample sur GitHub et sur le site Web de MR.Gestures . Ils montrent également comment utiliser tous les autres événements tactiles avec les gestionnaires d'événements, les commandes, MVVM, ...

Lire Gestes en ligne: https://riptutorial.com/fr/xamarin-forms/topic/3914/gestes

Chapitre 21: Gestion des exceptions

Examples

Une façon de signaler les exceptions sur iOS

Accédez au fichier `Main.cs` dans le **projet iOS** et modifiez le code existant, comme présenté ci-dessous:

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

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

`ILittleWatson` interface `ILittleWatson` , utilisée dans le code portable, pourrait ressembler à ceci:

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

    void SaveReport(Exception ex);
}
```

Implémentation pour **projet 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);
        }
    }
}
```

Et puis, quelque part, là où l'application démarre, mettez:

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

Lire Gestion des exceptions en ligne: https://riptutorial.com/fr/xamarin-forms/topic/6428/gestion-des-exceptions

Chapitre 22: Liaison de données

Remarques

Exceptions possibles

System.ArrayTypeMismatchException: Vous avez tenté d'accéder à un élément en tant que type incompatible avec le tableau.

Cette exception peut se produire lors de la tentative de liaison d'une collection à une propriété non-liée lorsque la pré-compilation XAML est activée. Un exemple courant est la tentative de liaison à `Picker.Items` . Voir ci-dessous.

System.ArgumentException: objet de type 'Xamarin.Forms.Binding' ne peut pas être converti en type 'System.String'.

Cette exception peut se produire lorsque vous tentez de lier une collection à une propriété non-liée lorsque la pré-compilation XAML est désactivée. Un exemple courant est la tentative de liaison à `Picker.Items` . Voir ci-dessous.

La `Picker.Items` propriété est pas Bindable

Ce code provoquera une erreur:

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

L'exception peut être l'une des suivantes:

> System.ArrayTypeMismatchException: Vous avez tenté d'accéder à un élément en tant que type incompatible avec le tableau.

ou

> System.ArgumentException: objet de type 'Xamarin.Forms.Binding' ne peut pas être converti en type 'System.String'.

Plus précisément, la propriété `Items` est non-bindable. Les solutions incluent la création de votre propre contrôle personnalisé ou l'utilisation d'un contrôle tiers, tel que `BindablePicker` partir de

FreshEssentials . Après avoir installé le package FreshEssentials NuGet dans votre projet, le contrôle `BindablePicker` du `BindablePicker` avec une propriété `ItemsSource` pouvant être `ItemsSource` est 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

Liaison de base à 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));
        }
    }
}
```

Lire Liaison de données en ligne: https://riptutorial.com/fr/xamarin-forms/topic/3915/liaison-de-donnees

Chapitre 23: MessagingCenter

Introduction

Xamarin.Forms dispose d'un mécanisme de messagerie intégré pour promouvoir le code découplé. De cette façon, les modèles de vue et les autres composants n'ont pas besoin de se connaître. Ils peuvent communiquer via un simple contrat de messagerie.

Il existe essentiellement deux ingrédients principaux pour utiliser le `MessagingCenter` .

S'abonner écoutez les messages avec une certaine signature (le contrat) et exécutez le code lorsqu'un message est reçu. Un message peut avoir plusieurs abonnés.

Envoyer ; envoyer un message aux abonnés pour qu'ils agissent.

Examples

Exemple simple

Nous verrons ici un exemple simple d'utilisation de MessagingCenter dans Xamarin.Forms.

Tout d'abord, regardons l'abonnement à un message. Dans le modèle `FooMessaging` , nous sommes abonnés à un message provenant de `MainPage` . Le message doit être "Hi" et lorsque nous le recevons, nous enregistrons un gestionnaire qui définit la propriété `Greeting` . Enfin, `this` signifie que l'instance actuelle de `FooMessaging` s'enregistre pour ce message.

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

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

Pour envoyer un message déclenchant cette fonctionnalité, nous devons avoir une page appelée `MainPage` et implémenter le code comme ci-dessous.

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

Dans notre `MainPage` nous avons un bouton avec un gestionnaire qui envoie un message. `this` devrait être une instance de `MainPage` .

Arguments de passage

Vous pouvez également transmettre des arguments avec un message avec lequel travailler.

Nous utiliserons les classifiés de notre exemple précédent et les étendrons. Dans la partie réceptrice, juste derrière l'appel de la méthode `Subscribe` , ajoutez le type d'argument attendu. Assurez-vous également de déclarer également les arguments dans la signature du gestionnaire.

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

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

Lors de l'envoi d'un message, veillez à inclure la valeur de l'argument. En outre, vous ajoutez ici le type juste derrière la méthode `Send` et ajoutez la valeur de l'argument.

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

Dans cet exemple, une chaîne simple est utilisée, mais vous pouvez également utiliser tout autre type d'objets (complexes).

Se désabonner

Lorsque vous n'avez plus besoin de recevoir de messages, vous pouvez simplement vous désabonner. Vous pouvez le faire comme ceci:

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

Lorsque vous fournissez des arguments, vous devez vous désabonner de la signature complète, comme ceci:

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

Lire MessagingCenter en ligne: https://riptutorial.com/fr/xamarin-forms/topic/9672/messagingcenter

Chapitre 24: Mise en cache

Examples

Mise en cache avec Akavache

A propos d'Akavache

Akavache est une bibliothèque incroyablement utile fournissant des fonctionnalités de mise en cache pour la mise en cache de vos données. Akavache fournit une interface de stockage de valeur clé et travaille sur le dessus de SQLite3. Vous n'avez pas besoin de garder votre schéma synchronisé car il ne s'agit en fait que de la solution No-SQL, ce qui le rend parfait pour la plupart des applications mobiles, en particulier si votre application doit être mise à jour souvent sans perte de données.

Recommandations pour Xamarin

Akavache est sans aucun doute la meilleure bibliothèque de mise en cache pour l'application Xamarin si vous n'avez pas besoin d'opérer avec des données fortement relatives, binaires ou de très grandes quantités de données. Utilisez Akavache dans les cas suivants:

- Vous avez besoin de votre application pour mettre en cache les données pendant une période donnée (vous pouvez configurer le délai d'expiration pour chaque entité enregistrée;
- Vous voulez que votre application fonctionne hors connexion;
- Il est difficile de déterminer et de geler le schéma de vos données. Par exemple, vous avez des listes contenant différents objets typés;
- Il suffit que vous ayez un accès simple à la valeur-clé aux données et que vous n'ayez pas besoin de faire des requêtes complexes.

Akavache n'est pas une "solution miracle" pour le stockage des données, alors réfléchissez bien à son utilisation dans les cas suivants:

- Vos entités de données ont de nombreuses relations entre elles;
- Vous n'avez pas vraiment besoin que votre application fonctionne hors ligne;
- Vous avez une énorme quantité de données à enregistrer localement;
- Vous devez migrer vos données de version en version;
- Vous devez effectuer des requêtes complexes typiques pour SQL, comme le regroupement, les projections, etc.

  En fait, vous pouvez migrer manuellement vos données simplement en les lisant et en les écrivant avec des champs mis à jour.

Exemple simple

L'interaction avec Akavache se fait principalement via un objet appelé `BlobCache` .

La plupart des méthodes d'Akavache renvoient des observables réactifs, mais vous pouvez aussi les attendre grâce aux méthodes d'extension.

```
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!"));
```

La gestion des erreurs

```
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()));
```

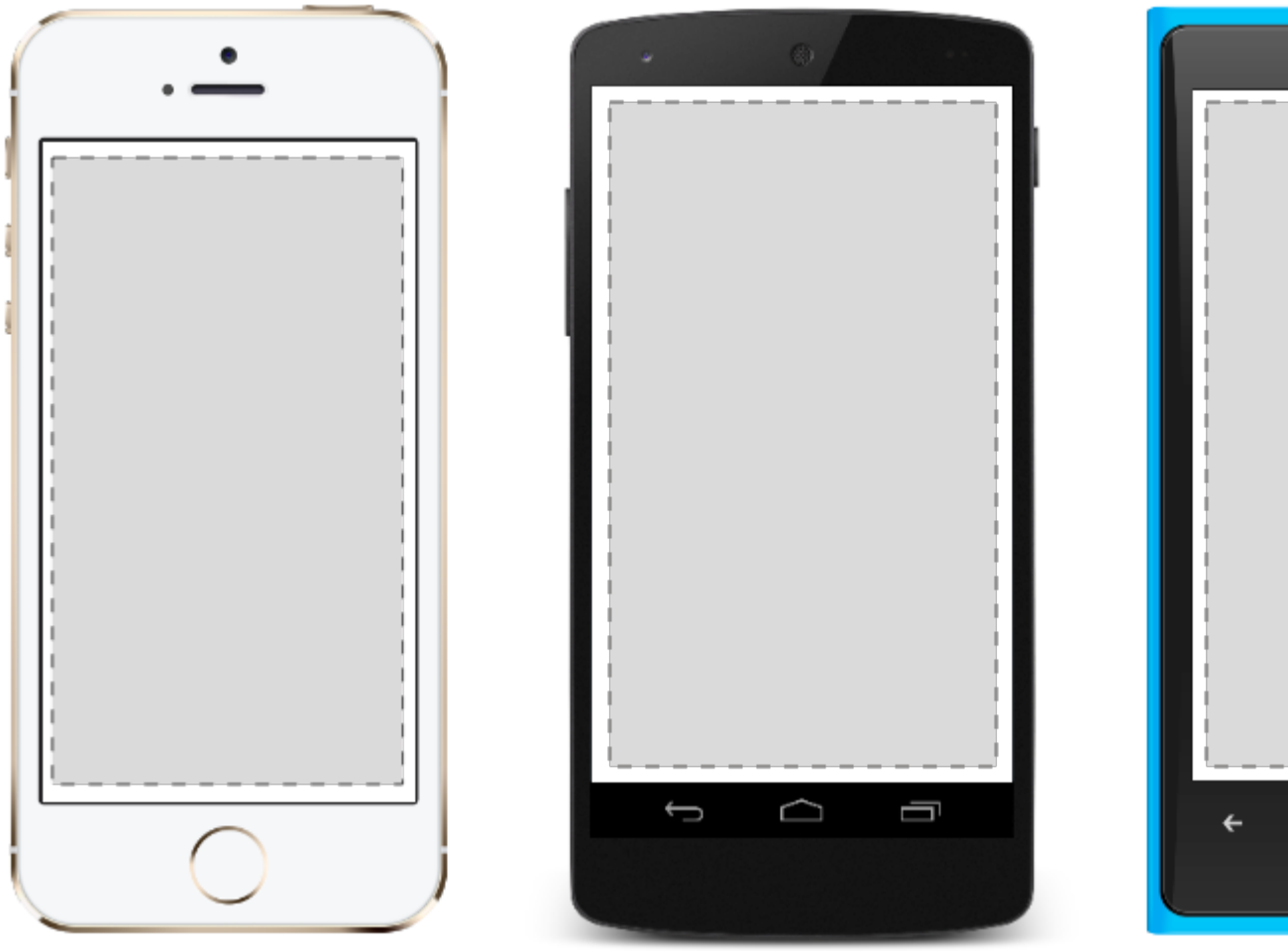
Lire Mise en cache en ligne: https://riptutorial.com/fr/xamarin-forms/topic/3644/mise-en-cache

Chapitre 25: Mise en forme de Xamarin

Examples

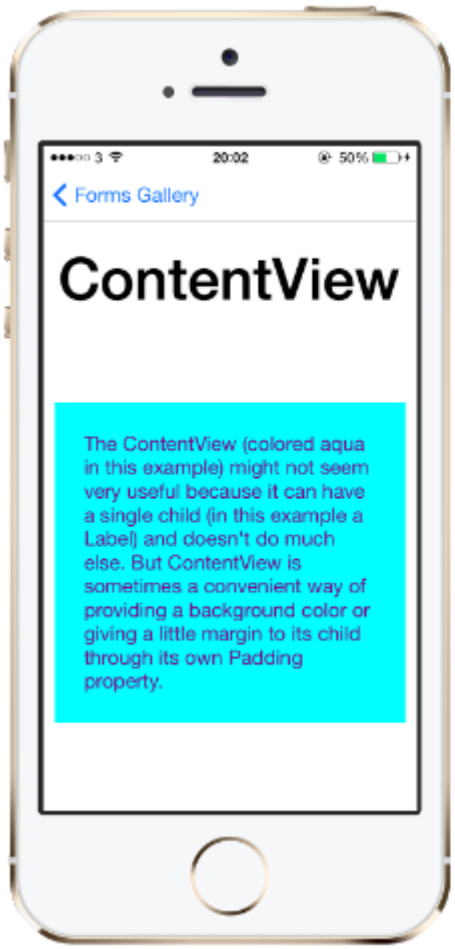
ContentPresenter

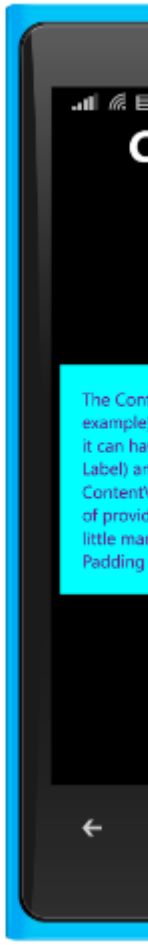
Un gestionnaire de disposition pour les vues modélisées. Utilisé dans un ControlTemplate pour marquer l'emplacement du contenu à présenter.



ContentView

Un élément avec un seul contenu. ContentView a très peu d'utilisation. Son but est de servir de classe de base pour les vues composées définies par l'utilisateur.





XAML

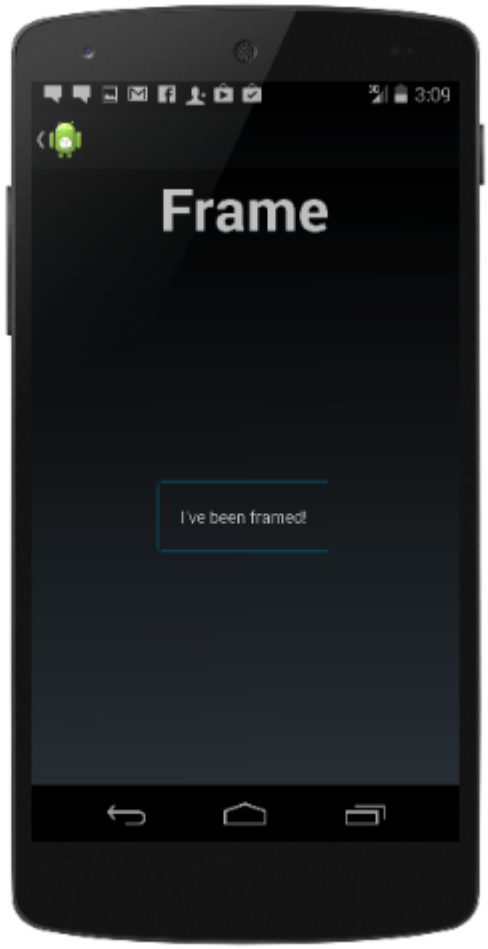
```
<ContentView>
<Label Text="Hi, I'm a simple Label inside of a simple ContentView"
HorizontalOptions="Center"
VerticalOptions="Center"/>
</ContentView>
```

Code

```
var contentView = new ContentView {
Content = new Label {
 Text = "Hi, I'm a simple Label inside of a simple ContentView",
 HorizontalOptions = LayoutOptions.Center,
 VerticalOptions = LayoutOptions.Center
}
};
```

Cadre

Un élément contenant un seul enfant, avec des options de cadrage. Frame ont un défaut Xamarin.Forms.Layout.Padding de 20.



XAML

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

Code

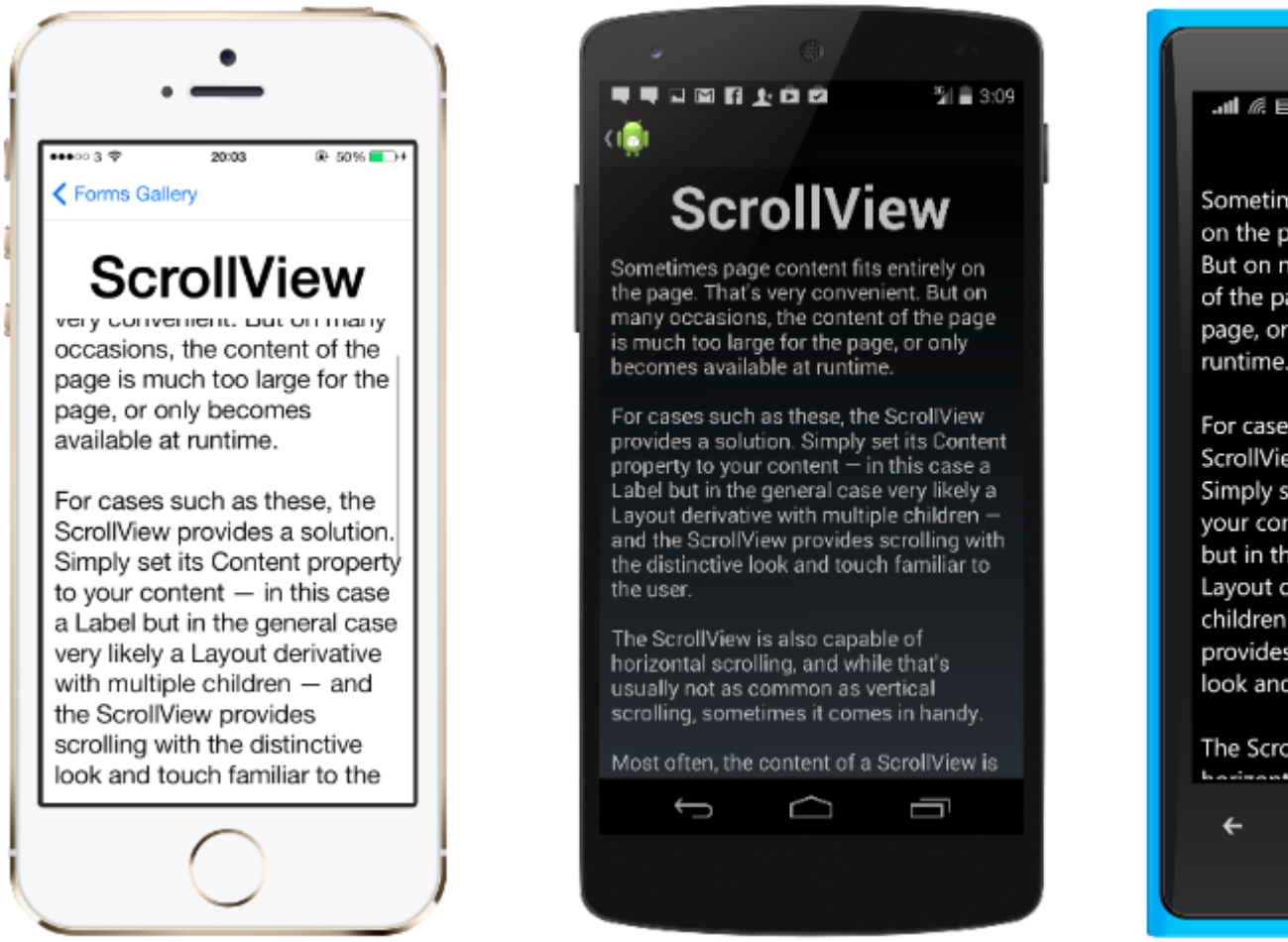
```
var frameView = new Frame {
Content = new Label {
 Text = "I've been framed!",
HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center
},
OutlineColor = Color.Red
};
```

ScrollView

Un élément capable de défiler si son `Content` requis.

`ScrollView` contient des dispositions et leur permet de faire défiler l'écran. `ScrollView` est également utilisé pour permettre aux vues de se déplacer automatiquement dans la partie visible de l'écran

lorsque le clavier est affiché.



Remarque: les `ScrollViews` ne doivent pas être imbriquées. De plus, les `ScrollViews` ne doivent pas être imbriquées avec d'autres contrôles fournissant un défilement, tels que `ListView` et `WebView` .

Un `ScrollView` est facile à définir. Dans XAML:

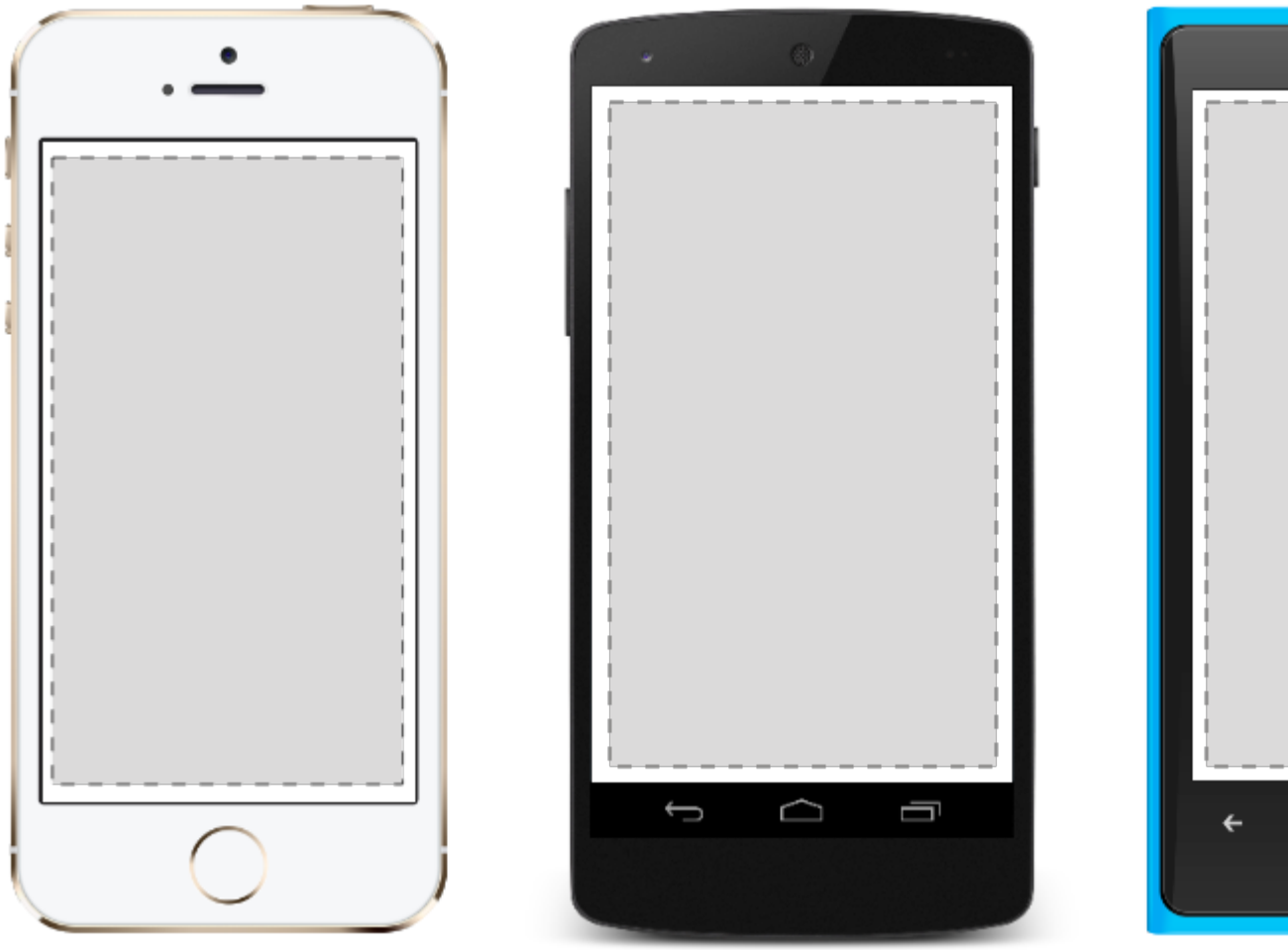
```
<ContentPage.Content>
    <ScrollView>
        <StackLayout>
            <BoxView BackgroundColor="Red" HeightRequest="600" WidthRequest="150" />
            <Entry />
        </StackLayout>
    </ScrollView>
</ContentPage.Content>
```

La même définition dans le code:

```
var scroll = new ScrollView();
Content = scroll;
var stack = new StackLayout();
stack.Children.Add(new BoxView { BackgroundColor = Color.Red,   HeightRequest = 600,
WidthRequest = 600 });
stack.Children.Add(new Entry());
```

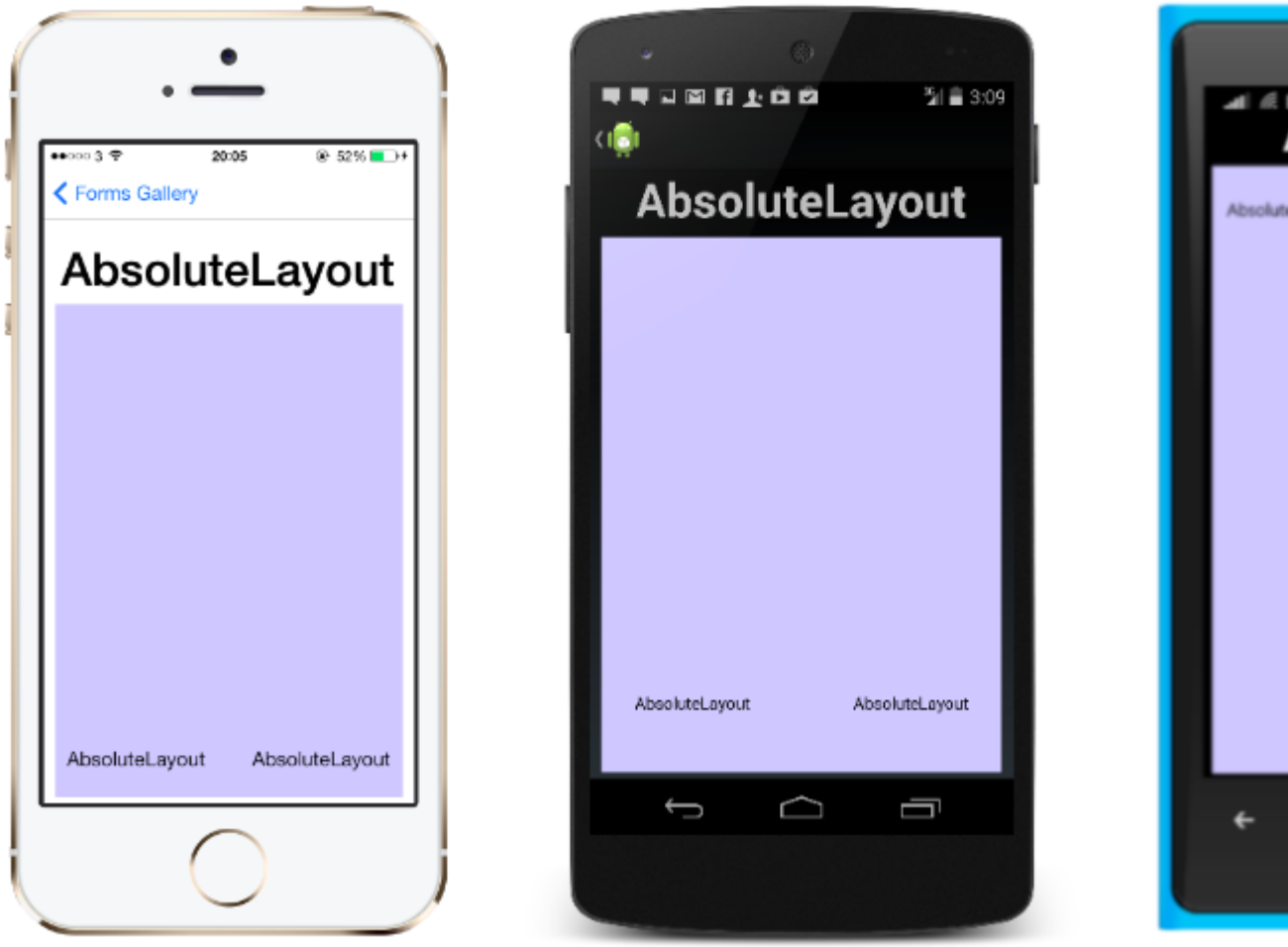
TemplatedView

Un élément qui affiche le contenu avec un modèle de contrôle et la classe de base pour `ContentView` .



AbsoluteLayout

`AbsoluteLayout` positionne et dimensionne les éléments enfants proportionnellement à sa propre taille et position ou à des valeurs absolues. Les vues enfants peuvent être positionnées et dimensionnées à l'aide de valeurs proportionnelles ou de valeurs statiques, et les valeurs proportionnelles et statiques peuvent être mélangées.



Une définition d'un `AbsoluteLayout` dans XAML ressemble à ceci:

```
<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>
```

La même disposition ressemblerait à ceci dans le code:

```
Title = "Absolute Layout Exploration - Code";
var layout = new AbsoluteLayout();

var centerLabel = new Label {
Text = "I'm centered on iPhone 4 but no other device.",
LineBreakMode = LineBreakMode.WordWrap};
```

```
AbsoluteLayout.SetLayoutBounds (centerLabel, new Rectangle (115, 159, 100, 100));
// No need to set layout flags, absolute positioning is the default

var bottomLabel = new Label { Text = "I'm bottom center on every device.", LineBreakMode =
LineBreakMode.WordWrap };
AbsoluteLayout.SetLayoutBounds (bottomLabel, new Rectangle (.5, 1, .5, .1));
AbsoluteLayout.SetLayoutFlags (bottomLabel, AbsoluteLayoutFlags.All);

var rightBox = new BoxView{ Color = Color.Olive };
AbsoluteLayout.SetLayoutBounds (rightBox, new Rectangle (1, .5, 25, 100));
AbsoluteLayout.SetLayoutFlags (rightBox, AbsoluteLayoutFlags.PositionProportional);

var leftBox = new BoxView{ Color = Color.Red };
AbsoluteLayout.SetLayoutBounds (leftBox, new Rectangle (0, .5, 25, 100));
AbsoluteLayout.SetLayoutFlags (leftBox, AbsoluteLayoutFlags.PositionProportional);

var topBox = new BoxView{ Color = Color.Blue };
AbsoluteLayout.SetLayoutBounds (topBox, new Rectangle (.5, 0, 100, 25));
AbsoluteLayout.SetLayoutFlags (topBox, AbsoluteLayoutFlags.PositionProportional);

var twoFlagsBox = new BoxView{ Color = Color.Blue };
AbsoluteLayout.SetLayoutBounds (topBox, new Rectangle (.5, 0, 1, 25));
AbsoluteLayout.SetLayoutFlags (topBox, AbsoluteLayoutFlags.PositionProportional |
AbsoluteLayout.WidthProportional);

layout.Children.Add (bottomLabel);
layout.Children.Add (centerLabel);
layout.Children.Add (rightBox);
layout.Children.Add (leftBox);
layout.Children.Add (topBox);
```

Le contrôle AbsoluteLayout de Xamarin.Forms vous permet de spécifier exactement à l'écran les éléments enfants, ainsi que leur taille et leur forme (limites).

Il existe différentes manières de définir les limites des éléments enfants en fonction de l'énumération AbsoluteLayoutFlags utilisée pendant ce processus. L'énumération **AbsoluteLayoutFlags** contient les valeurs suivantes:

- **Tous** : Toutes les dimensions sont proportionnelles.
- **HeightProportional** : la hauteur est proportionnelle à la disposition.
- **Aucun** : Aucune interprétation n'est effectuée.
- **PositionProportional** : Combine XProportional et YProportional.
- **SizeProportional** : Combine WidthProportional et HeightProportional.
- **WidthProportional** : La largeur est proportionnelle à la disposition.
- **XProportional** : la propriété X est proportionnelle à la disposition.
- **YProportional** : la propriété Y est proportionnelle à la disposition.

Le processus de travail avec la mise en page du conteneur AbsoluteLayout peut sembler un peu contre-intuitif au début, mais avec un peu d'utilisation, il deviendra familier. Une fois que vous avez créé vos éléments enfants, pour les définir dans une position absolue dans le conteneur, vous devez suivre trois étapes. Vous souhaiterez définir les indicateurs assignés aux éléments à l'aide de la méthode **AbsoluteLayout.SetLayoutFlags ()** . Vous voudrez également utiliser la méthode **AbsoluteLayout.SetLayoutBounds ()** pour donner aux éléments leurs limites. Enfin, vous voudrez ajouter les éléments enfants à la collection Children. Étant donné que

Xamarin.Forms est une couche d'abstraction entre Xamarin et les implémentations spécifiques aux périphériques, les valeurs de position peuvent être indépendantes des pixels du périphérique. C'est là que les indicateurs de disposition mentionnés précédemment entrent en jeu. Vous pouvez choisir comment le processus de mise en page des contrôles Xamarin.Forms doit interpréter les valeurs que vous définissez.

la grille

Une présentation contenant des vues organisées en lignes et en colonnes.

Ceci est une définition de `Grid` typique dans 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 même `Grid` définie dans le code ressemble à ceci:

```
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) });
```

Pour ajouter des éléments à la grille: Dans `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 code 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);
```

Pour la `Height` et la `Width` plusieurs unités sont disponibles.

- **Auto** - taille automatiquement pour adapter le contenu de la ligne ou de la colonne. Spécifié comme GridUnitType.Auto en C # ou comme Auto dans XAML.
- **Proportionnel** - dimensionne les colonnes et les lignes en proportion de l'espace restant. Spécifié comme valeur et GridUnitType.Star en C # et en tant que # * dans XAML, # étant la valeur souhaitée. Si vous spécifiez une ligne / colonne avec *, cela remplira l'espace disponible.
- **Absolute** - dimensionne les colonnes et les lignes avec des valeurs de hauteur et de largeur

spécifiques et fixes. Spécifié comme valeur et GridUnitType.Absolute en C # et en # dans XAML, # étant la valeur souhaitée.

Remarque: Les valeurs de largeur des colonnes sont définies par défaut sur Auto dans Xamarin.Forms, ce qui signifie que la largeur est déterminée par la taille des enfants. Notez que cela diffère de l'implémentation de XAML sur les plates-formes Microsoft, où la largeur par défaut est *, ce qui remplira l'espace disponible.

Disposition relative

Une `Layout` qui utilise des `Constraints` pour mettre en forme ses enfants.

`RelativeLayout` permet de positionner et de dimensionner les vues relatives aux propriétés de la présentation ou des vues frères. Contrairement à `AbsoluteLayout` , `RelativeLayout` n'a pas le concept de l'ancre mobile et n'a pas de possibilité de positionner les éléments par rapport aux bords inférieur ou droit de la mise en page. `RelativeLayout` prend en charge le positionnement d'éléments en dehors de ses propres limites.

Un RelativeLayout dans XAML, est comme ceci:

```
<RelativeLayout>
    <BoxView Color="Red" x:Name="redBox"
        RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToParent,
            Property=Height,Factor=.15,Constant=0}"
```

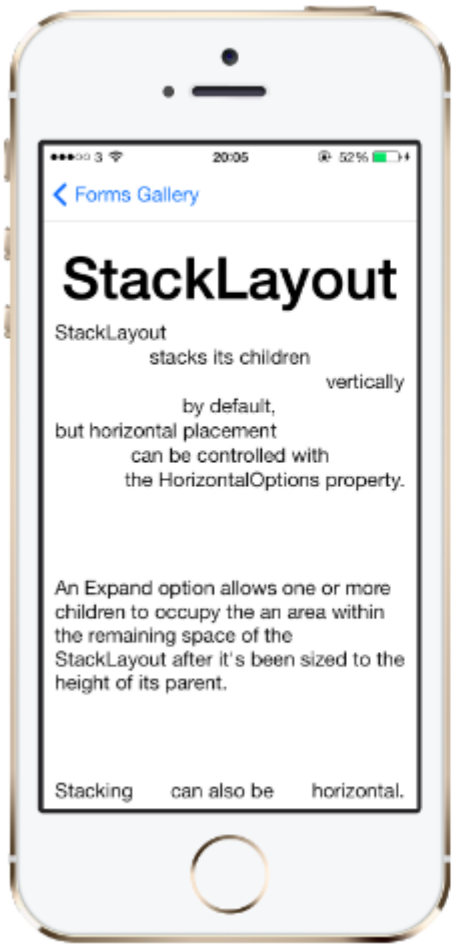
```
        RelativeLayout.WidthConstraint="{ConstraintExpression
            Type=RelativeToParent,Property=Width,Factor=1,Constant=0}"
        RelativeLayout.HeightConstraint="{ConstraintExpression
            Type=RelativeToParent,Property=Height,Factor=.8,Constant=0}" />
    <BoxView Color="Blue"
        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>
```

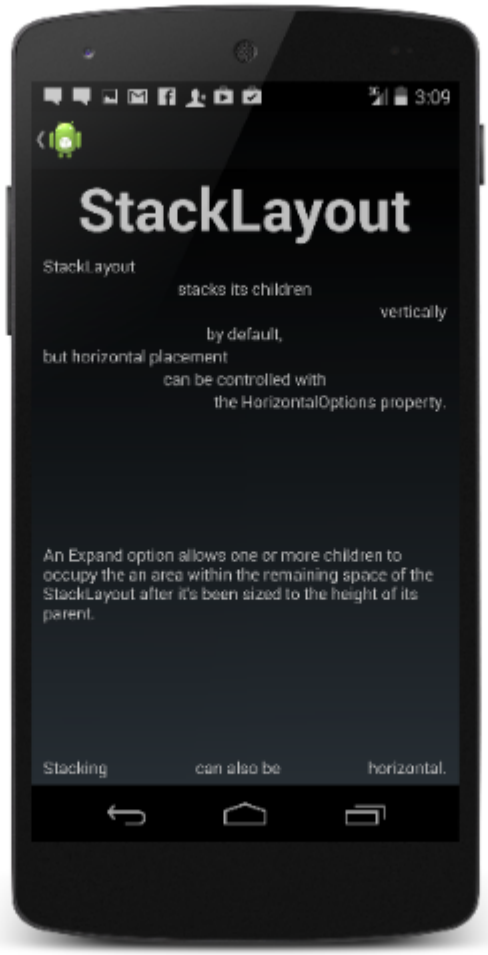
La même disposition peut être réalisée avec ce code:

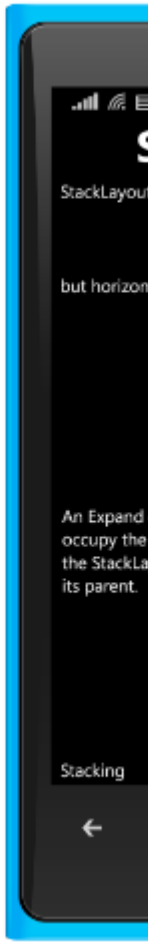
```
layout.Children.Add (redBox, Constraint.RelativeToParent ((parent) => {
        return parent.X;
    }), Constraint.RelativeToParent ((parent) => {
        return parent.Y * .15;
    }), Constraint.RelativeToParent((parent) => {
        return parent.Width;
    }), Constraint.RelativeToParent((parent) => {
        return parent.Height * .8;
    }));
layout.Children.Add (blueBox, Constraint.RelativeToView (redBox, (Parent, sibling) => {
        return sibling.X + 20;
    }), Constraint.RelativeToView (blueBox, (parent, sibling) => {
        return sibling.Y + 20;
    }), Constraint.RelativeToParent((parent) => {
        return parent.Width * .5;
    }), Constraint.RelativeToParent((parent) => {
        return parent.Height * .5;
    }));
```

StackLayout

`StackLayout` organise les vues dans une ligne unidimensionnelle ("pile"), horizontalement ou verticalement. `Views` dans un `StackLayout` peuvent être dimensionnées en fonction de l'espace dans la mise en page à l'aide des options de présentation. Le positionnement est déterminé par les vues de commande qui ont été ajoutées à la mise en page et aux options de disposition des vues.







Utilisation dans XAML

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

Utilisation dans le code

```
StackLayout stackLayout = new StackLayout
{
    Spacing = 0,
    VerticalOptions = LayoutOptions.FillAndExpand,
    Children =
    {
        new Label
        {
            Text = "StackLayout",
            HorizontalOptions = LayoutOptions.Start
        },
        new Label
        {
            Text = "stacks its children",
```

```
            HorizontalOptions = LayoutOptions.Center
        },
        new Label
        {
            Text = "vertically",
            HorizontalOptions = LayoutOptions.End
        },
        new Label
        {
            Text = "by default,",
            HorizontalOptions = LayoutOptions.Center
        },
        new Label
        {
            Text = "but horizontal placement",
            HorizontalOptions = LayoutOptions.Start
        },
        new Label
        {
            Text = "can be controlled with",
            HorizontalOptions = LayoutOptions.Center
        },
        new Label
        {
            Text = "the HorizontalOptions property.",
            HorizontalOptions = LayoutOptions.End
        },
        new Label
        {
            Text = "An Expand option allows one or more children " +
                   "to occupy the an area within the remaining " +
                   "space of the StackLayout after it's been sized " +
                   "to the height of its parent.",
            VerticalOptions = LayoutOptions.CenterAndExpand,
            HorizontalOptions = LayoutOptions.End
        },
        new StackLayout
        {
            Spacing = 0,
            Orientation = StackOrientation.Horizontal,
            Children =
            {
                new Label
                {
                    Text = "Stacking",
                },
                new Label
                {
                    Text = "can also be",
                    HorizontalOptions = LayoutOptions.CenterAndExpand
                },
                new Label
                {
                    Text = "horizontal.",
                },
            }
        }
    }
};
```

Lire Mise en forme de Xamarin en ligne: https://riptutorial.com/fr/xamarin-forms/topic/6273/mise-

en-forme-de-xamarin

Chapitre 26: Mise en page relative au Xamarin

Remarques

L'utilisation de *ForceLayout* dans ce cas

La taille des étiquettes et des boutons change en fonction du texte à l'intérieur d'eux. Par conséquent, lorsque les enfants sont ajoutés à la mise en page, leur taille reste égale à 0 en largeur et en hauteur. Par exemple:

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

L'expression ci-dessus renverra 0 car la largeur est 0 pour le moment. Pour contourner ce *problème* , nous devons écouter l'événement *SizeChanged* et, lorsque la taille change, nous devons forcer la mise en page afin de la redessiner.

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

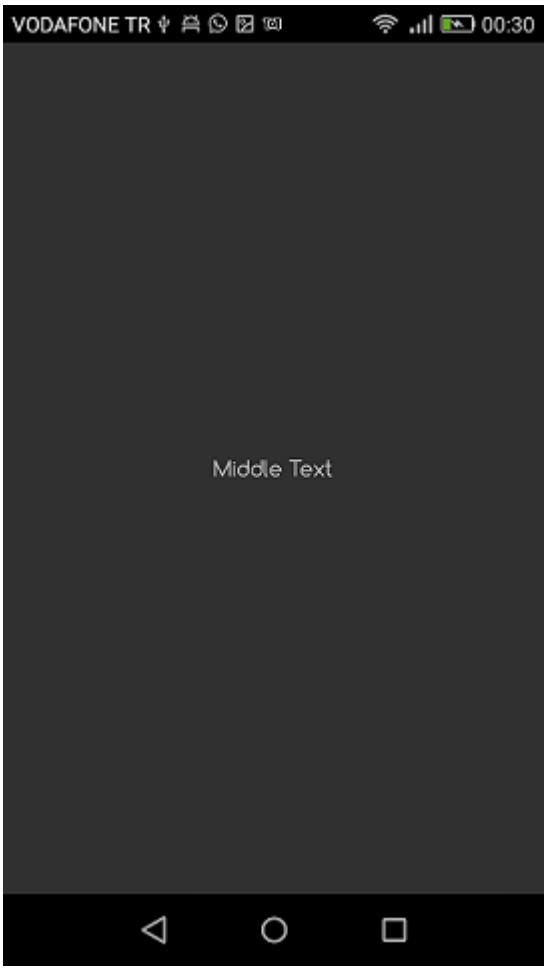
Pour une vue comme *BoxView,* c'est inutile. Parce que nous pouvons définir leurs tailles lors de l'instanciation. L'autre chose est que, dans les deux cas, nous pouvons définir leur largeur et leur hauteur comme une contrainte lorsque nous les ajoutons à la mise en page. Par exemple:

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

Cela ajoutera l'étiquette au point 0, 0. La largeur et la hauteur de l'étiquette seront 30 et 40. Cependant, si le texte est trop long, une partie pourrait ne pas apparaître. Si votre étiquette a ou peut avoir une hauteur élevée, vous pouvez utiliser la propriété *LineBreakMode* de l'étiquette. Qui peut envelopper le texte. Il y a beaucoup d'options dans *LineBreakMode enum* .

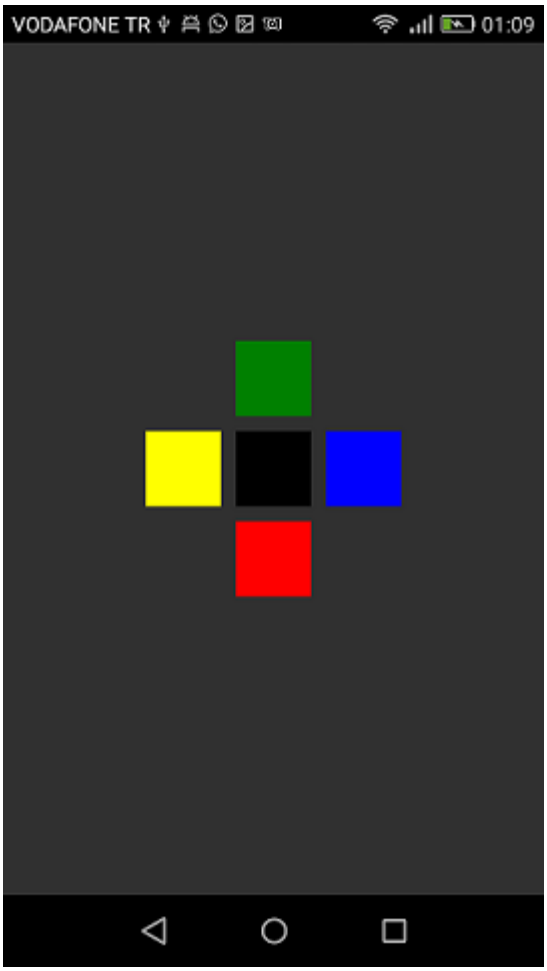
Examples

Page avec une étiquette simple au milieu




```
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;
    }
}
```

Boîte après boîte




```
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;
    }
}
```

Lire Mise en page relative au Xamarin en ligne: https://riptutorial.com/fr/xamarin-forms/topic/6583/mise-en-page-relative-au-xamarin

Chapitre 27: Navigation dans Xamarin.Forms

Examples

Flux de navigation

```
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;
```

```
        }
    }
}
```

Flux de navigationPage avec XAML

Fichier App.xaml.cs (le fichier App.xaml est défini par défaut, donc ignoré)

```
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;
        }
    }
}
```

Fichier 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>
```

Dans certains cas, vous devez ouvrir la nouvelle page non pas dans la navigation actuelle, mais dans la navigation globale. Par exemple, si votre page actuelle contient un menu inférieur, elle sera visible lorsque vous poussez la nouvelle page dans la navigation actuelle. Si vous souhaitez que la page soit ouverte sur tout le contenu visible en masquant le menu inférieur et le contenu de la page en cours, vous devez insérer la nouvelle page en tant que modal dans la navigation

globale. Voir la propriété `App.GlobalNavigation` et l'exemple ci-dessous.

Fichier 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);
        }
    }
}
```

Fichier SecondPage.xaml (le fichier xaml.cs est défini par défaut, donc ignoré)

```
<?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>
```

Navigation hiérarchique avec XAML

Par défaut, le modèle de navigation fonctionne comme une pile de pages, appelant les pages les plus récentes sur les pages précédentes. Vous devrez utiliser l'objet NavigationPage pour cela.

Pousser de nouvelles pages

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

```
    }
}
...
```

Page1.xaml

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

Page1.xaml.cs

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

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

Page2.xaml

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

Page2.xaml.cs

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

    protected async void GoToNextPage(object sender, EventArgs e)
```

```
    {
        await Navigation.PushAsync(new Page3());
    }
}
...
```

Pages de découpage

Normalement, l'utilisateur utilise le bouton Précédent pour renvoyer des pages, mais il est parfois nécessaire de le contrôler par programmation, vous devez donc appeler la méthode **NavigationPage.PopAsync ()** pour revenir à la page précédente ou **NavigationPage.PopToRootAsync ()** pour revenir au début. comme par exemple ...

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();
    }
}
...
```

Navigation modale avec XAML

Les pages modales peuvent être créées de trois manières:

- De l'objet **NavigationPage** pour les pages en plein écran
- Pour les alertes et les notifications

- Pour les feuilles d'action qui sont des menus contextuels

Modaux plein écran

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

Alertes / Confirmations et Notifications

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

Feuilles d'Action

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

Page racine de détail principal

```
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;
    }
}
```

Navigation de détail principale

Le code ci-dessous montre comment effectuer une navigation asynchrone lorsque l'application se trouve dans un contexte 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;
    }
```

Lire Navigation dans Xamarin.Forms en ligne: https://riptutorial.com/fr/xamarin-forms/topic/1571/navigation-dans-xamarin-forms

Chapitre 28: Navigation dans Xamarin.Forms

Remarques

La navigation sur Xamarin.Forms repose sur deux principaux modes de navigation: hiérarchique et modal.

Le modèle hiérarchique permet à l'utilisateur de descendre dans une pile de pages et de revenir en appuyant sur le bouton "retour" / "haut".

Le modèle modal est une page d'interruption nécessitant une action spécifique de l'utilisateur, mais peut normalement être annulée en appuyant sur le bouton Annuler. Quelques exemples sont les notifications, les alertes, les boîtes de dialogue et les pages de registre / édition.

Examples

Utiliser INavigation à partir du modèle de vue

La première étape consiste à créer une interface de navigation que nous utiliserons sur le modèle de vue:

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

Dans la méthode `Initialize` , j'utilise mon mappeur personnalisé où je conserve une collection de types de pages avec des clés associées.

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

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

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

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

```
        return typeSource;
    }

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

        return associatedSource;
    }
}
```

Enum avec des pages:

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

Fichier `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
}
```

Dans mapper, j'ai associé le type d'une page à une valeur enum.

`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.");
        }
    }
}
```

J'obtiens le type de page sur lequel l'utilisateur souhaite naviguer et créer son instance en utilisant la réflexion.

Et puis je pourrais utiliser le service de navigation sur le modèle de vue:

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

Lire Navigation dans Xamarin.Forms en ligne: https://riptutorial.com/fr/xamarin-forms/topic/2507/navigation-dans-xamarin-forms

Chapitre 29: Notifications push

Remarques

Il n'y a pas de méthode uniforme pour gérer les notifications push dans Xamarin Forms, car l'implémentation dépend fortement des fonctionnalités et événements spécifiques à la plateforme. Le code spécifique à la plate-forme sera donc toujours nécessaire.

Cependant, en utilisant `DependencyService` vous pouvez partager autant de code que possible. Il y a aussi un plugin conçu pour cela par rdelrosario, qui peut être trouvé sur son GitHub .

Le code et les captures d'écran sont tirés d'une série de blogs de Gerald Versluis qui explique le processus plus en détail.

Examples

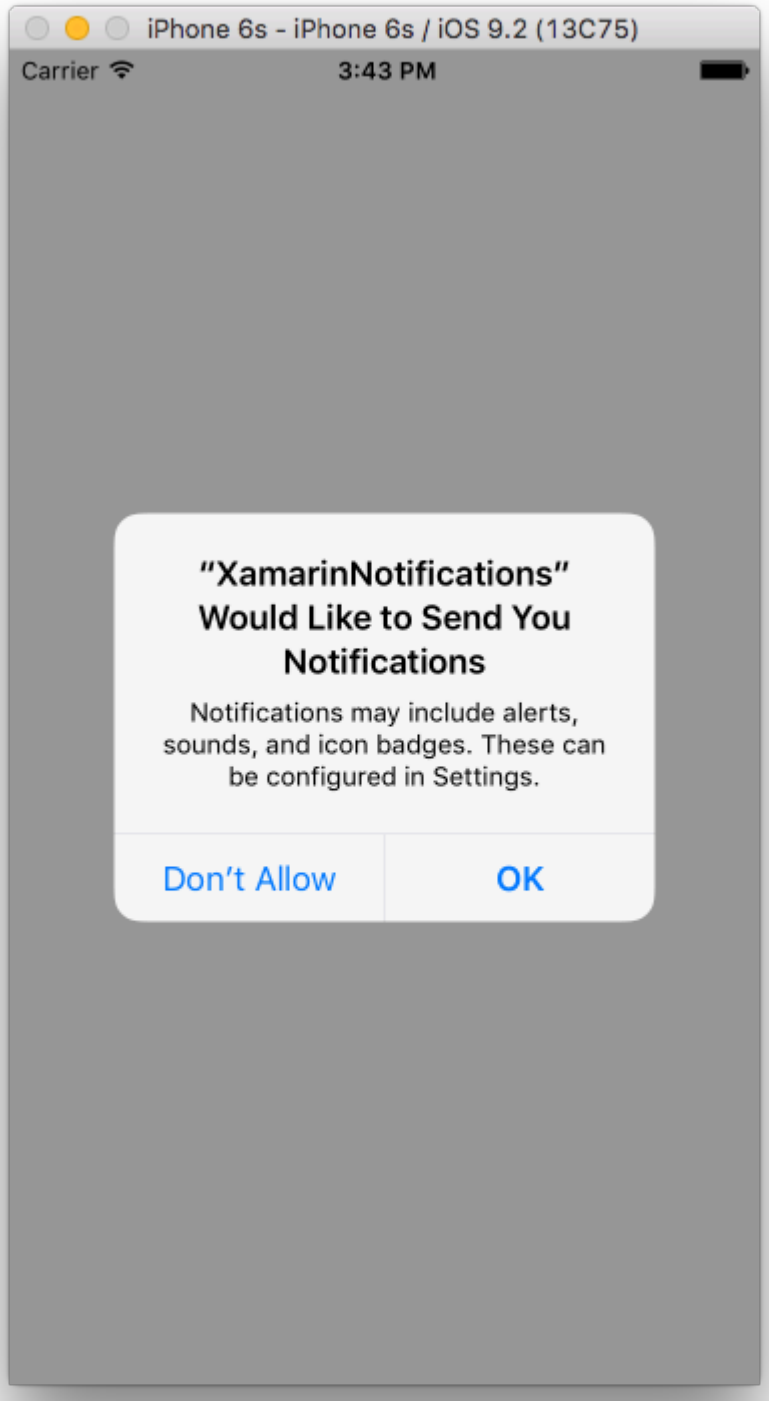
Notifications Push pour iOS avec Azure

Pour démarrer l'enregistrement des notifications push, vous devez exécuter le code ci-dessous.

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

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

Ce code peut être soit directement couru lorsque l'application démarre dans le `FinishedLaunching` dans le `AppDelegate.cs` fichier. Vous pouvez également le faire chaque fois qu'un utilisateur décide d'activer les notifications push.

L'exécution de ce code déclenchera une alerte pour inviter l'utilisateur à accepter que l'application puisse lui envoyer des notifications. Donc, implémentez également un scénario où l'utilisateur le nie!

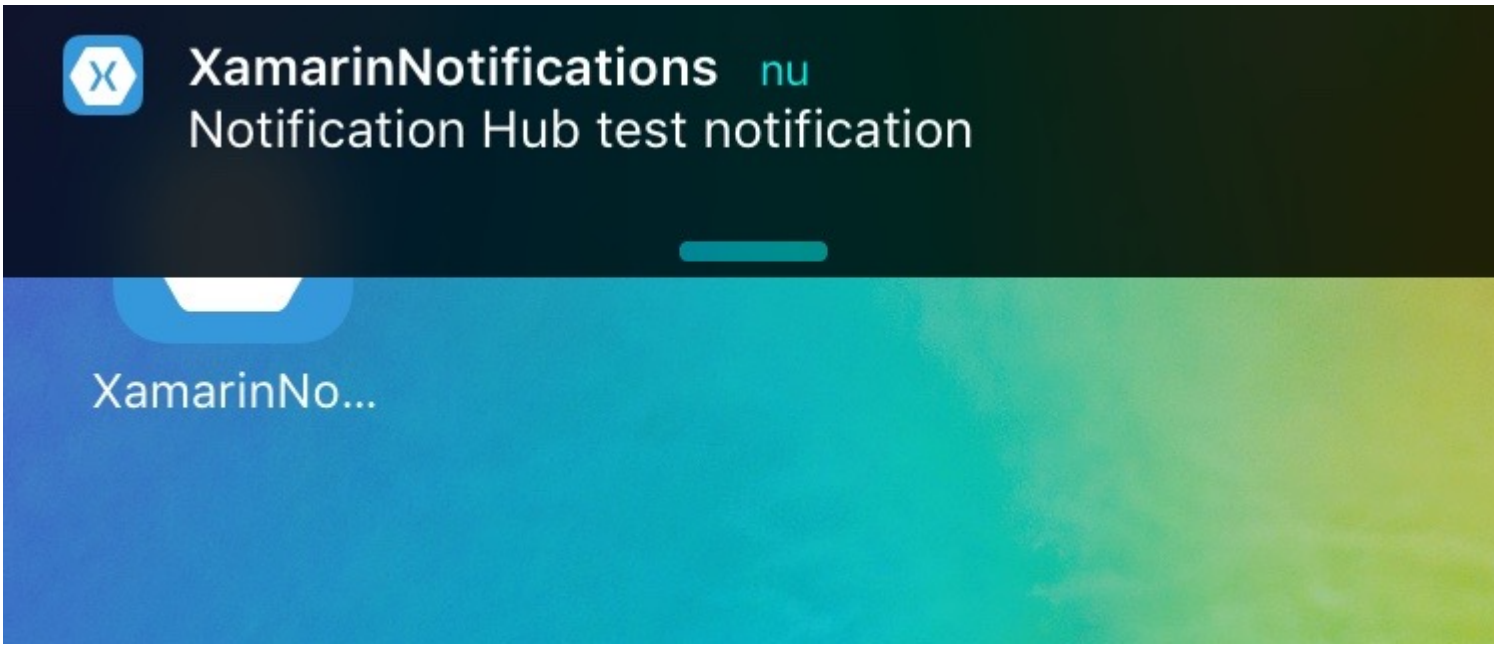


Ce sont les événements qui nécessitent une implémentation pour implémenter les notifications push sur iOS. Vous pouvez les trouver dans le fichier `AppDelegate.cs` .

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

Lorsqu'une notification est reçue, voici à quoi cela ressemble.



Notifications push pour Android avec Azure

L'implémentation sur Android est un peu plus complexe et nécessite la mise en œuvre d'un `Service` spécifique.

Tout d'abord, vérifions si notre appareil est capable de recevoir des notifications push, et si oui, enregistrez-le avec Google. Cela peut être fait avec ce code dans notre fichier `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());
}
```

Les SenderIDs se trouvent dans le code ci-dessous et correspondent au numéro de projet que vous obtenez depuis le tableau de bord du développeur Google afin de pouvoir envoyer des messages Push.

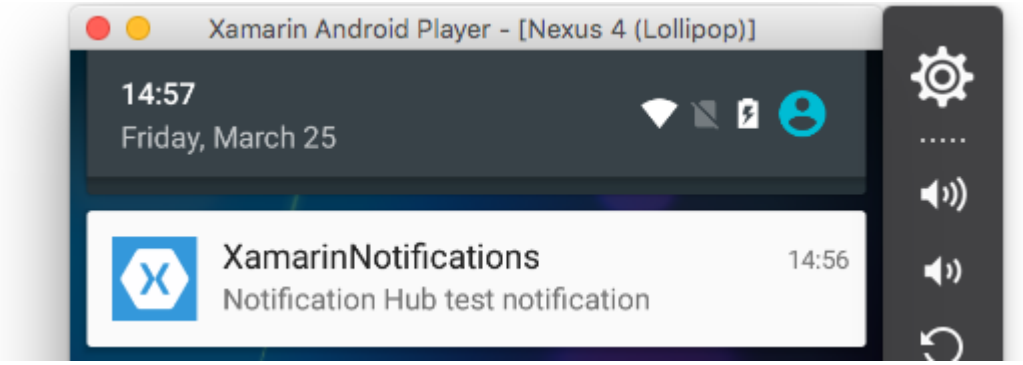
```
using Android.App;
using Android.Content;
using Gcm.Client;
using Java.Lang;
using System;
using WindowsAzure.Messaging;
using XamarinNotifications.Helpers;

// 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);
        }
    }
}
```

Un exemple de notification sur Android ressemble à ceci.

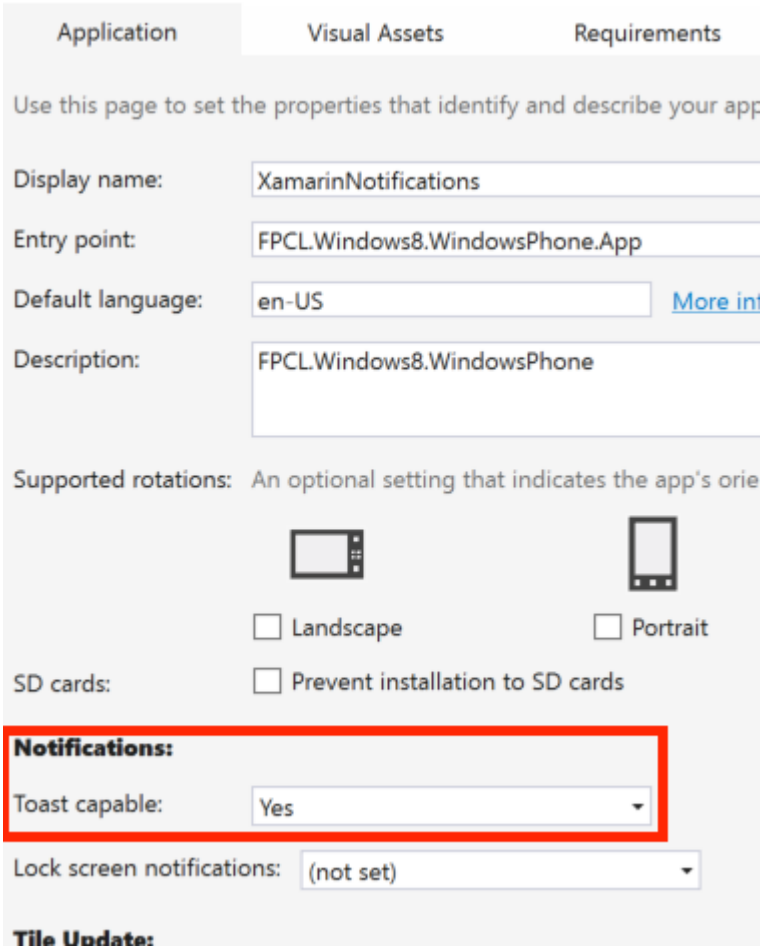


Notifications Push pour Windows Phone avec Azure

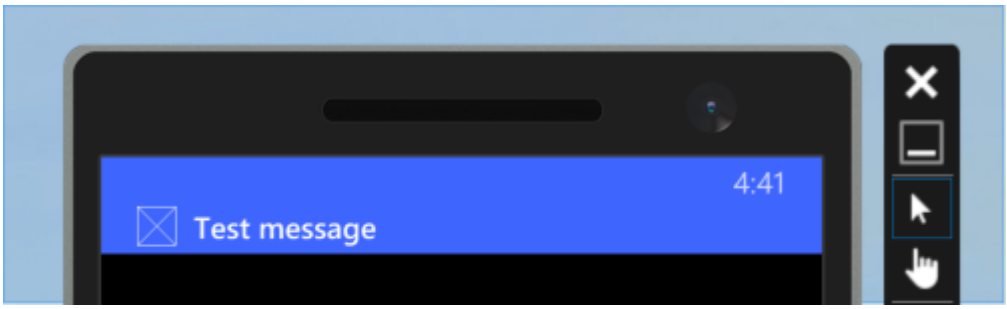
Sur Windows Phone, le code ci-dessous doit être implémenté pour commencer à travailler avec les notifications push. Cela peut être trouvé dans le fichier `App.xaml.cs`

```
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
}
```

N'oubliez pas non plus d'activer les fonctionnalités du fichier `Package.appxmanifest` .



Un exemple de notification push peut ressembler à ceci:



Lire Notifications push en ligne: https://riptutorial.com/fr/xamarin-forms/topic/5042/notifications-push

Chapitre 30: Notifications push

Remarques

AWS Simple Notification Service Lingo:

Point de terminaison - Le point d'extrémité peut être un téléphone, une adresse électronique ou autre, c'est ce que AWS SNS peut envoyer avec une notification.

Topic - Essentiellement un groupe contenant tous vos points de terminaison

S'abonner - Vous vous inscrivez sur votre téléphone / client pour recevoir des notifications

Lingo de notification push générique:

APNS - Apple Push Notification Service. Apple est le seul à pouvoir envoyer des notifications push. C'est pourquoi nous fournissons notre application avec le certificat approprié. Nous fournissons à AWS SNS le certificat que Apple nous fournit pour autoriser SNS à envoyer une notification à APNS en notre nom.

GCM - Google Cloud Messaging est très similaire à APNS. Google est le seul à pouvoir envoyer directement des notifications push. Nous enregistrons donc d'abord notre application dans GCM et remettons notre jeton à AWS SNS. SNS gère tous les problèmes complexes liés à GCM et à l'envoi des données.

Examples

Exemple iOS

1. Vous aurez besoin d'un périphérique de développement
2. Accédez à votre compte de développeur Apple et créez un profil de configuration avec les notifications Push activées.
3. Vous aurez besoin d'un moyen de notifier votre téléphone (AWS, Azure..etc) **Nous utiliserons AWS ici**

```
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"

            });

        }

    }
```

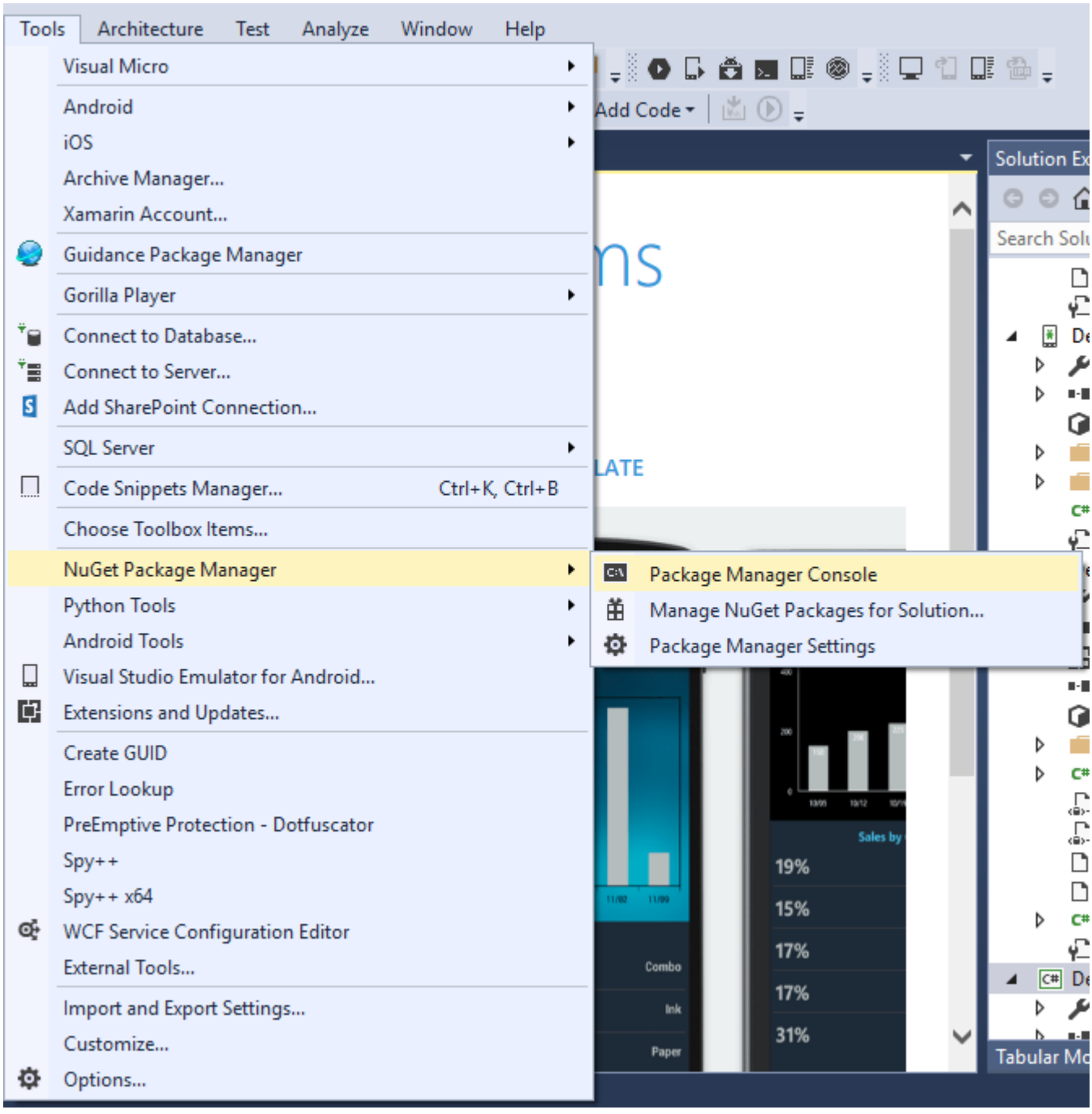
Lire Notifications push en ligne: https://riptutorial.com/fr/xamarin-forms/topic/5998/notifications-push

Chapitre 31: OAuth2

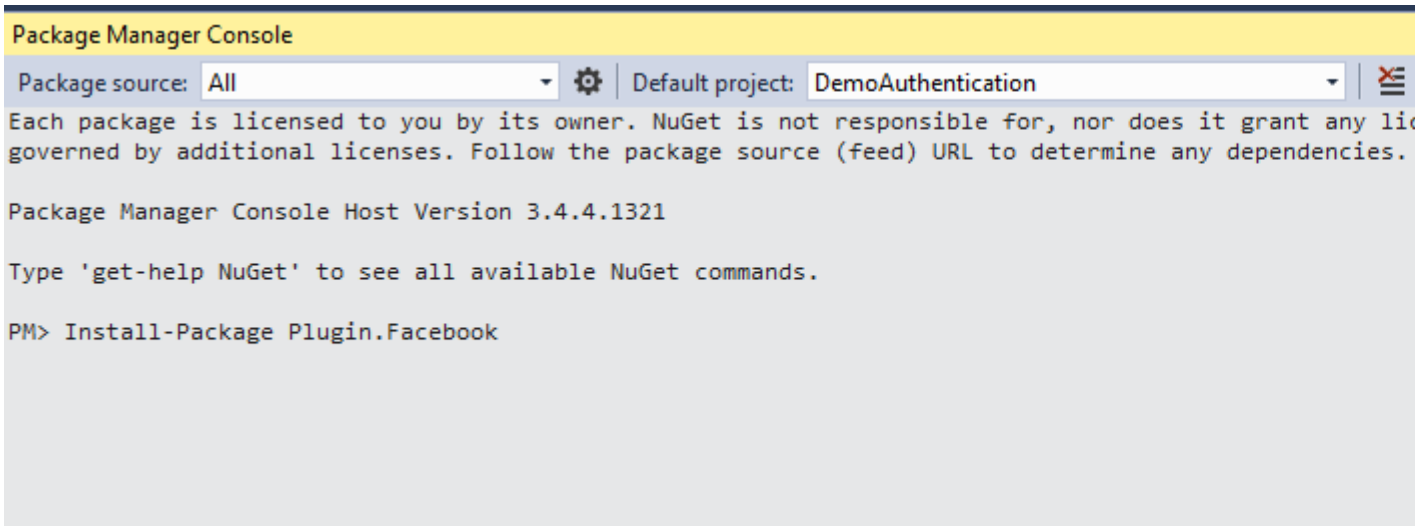
Examples

Authentification à l'aide du plugin

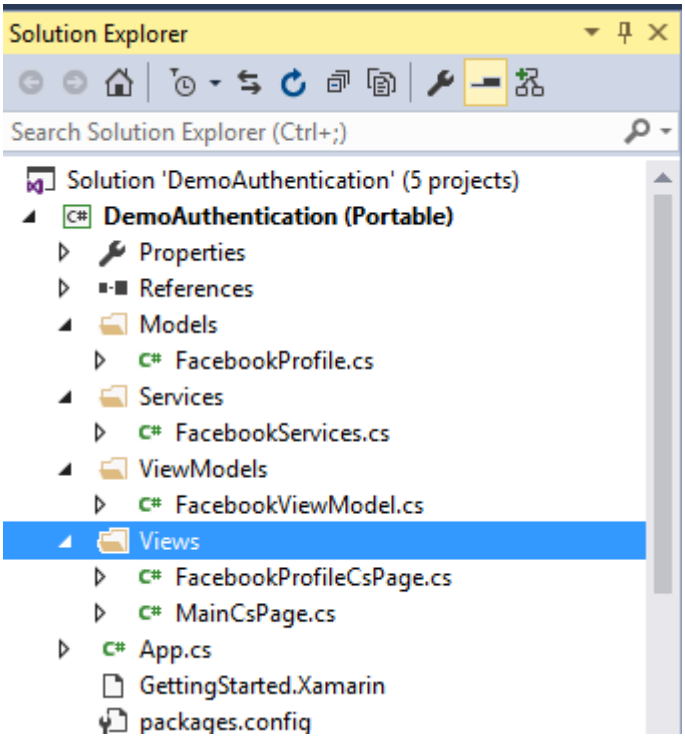
1. Tout d'abord, allez dans **Outils** > **Gestionnaire de packages NuGet** > **Console du gestionnaire de packages** .



2. Entrez cette commande " **Install-Package Plugin.Facebook** " dans la console du **gestionnaire** de paquets.



3. Maintenant tout le fichier est automatiquement créé.



Vidéo : Connectez - vous avec Facebook dans Xamarin Forms

Autre authentification à l'aide du plugin. Veuillez placer la commande dans la console du gestionnaire de packages, comme indiqué à l'étape 2.

1. **Youtube** : Install-Package Plugin.Youtube
2. **Twitter** : Plugin du package d'installation
3. **Foursquare** : Install-Package Plugin.Foursquare
4. **Google** : Plugin Package-Install.Google
5. **Instagram** : Install-Package Plugin.Instagram
6. **Eventbrite** : Plugin du package d'installation.Eventbrite

Lire OAuth2 en ligne: https://riptutorial.com/fr/xamarin-forms/topic/8828/oauth2

Chapitre 32: Page Xamarin.Forms

Examples

TabbedPage

Une TabbedPage est similaire à une NavigationPage en ce sens qu'elle permet et gère une navigation simple entre plusieurs objets Page enfant. La différence est que, en règle générale, chaque plate-forme affiche une sorte de barre en haut ou en bas de l'écran qui affiche la plupart, voire la totalité, des objets enfant disponibles. Dans les applications Xamarin.Forms, une TabbedPage est généralement utile lorsque vous disposez d'un petit nombre prédéfini de pages entre lesquelles les utilisateurs peuvent naviguer, comme un menu ou un assistant simple pouvant être placé en haut ou en bas de l'écran.

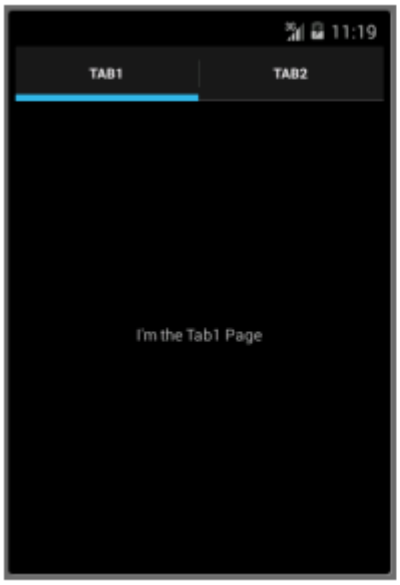
XAML

```
<?xml version="1.0" encoding="utf-8" ?>
<TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            x:Class="XamlBasics.SampleXaml">
  <TabbedPage.Children>
    <ContentPage Title="Tab1">
      <Label Text="I'm the Tab1 Page"
             HorizontalOptions="Center"
             VerticalOptions="Center"/>
    </ContentPage>
    <ContentPage Title="Tab2">
    <Label Text="I'm the Tab2 Page"
           HorizontalOptions="Center"
           VerticalOptions="Center"/>
    </ContentPage>
  </TabbedPage.Children>
</TabbedPage>
```

Code

```
var page1 = new ContentPage {
Title = "Tab1",
Content = new Label {
Text = "I'm the Tab1 Page",
HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center
}
};
var page2 = new ContentPage {
Title = "Tab2",
Content = new Label {
Text = "I'm the Tab2 Page",
HorizontalOptions = LayoutOptions.Center,
66
VerticalOptions = LayoutOptions.Center
```

```
}
};
var tabbedPage = new TabbedPage {
Children = { page1, page2 }
};
```



Contenu de la page

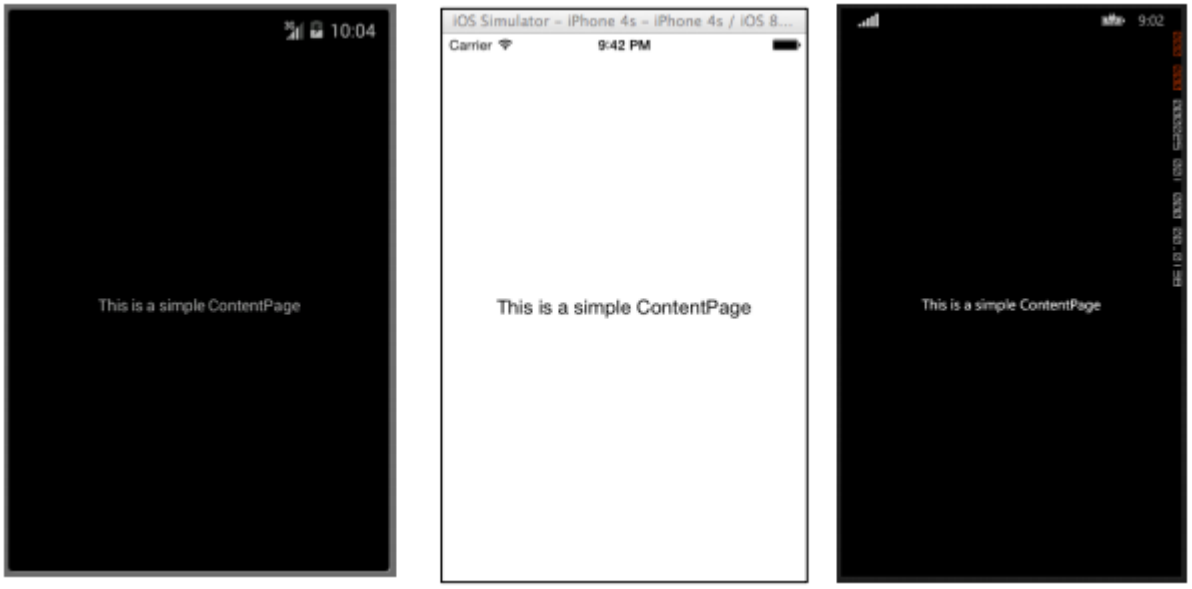
ContentPage: Affiche une seule vue.

XAML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="XamlBasics.SampleXaml">
<Label Text="This is a simple ContentPage"
HorizontalOptions="Center"
VerticalOptions="Center" />
</ContentPage>
```

Code

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



MasterDetailPage

MasterDetailPage: Gère deux pages distinctes (volets) d'informations.

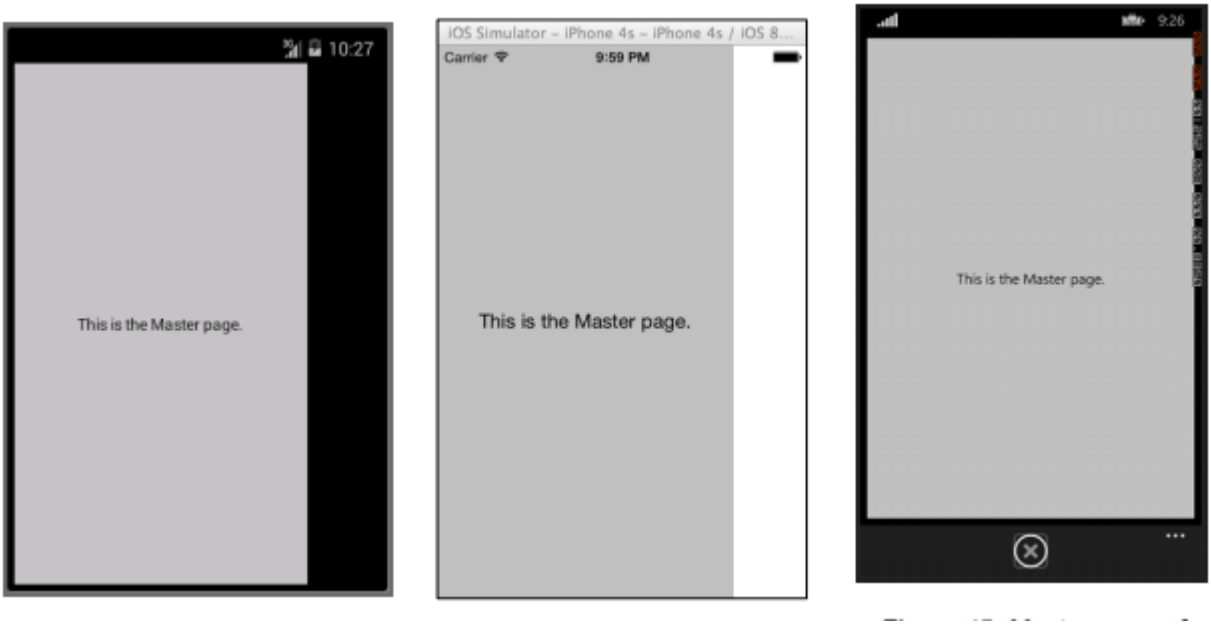
XAML

```
<?xml version="1.0" encoding="utf-8" ?>
<MasterDetailPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="XamlBasics.SampleXaml">
<MasterDetailPage.Master>
<ContentPage Title = "Master" BackgroundColor = "Silver">
<Label Text="This is the Master page."
TextColor = "Black"
HorizontalOptions="Center"
VerticalOptions="Center" />
</ContentPage>
</MasterDetailPage.Master>
<MasterDetailPage.Detail>
<ContentPage>
<Label Text="This is the Detail page."
HorizontalOptions="Center"
VerticalOptions="Center" />
</ContentPage>
</MasterDetailPage.Detail>
</MasterDetailPage>
```

Code

```
var masterDetailPage = new MasterDetailPage {
Master = new ContentPage {
Content = new Label {
Title = "Master",
BackgroundColor = Color.Silver,

TextColor = Color.Black,
Text = "This is the Master page.",
HorizontalOptions = LayoutOptions.Center,
```

```
VerticalOptions = LayoutOptions.Center
}
},
Detail = new ContentPage {
Content = new Label {
Title = "Detail",
Text = "This is the Detail page.",
HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center
}
}
};
```



Lire Page Xamarin.Forms en ligne: https://riptutorial.com/fr/xamarin-forms/topic/7018/page-xamarin-forms

Chapitre 33: Polices personnalisées dans les styles

Remarques

Ressources à regarder:

- Styles Xamarin
- Utilisation de polices personnalisées sur iOS et Android avec Xamarin.Forms
- Renderers personnalisés
- Dictionnaires de ressources
- Propriétés attachées

Examples

Accéder aux polices personnalisées dans Syles

Xamarin.Forms fournit un excellent mécanisme pour styliser votre application multi-plateformes avec des styles globaux.

Dans le monde mobile, votre application doit être jolie et se démarquer des autres applications. L'un de ces caractères est les polices personnalisées utilisées dans l'application.

Grâce à la prise en charge de XAML Styling dans Xamarin.Forms, vous venez de créer un style de base pour toutes les étiquettes avec vos polices personnalisées.

Pour inclure des polices personnalisées dans votre projet iOS et Android, suivez le guide dans Utilisation de polices personnalisées sur iOS et Android avec le message Xamarin.Forms écrit par Gerald.

Déclarez le style dans la section des ressources du fichier App.xaml. Cela rend tous les styles globalement visibles.

Dans Gerald post ci-dessus, nous devons utiliser la propriété StyleId, mais ce n'est pas une propriété pouvant être liée, donc pour l'utiliser dans Style Setter, vous devez créer une propriété attachable:

```
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);
        }
    }
}
```

Ajoutez ensuite du style dans la ressource 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>
```

Selon l'article ci-dessus, nous devons créer un moteur de rendu personnalisé pour Label qui hérite de la plateforme LabelRenderer On 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;
        }
    }
}
```

Pour la plate-forme iOS, aucun moteur de rendu personnalisé n'est requis.

Vous pouvez maintenant obtenir du style dans le balisage de votre page:

Pour étiquette spécifique

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

Ou appliquer un style à toutes les étiquettes de la page en créant Style basé sur 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>
```

Lire Polices personnalisées dans les styles en ligne: https://riptutorial.com/fr/xamarin-forms/topic/4854/polices-personnalisees-dans-les-styles

Chapitre 34: Pourquoi utiliser les formulaires Xamarin et quand utiliser les formulaires Xamarin

Remarques

Vous pouvez vous référer à la documentation officielle de Xamarin Forms pour en savoir plus:

https://www.xamarin.com/forms

Examples

Pourquoi utiliser les formulaires Xamarin et quand utiliser les formulaires Xamarin

Xamarin devient de plus en plus populaire - il est difficile de décider quand utiliser Xamarin.Forms et quand Xamarin.Platform (donc Xamarin.iOS et Xamarin.Android).

Tout d'abord, vous devez savoir pour quel type d'applications vous pouvez utiliser Xamarin.Forms:

1. Prototypes - pour visualiser comment votre application regardera les différents appareils.
2. Les applications ne nécessitant pas de fonctionnalité spécifique à une plate-forme (comme les API), mais veuillez noter que Xamarin travaille activement pour fournir autant de compatibilité multiplate-forme que possible.
3. Les applications où le partage de code est crucial - plus important que l'interface utilisateur.
4. Applications où les données affichées sont plus importantes que les fonctionnalités avancées

Il y a aussi beaucoup d'autres facteurs:

1. Qui sera responsable du développement des applications - si votre équipe est composée de développeurs mobiles expérimentés, ils seront en mesure de gérer facilement les Xamarin.Forms. Mais si vous avez un développeur par plate-forme (développement natif), les formulaires peuvent être plus difficiles.
2. Notez également qu'avec Xamarin.Forms, vous pouvez toujours rencontrer certains problèmes - la plate-forme Xamarin.Forms est toujours en cours d'amélioration.
3. Le développement rapide est parfois très important - pour réduire les coûts et les délais, vous pouvez décider d'utiliser des formulaires.

4. Lorsque vous développez des applications d'entreprise sans aucune fonctionnalité avancée, il est préférable d'utiliser Xamarin.Forms - il vous permet de partager le code de mode et non l'événement dans la zone mobile, mais en général. Certaines parties du code peuvent être partagées sur plusieurs plates-formes.

Vous ne devez pas utiliser Xamarin.Forms lorsque:

1. Vous devez créer des fonctionnalités personnalisées et accéder à des API spécifiques à la plate-forme

2. Vous devez créer une interface utilisateur personnalisée pour l'application mobile

3. Lorsque certaines fonctionnalités ne sont pas prêtes pour Xamarin.Forms (comme certains comportements spécifiques sur le périphérique mobile)

4. Votre équipe est composée de développeurs mobiles spécifiques à la plate-forme (développement mobile en Java et / ou Swift / Objective C).

Lire Pourquoi utiliser les formulaires Xamarin et quand utiliser les formulaires Xamarin en ligne: https://riptutorial.com/fr/xamarin-forms/topic/6869/pourquoi-utiliser-les-formulaires-xamarin-et-quand-utiliser-les-formulaires-xamarin

Chapitre 35: Renderers personnalisés

Examples

Rendu personnalisé pour ListView

Les moteurs de rendu personnalisés permettent aux développeurs de personnaliser l'apparence et le comportement des contrôles Xamarin.Forms sur chaque plate-forme. Les développeurs peuvent utiliser les fonctionnalités des contrôles natifs.

Par exemple, nous devons désactiver le défilement dans `ListView` . Sur iOS, `ListView` est défilable même si tous les éléments sont placés à l'écran et que l'utilisateur ne peut pas faire défiler la liste. `Xamarin.Forms.ListView` ne gère pas ce paramètre. Dans ce cas, un moteur de rendu vient en aide.

Tout d'abord, nous devons créer un contrôle personnalisé dans le projet PCL, qui déclarera une propriété obligatoire pouvant être liée:

```
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); }
    }
}
```

La prochaine étape consistera à créer un moteur de rendu pour chaque plate-forme.

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;
        }
```

```
    }
}
```

Et Android (la liste d'Android n'a pas de défilement si tous les éléments sont placés à l'écran, donc nous ne désactiverons pas le défilement, mais nous pouvons toujours utiliser les propriétés natives):

```
[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` propriété `SuperListView` du moteur de rendu est mon contrôle `SuperListView` du projet PCL.

`Control` propriété de `Control` du rendu est un contrôle natif. `Android.Widget.ListView` pour Android et `UIKit.UITableView` pour iOS.

Et comment nous allons l'utiliser dans `XAML` :

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


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

`.cs` fichier de page:

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

```
    public ObservableCollection<string> Items
    {
        get { return _items; }
        set
        {
            _items = value;
            OnPropertyChanged();
        }
    }

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

        InitializeComponent();

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

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

Renderer personnalisé pour BoxView

L'aide du moteur de rendu personnalisé permet d'ajouter de nouvelles propriétés et de les rendre différemment dans une plate-forme native qui ne peut pas l'être autrement via un code partagé. Dans cet exemple, nous allons ajouter un rayon et une ombre à une vue de boîte.

Tout d'abord, nous devons créer un contrôle personnalisé dans le projet PCL, qui déclarera une propriété obligatoire pouvant être liée:

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

La prochaine étape consistera à créer un moteur de rendu pour chaque plate-forme.

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);
        }
    }

 }
}
```

Encore une fois, vous pouvez personnaliser comme vous le souhaitez dans la méthode de dessin.

Et même pour 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);
        }
}
```

}

Le XAML:

Nous faisons d'abord référence à notre contrôle avec l'espace de noms que nous avons défini précédemment.

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

Nous utilisons ensuite le contrôle comme suit et utilisons les propriétés définies au début:

```
<Controls:ExtendedBoxView
    x:Name="search_boxview"
    Color="#444"
```

```
    BorderRadius="5"
    HorizontalOptions="CenterAndExpand"
    />
```

Accéder au rendu d'un projet natif

```
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
```

Étiquette arrondie avec un moteur de rendu personnalisé pour Frame (parties PCL et iOS)

Première étape: partie 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); }
        }
    }
}
```

Deuxième étape: partie 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;
            }
        }
    }
}
```

Troisième étape: code XAML pour appeler un fichier ExtendedFrame

Si vous voulez l'utiliser dans une partie XAML, n'oubliez pas d'écrire ceci:

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

après

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

Maintenant, vous pouvez utiliser le ExtendedFrame comme ceci:

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

BoxView arrondi avec couleur de fond sélectionnable

Première étape: partie 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);
        }
    }
}
```

Deuxième étape: partie Droid

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

Troisième étape: partie 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();
            }
        }
    }
}
```

Lire Renderers personnalisés en ligne: https://riptutorial.com/fr/xamarin-forms/topic/2949/renderers-personnalises

Chapitre 36: Services de dépendance

Remarques

Accéder à l'API spécifique à la plate-forme à partir du projet PCL ou Shared.

Examples

Accéder à la caméra et à la galerie.

(Code d'accès) [https://github.com/vDoers/vDoersCameraAccess]

Lire Services de dépendance en ligne: https://riptutorial.com/fr/xamarin-forms/topic/6127/services-de-dependance

Chapitre 37: Test d'unité

Examples

Test des modèles de vue

Avant de commencer...

En termes de couches d'application, votre ViewModel est une classe contenant toute la logique métier et les règles permettant à l'application de faire ce qu'elle doit selon les besoins. Il est également important de le rendre le plus indépendant possible, en réduisant les références à l'interface utilisateur, à la couche de données, aux fonctionnalités natives et aux appels d'API, etc. Tous ces éléments permettent de tester votre machine virtuelle.
En bref, votre ViewModel:

- Ne devrait pas dépendre des classes de l'interface utilisateur (vues, pages, styles, événements);
- Ne devrait pas utiliser les données statiques d'une autre classe (autant que vous le pouvez);
- Doit implémenter la logique métier et préparer les données sur l'interface utilisateur;
- Devrait utiliser d'autres composants (base de données, HTTP, spécifique à l'interface utilisateur) via des interfaces en cours de résolution à l'aide de l'injection de dépendances.

  Votre ViewModel peut également avoir les propriétés d'un autre type de machine virtuelle. Par exemple, `ContactsPageViewModel` aura la propriété du type de collection comme `ObservableCollection<ContactListItemViewModel>`

Besoins de l'entreprise

Disons que nous avons les fonctionnalités suivantes à implémenter:

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

Après avoir clarifié la user story, nous avons défini les scénarios suivants:

```
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
```

Nous resterons avec seulement ces deux scénarios. Bien sûr, il devrait y avoir beaucoup plus de cas et vous devriez tous les définir avant le codage réel, mais il est assez suffisant pour nous familiariser maintenant avec les tests unitaires des modèles de vue.

Suivons l'approche TDD classique et commençons par écrire une classe vide en cours de test. Ensuite, nous écrirons des tests et les rendrons verts en implémentant la fonctionnalité métier.

Cours communs

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

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

Prestations de service

Vous souvenez-vous que notre modèle de vue ne doit pas utiliser directement les classes UI et HTTP? Vous devez les définir comme des abstractions à la place et ne pas dépendre des détails de l'implémentation .

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

Construire le talon ViewModel

Ok, nous allons avoir la classe de page pour l'écran de connexion, mais commençons par ViewModel:

```
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);
    }
}
```

Nous avons défini deux propriétés de `string` et une commande à lier sur l'interface utilisateur. Nous ne décrirons pas comment créer une classe de page, un balisage XAML et lier ViewModel à cette rubrique car ils n'ont rien de spécifique.

Comment créer une instance LoginPageViewModel?

Je pense que vous créiez probablement les VM uniquement avec constructeur. Maintenant, comme vous pouvez le voir, notre machine virtuelle dépend de 2 services injectés en tant que paramètres de constructeur, alors vous ne pouvez pas simplement faire `var viewModel = new LoginPageViewModel()` . Si vous n'êtes pas familier avec l' injection de dépendance, c'est le meilleur moment pour en apprendre davantage. Des tests unitaires appropriés sont impossibles sans connaître et suivre ce principe.

Des tests

Maintenant, écrivons quelques tests en fonction des cas d'utilisation listés ci-dessus. Tout d'abord, vous devez créer un nouvel assembly (juste une bibliothèque de classes ou sélectionner un projet de test spécial si vous souhaitez utiliser les outils de test unitaire Microsoft). Nommez-le quelque chose comme `ProjectName.Tests` et ajoutez une référence à votre projet PCL original.

Dans cet exemple, je vais utiliser NUnit et Moq, mais vous pouvez continuer avec tous les tests de votre choix. Il n'y aura rien de spécial avec eux.

Ok, c'est la classe de test:

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

Tests d'écriture

Voici les méthodes de test pour les deux premiers scénarios. Essayez de conserver 1 méthode de test par 1 résultat attendu et de ne pas tout vérifier en un seul test. Cela vous aidera à recevoir des rapports plus clairs sur ce qui a échoué dans le code.

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

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

        Assert.IsTrue(model.IsLoading);
    }

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

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

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

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

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

```
string message)
    {
        CreateViewModelAndLogin(userName, password);

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

    private LoginPageViewModel CreateViewModelAndLogin(string userName, string password)
    {
        var model = new LoginPageViewModel(
            authenticationServiceMock.Object,
            alertServiceMock.Object);

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

        model.LoginCommand.Execute(null);

        return model;
    }
}
```

Et c'est reparti:

Maintenant, l'objectif est d'écrire une implémentation correcte pour la méthode de `Login` de ViewModel et c'est tout.

Implémentation de la logique métier

```
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);
```

```
    }
}
```

Et après avoir exécuté les tests à nouveau:

Maintenant, vous pouvez continuer à couvrir votre code avec de nouveaux tests le rendant plus stable et sans risque de régression.

Lire Test d'unité en ligne: https://riptutorial.com/fr/xamarin-forms/topic/3529/test-d-unite

Chapitre 38: Test d'unité BDD dans Xamarin.Forms

Remarques

- Le conteneur / résolveur DI que nous utilisons en interne dans cette bibliothèque est Autofac.
- Le framework de test est NUnit 3x.
- Vous devriez pouvoir utiliser cette bibliothèque avec n'importe quel framework Xamarin.Forms
- Source et exemple de projet disponible ici

Examples

Spécflow simple pour tester les commandes et la navigation avec NUnit Test Runner

Pourquoi avons nous besoin de ça?

La manière actuelle de faire des tests unitaires dans Xamarin.Forms est via un runner de plate-forme, votre test devra donc s'exécuter dans un environnement d'interface utilisateur ios, android, windows ou mac: Exécution de tests dans l'EDI

Xamarin fournit également des tests d'interface utilisateur impressionnants avec l'offre Xamarin.TestCloud , mais lorsque vous souhaitez implémenter des pratiques de développement BDD et que vous avez la possibilité de tester ViewModels et les commandes, tout en fonctionnant à peu de frais sur un serveur de test local ou sur un serveur de construction, construit de manière.

J'ai développé une bibliothèque qui permet d'utiliser Specflow avec Xamarin.Forms pour implémenter facilement vos fonctionnalités depuis vos définitions de scénarios jusqu'à ViewModel, indépendamment de tout framework MVVM utilisé pour l'application (tels que XLabs , MVVMCross , Prism).

Si vous découvrez le BDD, cochez la case Specflow out.

Usage:

- Si vous ne l'avez pas encore, installez l'extension visuelle studio specflow à partir d'ici (ou de votre IDE studio visuel): https://visualstudiogallery.msdn.microsoft.com/c74211e7-cb6e-4dfa-855d-df0ad4a37dd6

- Ajoutez une bibliothèque de classes à votre projet Xamarin.Forms. C'est votre projet de test.

- Ajoutez le package SpecFlow.Xamarin.Forms de nuget à vos projets de test.
- Ajoutez une classe à votre projet de test qui hérite de 'TestApp', et enregistrez vos paires views / viewmodels, ainsi que tout enregistrement DI, comme indiqué ci-dessous:

```
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();
    }
}
```

- Ajoutez une classe SetupHook à votre projet de test afin d'ajouter vos hooks Specflow. Vous devrez lancer l'application de test comme indiqué ci-dessous, en indiquant la classe que vous avez créée ci-dessus et le modèle de vue initial de votre application:

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

- Vous devrez ajouter un bloc catch à votre xamarin.forms views codebehind afin d'ignorer le framework xamarin.forms vous obligeant à exécuter l'application ui (ce que nous ne voulons pas faire):

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

```
        }
```

- Ajouter une fonctionnalité de spécification à votre projet (en utilisant les modèles vs specflow fournis avec l'extension vs specflow)
- Créez / générez une classe d'étape qui hérite de TestStepBase, en transmettant le paramètre scenarioContext à la base.
- Utilisez les services de navigation et les assistants pour naviguer, exécuter des commandes et tester vos modèles de vue:

```
  [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);
        }
    }
```

Utilisation avancée pour MVVM

Pour ajouter au premier exemple, afin de tester les instructions de navigation qui se produisent dans l'application, nous devons fournir le ViewModel avec un crochet à la navigation. Pour y parvenir:

- Ajoutez le package SpecFlow.Xamarin.Forms.IViewModel de nuget à votre projet PCL Xamarin.Forms
- Implémentez l'interface IViewModel dans votre ViewModel. Cela exposera simplement la propriété Xamarin.Forms INavigation:
- `public class MainViewModel : INotifyPropertyChanged, IViewModel.IViewModel { public INavigation Navigation { get; set; }`

- Le framework de test le récupérera et gérera la navigation interne
- Vous pouvez utiliser n'importe quel framework MVVM pour votre application (comme XLabs , MVVMCross , Prism , etc.) . Tant que l'interface IViewModel est implémentée dans votre ViewModel, le framework le récupérera.

Lire Test d'unité BDD dans Xamarin.Forms en ligne: https://riptutorial.com/fr/xamarin-forms/topic/6172/test-d-unite-bdd-dans-xamarin-forms

Chapitre 39: Travailler avec des bases de données locales

Examples

Utilisation de SQLite.NET dans un projet partagé

SQLite.NET est une bibliothèque open source qui permet d'ajouter le support des bases de données locales à l'aide de `SQLite` version 3 dans un projet `Xamarin.Forms` .

Les étapes ci-dessous montrent comment inclure ce composant dans un `Xamarin.Forms` partagé `Xamarin.Forms` :

1. Téléchargez la dernière version de la classe SQLite.cs et ajoutez-la au projet partagé.
2. Chaque table qui sera incluse dans la base de données doit être modélisée en tant que classe dans le projet partagé. Une table est définie en ajoutant au moins deux attributs dans la classe: `Table` (pour la classe) et `PrimaryKey` (pour une propriété).

Pour cet exemple, une nouvelle classe nommée `Song` est ajoutée au projet partagé, défini comme suit:

```
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. Ensuite, ajoutez une nouvelle classe appelée `Database` , qui hérite de la classe `SQLiteConnection` (incluse dans SQLite.cs). Dans cette nouvelle classe, le code d'accès à la base de données, la création des tables et les opérations CRUD pour chaque table sont définis. Exemple de code est indiqué ci-dessous:

```
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. Comme vous avez pu le voir à l'étape précédente, le constructeur de notre classe `Database` contient un paramètre `path` , qui représente l'emplacement du fichier qui stocke le fichier de base de données SQLite. Un objet de `Database` statique peut être déclaré dans `App.cs` Le `path` est spécifique à la plate-forme:

```
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. Maintenant, appelez simplement l'objet `DB` via la classe `App` chaque fois que vous devez effectuer une opération CRUD sur la table `Songs` . Par exemple, pour insérer un nouveau `Song` après que l'utilisateur a cliqué sur un bouton, vous pouvez utiliser le code suivant:

```
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);
}
```

Travailler avec des bases de données locales en utilisant xamarin.forms dans visual studio 2015

Exemple SQlite Etape par étape Explication

1. Les étapes ci-dessous montrent comment inclure ce composant dans un projet partagé Xamarin.Forms: ajouter des packages dans (pcl, Andriod, Windows, Ios) Ajouter des références Cliquez sur **Gérer les packages Nuget** -> cliquez sur Parcourir pour installer **SQLite.Net.Core- PCL** , **SQLite Net Extensions** après l'installation est terminée

2. Pour ajouter Class **Employee.cs au-** dessous du code

```
using SQLite.Net.Attributes;

    namespace DatabaseEmployeeCreation.SqlLite
    {
        public   class Employee
        {
            [PrimaryKey,AutoIncrement]
            public int Eid { get; set; }
            public string Ename { get; set; }
            public string Address { get; set; }
            public string phonenumber { get; set; }
            public string email { get; set; }
        }
    }
```

3. Ajouter une interface ISQLite

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

4. Créez une classe unique pour les logiques de base de données et les méthodes ci-dessous sont utilisées.

utiliser SQLite.Net; en utilisant System.Collections.Generic; en utilisant System.Linq; en utilisant Xamarin.Forms; namespace DatabaseEmployeeCreation.SqlLite.ViewModel {classe publique DatabaseLogic {static object locker = new object (); Base de données 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. créer 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. afficher EmployeeDetails en dessous du code

```
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. Pour implémenter la méthode dans Android et la méthode 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();
            }
        }
    }
```

J'espère que cet exemple ci-dessus est très facile, j'ai expliqué

Lire Travailler avec des bases de données locales en ligne: https://riptutorial.com/fr/xamarin-

forms/topic/5997/travailler-avec-des-bases-de-donnees-locales

Chapitre 40: Travailler avec des cartes

Remarques

Si vous envisagez d'exécuter votre projet sur un autre ordinateur, vous devrez générer une nouvelle clé API, car les empreintes SHA-1 ne correspondront pas aux différentes machines de génération.

Vous pouvez explorer le projet, décrit dans l'exemple *Ajouter une carte dans Xamarin.Forms* ici

Examples

Ajouter une carte dans Xamarin.Forms (Xamarin Studio)

Vous pouvez simplement utiliser les API de carte natives sur chaque plate-forme avec Xamarin Forms. Il vous suffit de télécharger le package *Xamarin.Forms.Maps à* partir de nuget et de l'installer dans chaque projet (y compris le projet PCL).

Initialisation des cartes

Tout d'abord, vous devez ajouter ce code à vos projets spécifiques à la plateforme. Pour ce faire, vous devez ajouter l' `Xamarin.FormsMaps.Init` méthode `Xamarin.FormsMaps.Init` , comme dans les exemples ci-dessous.

---

projet iOS

Fichier 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);
    }
}
```

Projet Android

Fichier 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());
    }
}
```

Configuration de la plate-forme

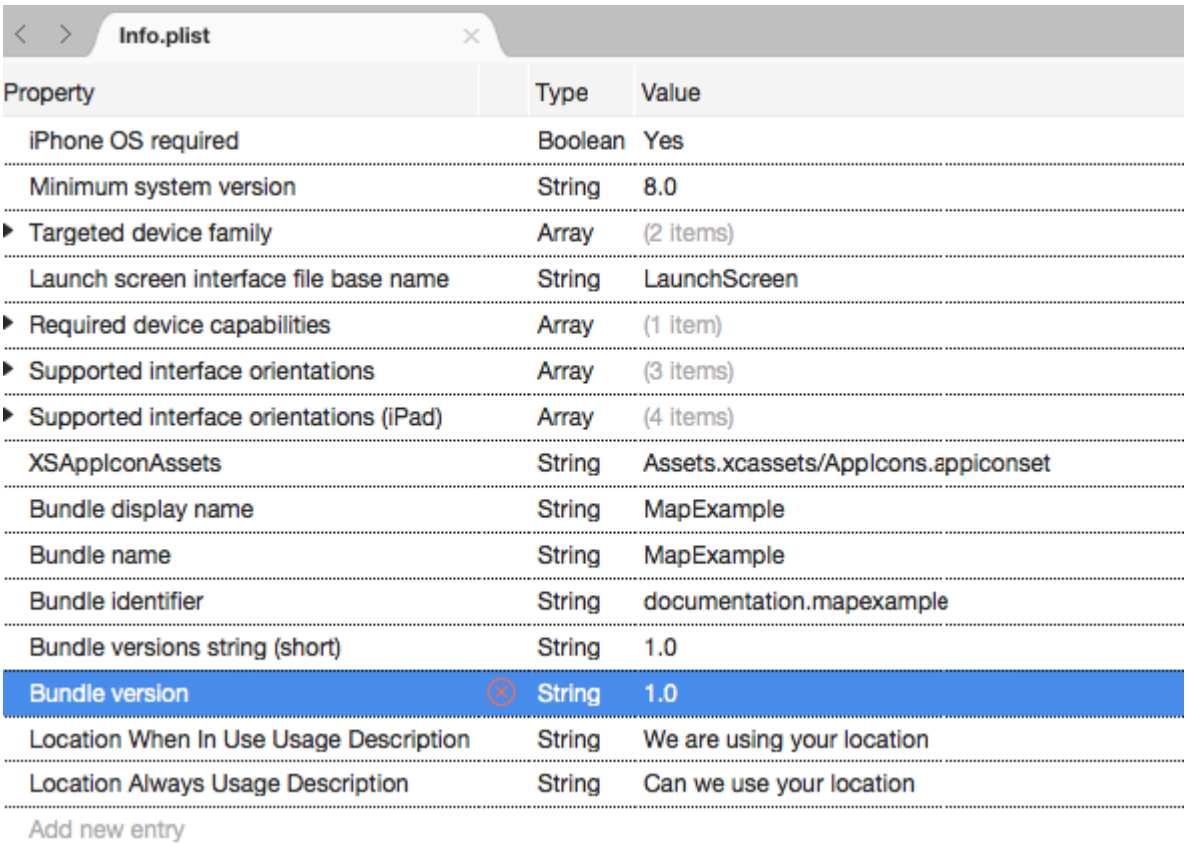
Des étapes de configuration supplémentaires sont nécessaires sur certaines plates-formes avant que la carte ne s'affiche.

---

projet iOS

Dans un projet iOS, il suffit d'ajouter 2 entrées à votre fichier *Info.plist* :

- *Chaîne* `NSLocationWhenInUseUsageDescription` avec la valeur `We are using your location`
- *Chaîne* `NSLocationAlwaysUsageDescription` avec valeur `Can we use your location`



Info.plist

Property	Type	Value
iPhone OS required	Boolean	Yes
Minimum system version	String	8.0
Targeted device family	Array	(2 items)
Launch screen interface file base name	String	LaunchScreen
Required device capabilities	Array	(1 item)
Supported interface orientations	Array	(3 items)
Supported interface orientations (iPad)	Array	(4 items)
XSAppIconAssets	String	Assets.xcassets/AppIcons.appiconset
Bundle display name	String	MapExample
Bundle name	String	MapExample
Bundle identifier	String	documentation.mapexample
Bundle versions string (short)	String	1.0
Bundle version	String	1.0
Location When In Use Usage Description	String	We are using your location
Location Always Usage Description	String	Can we use your location
Add new entry		

Projet Android

Pour utiliser Google Maps, vous devez générer une clé API et l'ajouter à votre projet. Suivez les instructions ci-dessous pour obtenir cette clé:

1. (Facultatif) Recherchez l'emplacement de votre outil keytool (par défaut, `/System/Library/Frameworks/JavaVM.framework/Versions/Current/Commands`)

2. (Facultatif) Ouvrez le terminal et accédez à votre répertoire d'outils:

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

3. Exécutez la commande keytool suivante:

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

   Où [NOM D'UTILISATEUR] est évidemment votre dossier d'utilisateur actuel. Vous devriez obtenir quelque chose de similaire dans la sortie:

   ```
   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
   ```

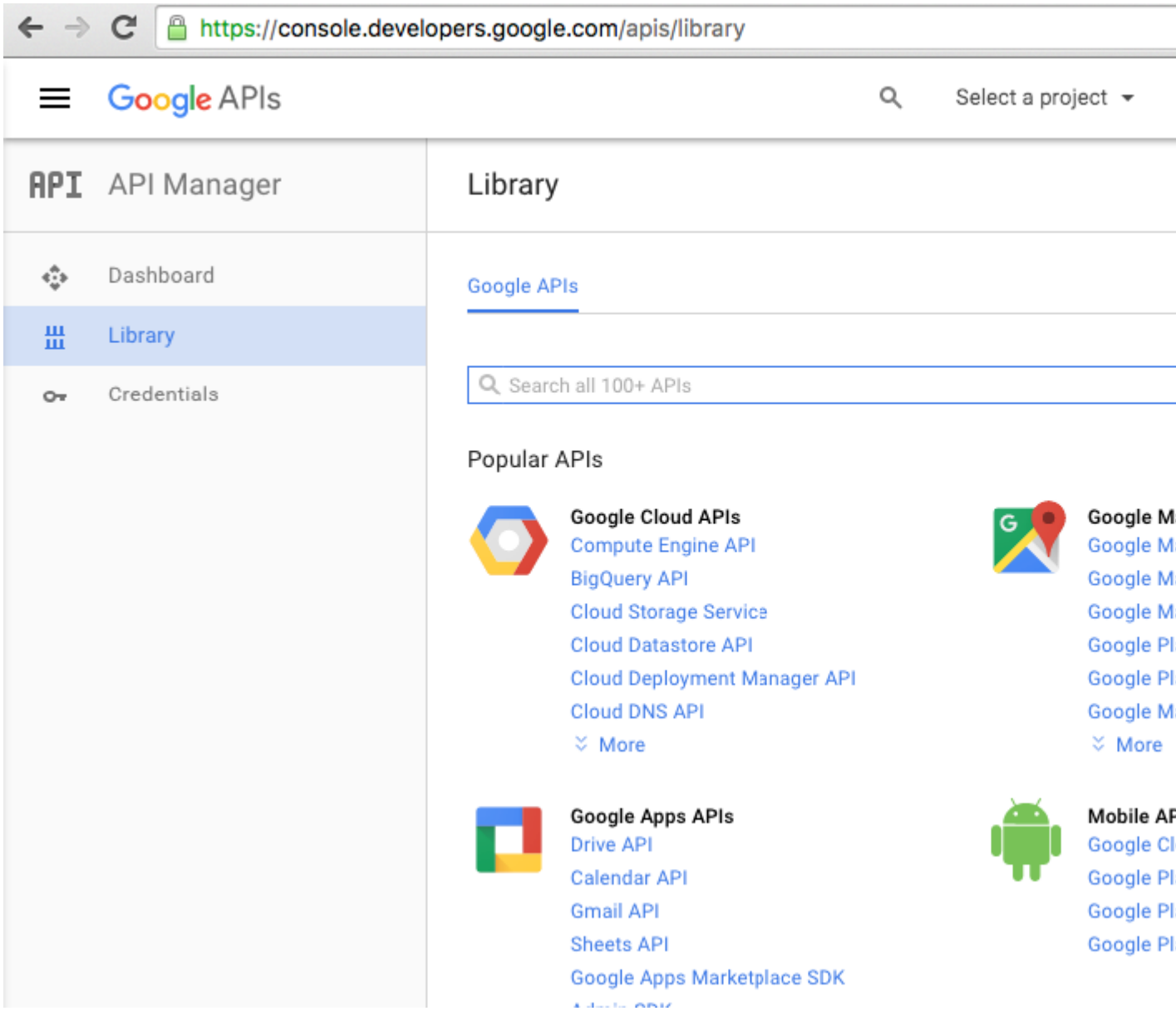
```
Valid from: Thu Jun 30 10:22:00 EEST 2016 until: Sat Jun 23 10:22:00 EEST 2046
Certificate fingerprints:
         MD5:  4E:49:A7:14:99:D6:AB:9F:AA:C7:07:E2:6A:1A:1D:CA
         SHA1: 57:A1:E5:23:CE:49:2F:17:8D:8A:EA:87:65:44:C1:DD:1C:DA:51:95
         SHA256:
70:E1:F3:5B:95:69:36:4A:82:A9:62:F3:67:B6:73:A4:DD:92:95:51:44:E3:4C:3D:9E:ED:99:03:09:9F:90:3F

         Signature algorithm name: SHA256withRSA
         Version: 3
```

4. Tout ce dont nous avons besoin dans cette sortie est l'empreinte du certificat SHA1. Dans notre cas, cela équivaut à ceci:

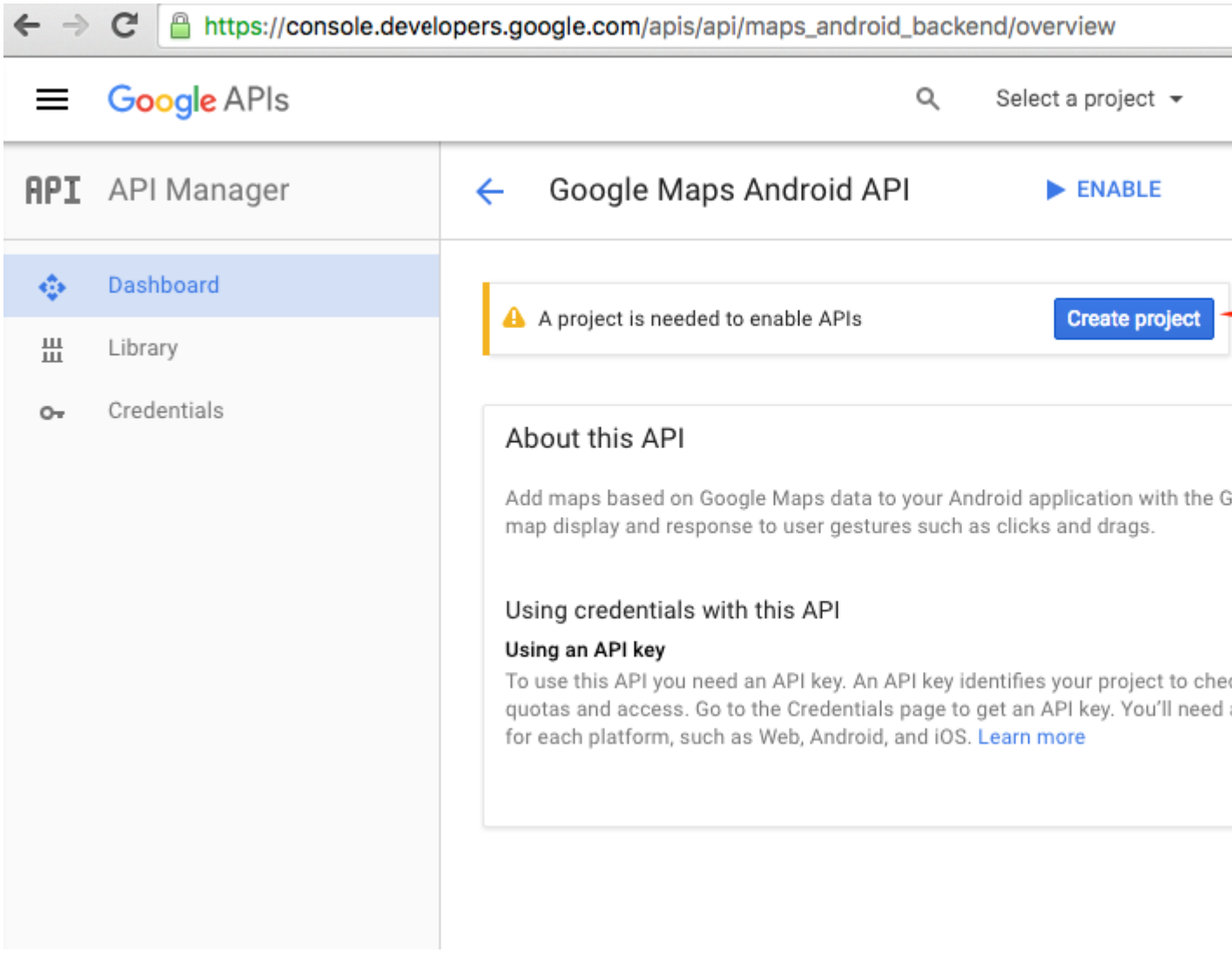
```
57:A1:E5:23:CE:49:2F:17:8D:8A:EA:87:65:44:C1:DD:1C:DA:51:95
```

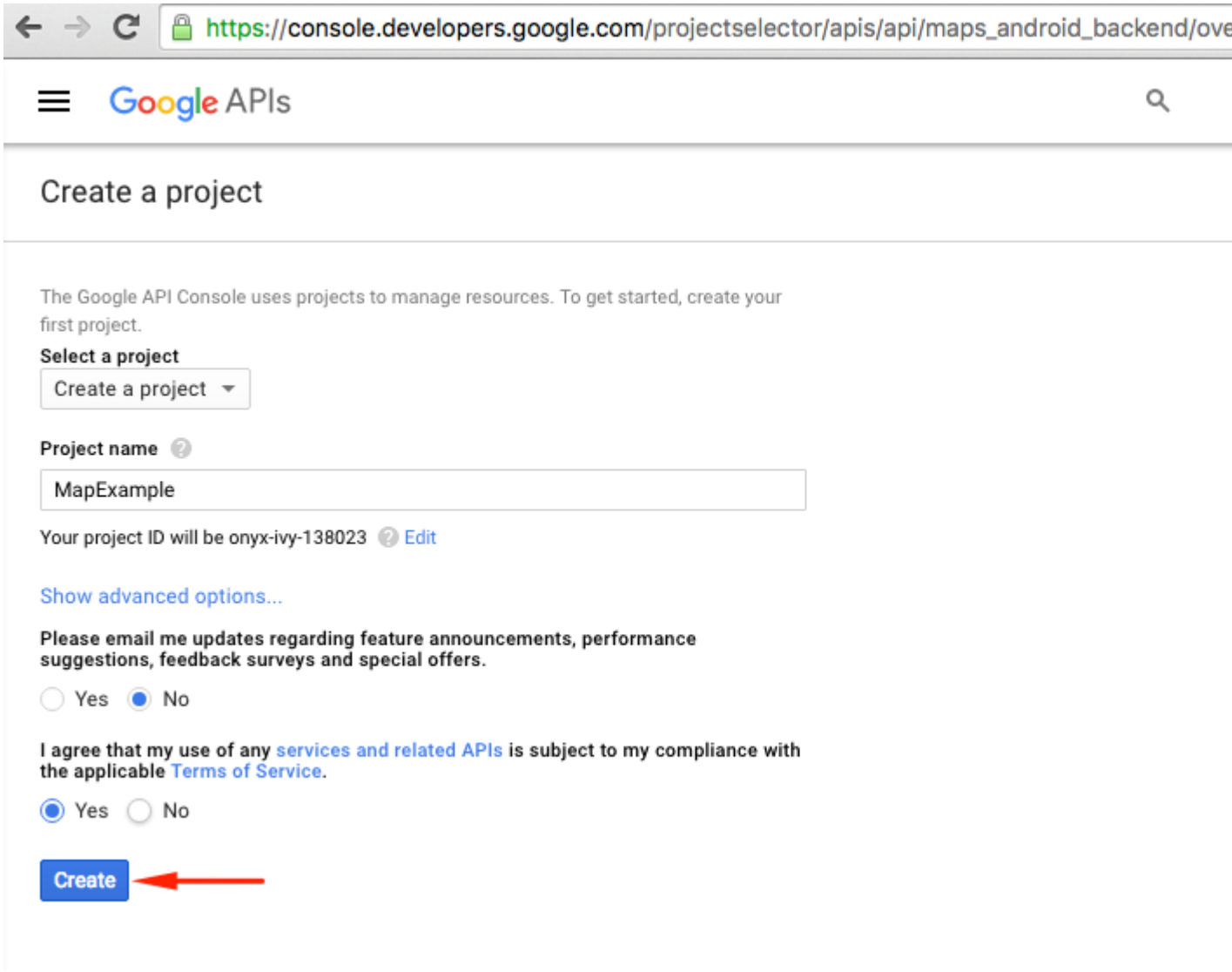
   Copiez ou enregistrez quelque part cette clé. Nous en aurons besoin plus tard.

5. Accédez à Google Developers Console , dans notre cas, nous devons ajouter l' API Google Maps Android , alors choisissez-la:

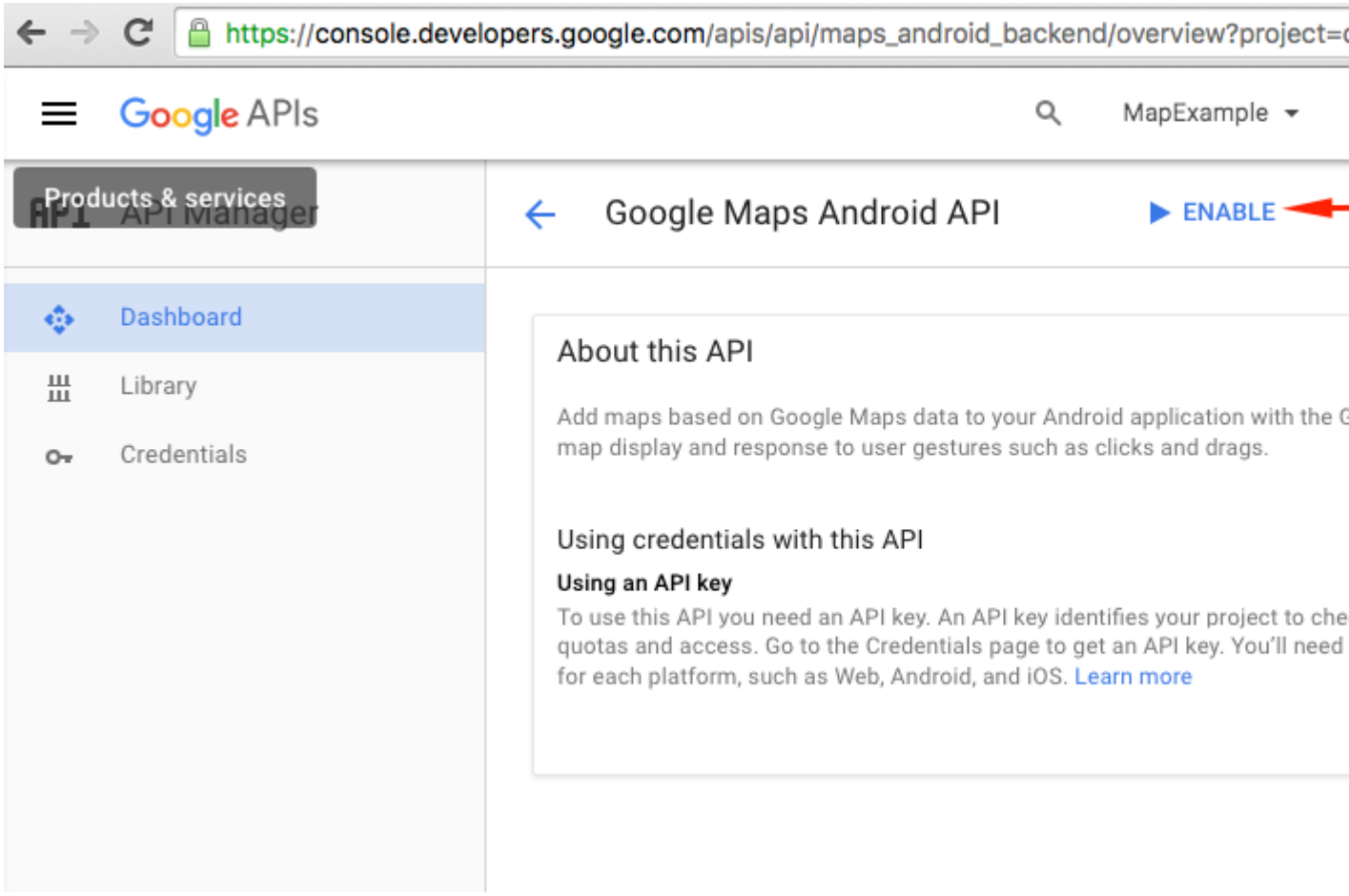


6. Google vous demandera de créer un projet pour activer les API, suivez cette astuce et créez le projet:

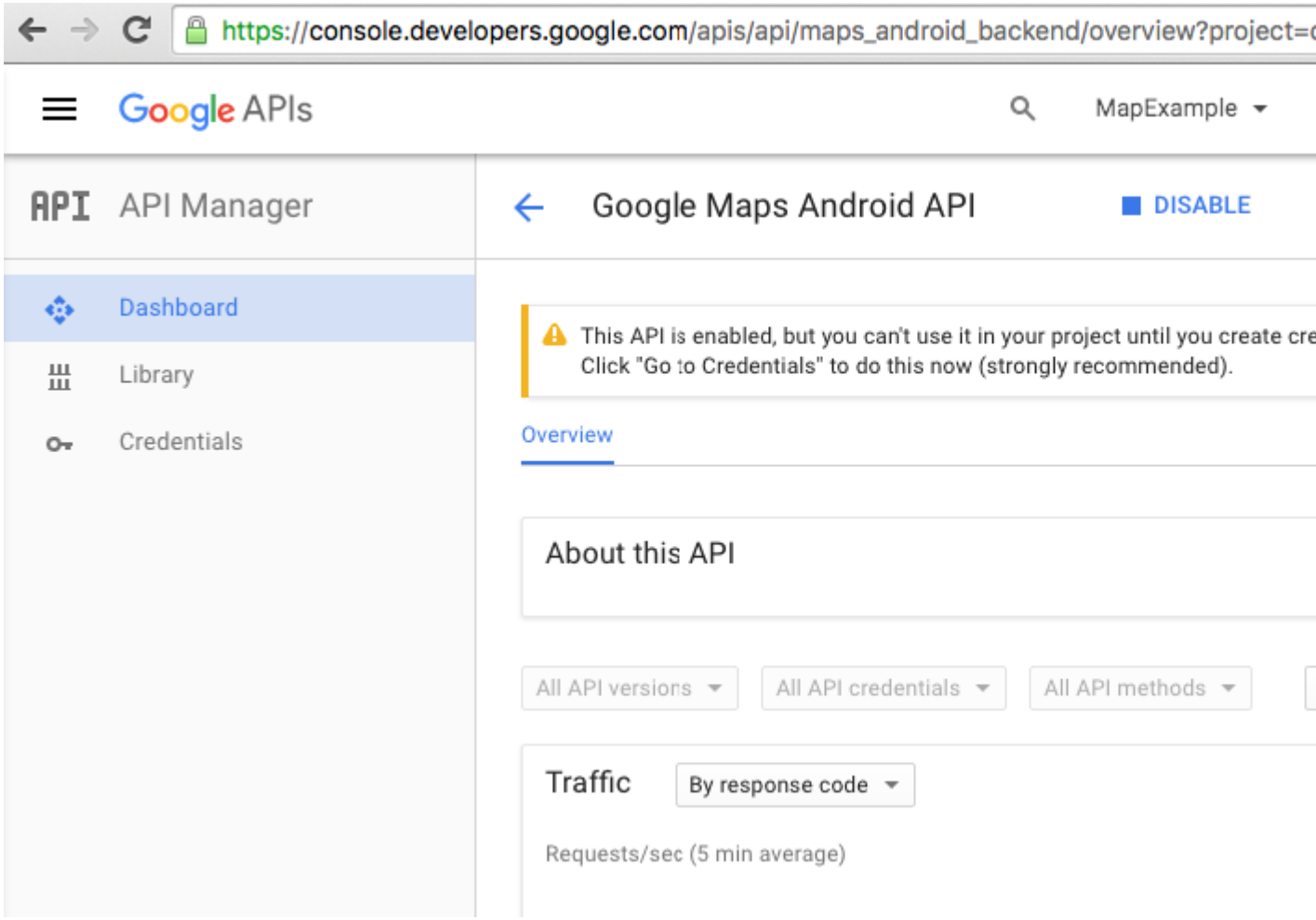




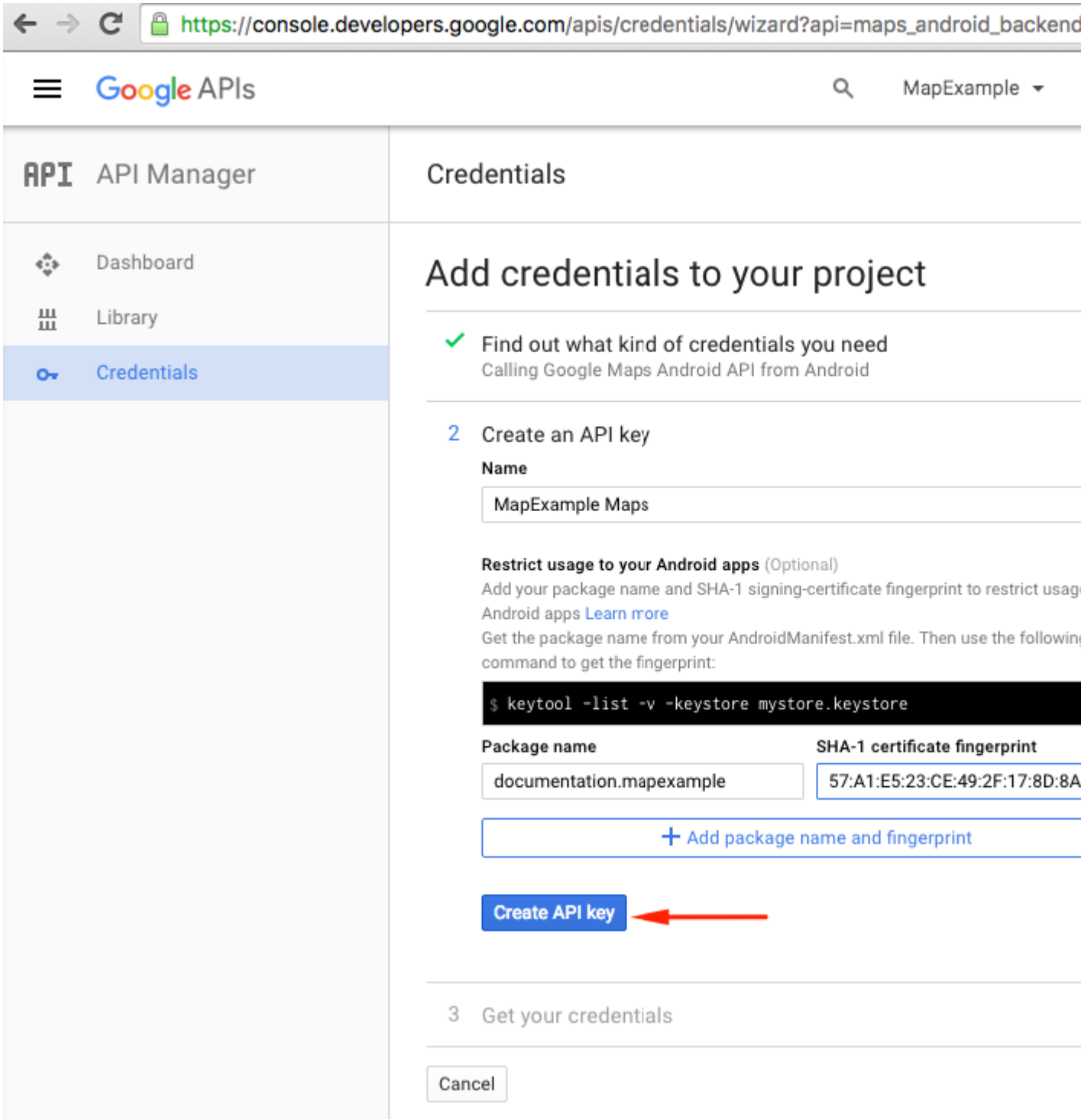
7. Activez l'API Google Maps pour votre projet:



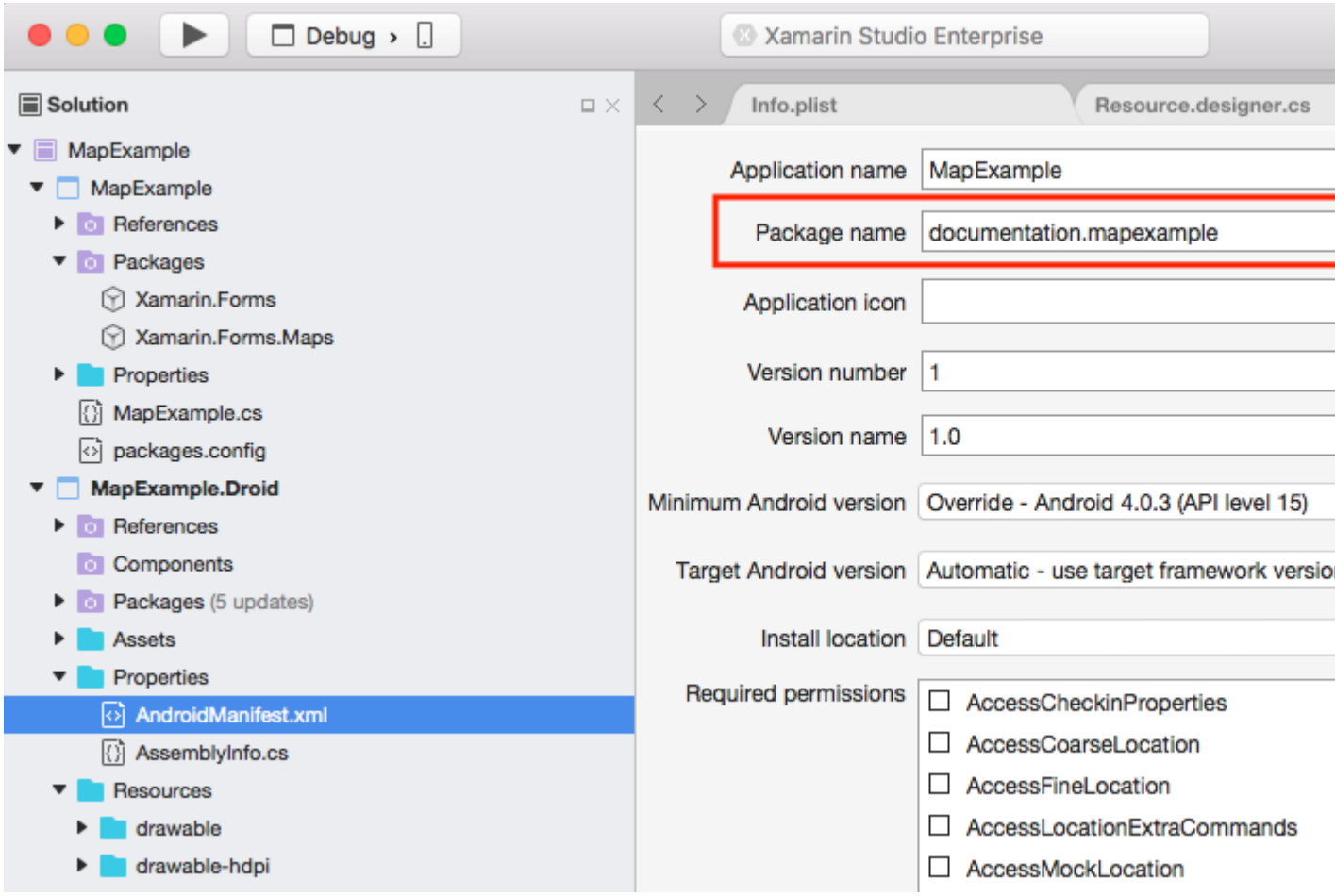
Après avoir activé api, vous devez créer des informations d'identification pour votre application. Suivez ce conseil:



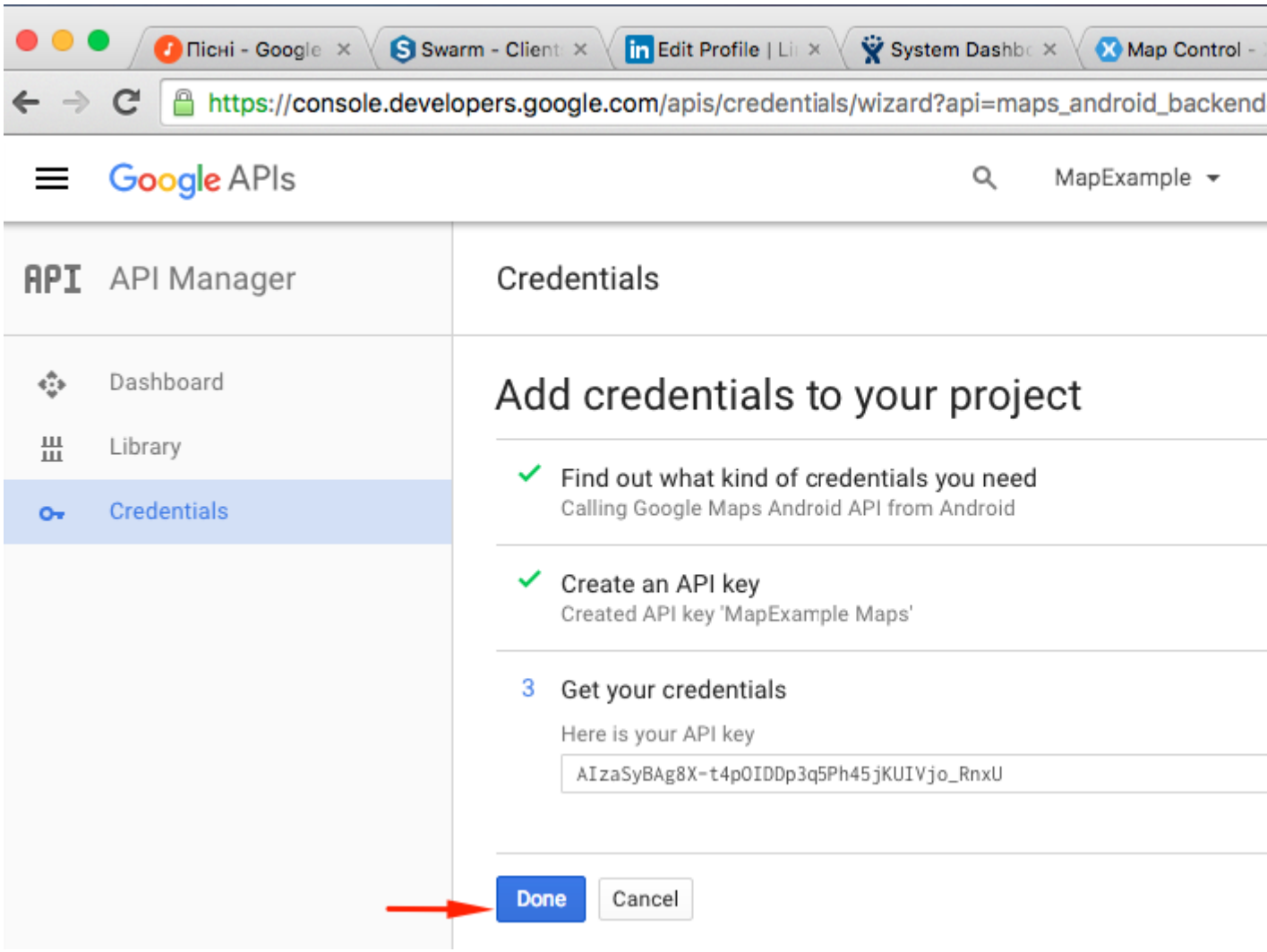
8. Sur la page suivante, choisissez la plate-forme Android, appuyez sur "Quelles sont les références dont j'ai besoin?" Cliquez sur le bouton, créez un nom pour votre clé API, tapez sur "Ajouter le nom et l'empreinte du package", entrez votre nom de package et votre empreinte SHA1 à l'étape 4 et créez enfin une clé API:



Pour trouver le nom de votre package dans Xamarin Studio, accédez à votre solution .Droid -> AndroidManifest.xml:



9. Après la création, copiez la nouvelle clé API (n'oubliez pas de cliquer sur le bouton "Terminé") et collez-la dans votre fichier `AndroidManifest.xml` :



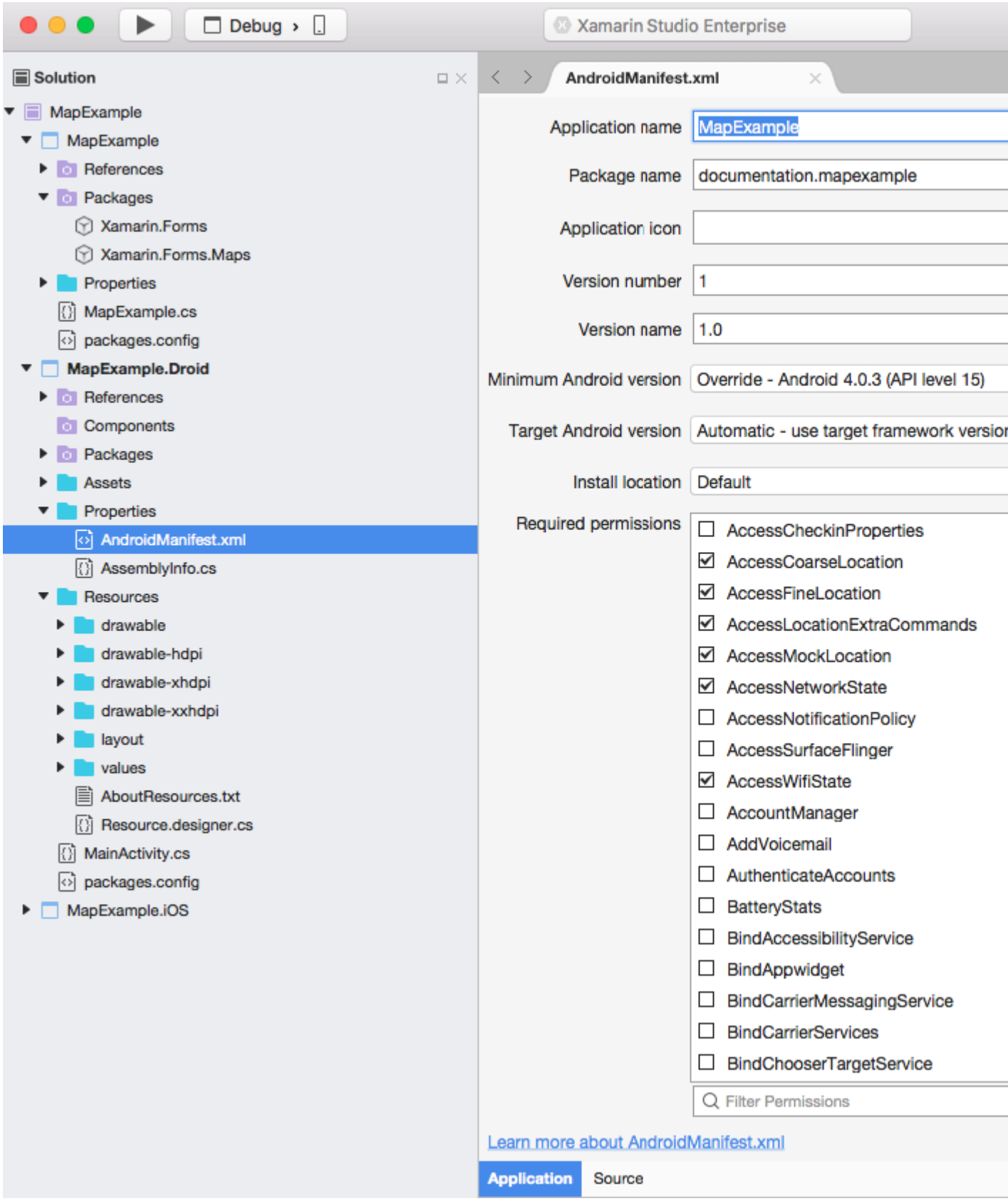
Fichier AndroidManifest.xml

```
<?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>
```

Vous devrez également activer certaines autorisations dans votre manifeste pour activer certaines fonctionnalités supplémentaires:

- Accès Emplacement grossier
- Accès Fine Location

- Accès Emplacement Commandes supplémentaires
- Emplacement du mock d'accès
- État du réseau d'accès
- Accès Wifi State
- l'Internet



Bien que les deux dernières autorisations soient nécessaires pour télécharger des données

Maps. Lisez les autorisations sur Android pour en savoir plus. C'est toutes les étapes pour la configuration Android.

> *Remarque* : si vous souhaitez exécuter votre application sur Android simulator, vous devez y installer les services Google Play. Suivez ce tutoriel pour installer Play Services sur Xamarin Android Player. Si vous ne trouvez pas la mise à jour des services Google Play après l'installation de Play Store, vous pouvez la mettre à jour directement depuis votre application, où vous êtes dépendant des services de cartes.

Ajouter une carte

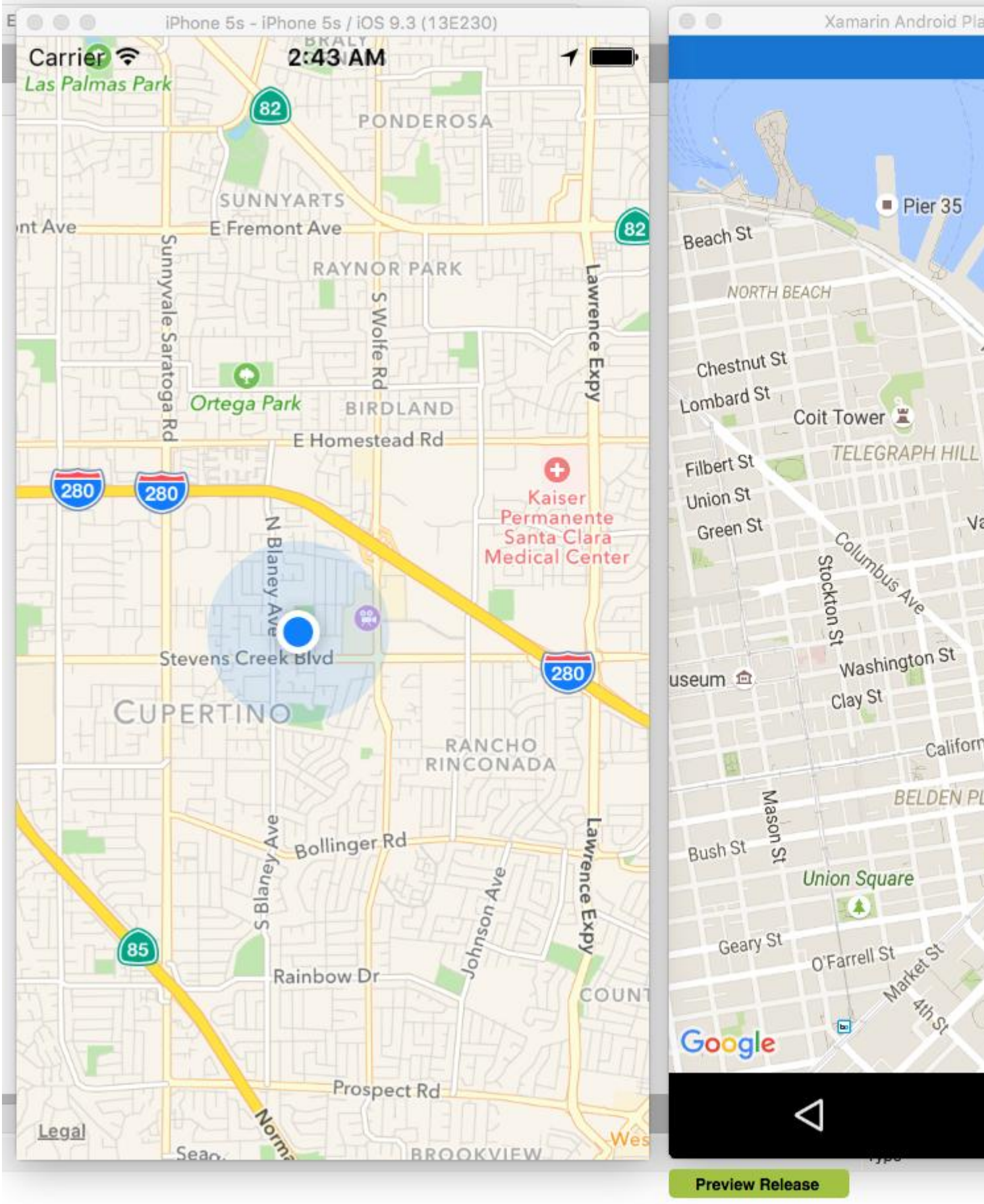
L'ajout d'une vue cartographique à votre projet de plateforme croisée est assez simple. Voici un exemple de la façon dont vous pouvez le faire (j'utilise un projet PCL sans XAML).

Projet PCL

Fichier MapExample.cs

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

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

        MainPage = rootPage;
    }
}
```

C'est tout. Maintenant, si vous exécutez votre application sur iOS ou Android, la carte s'affiche:



Lire Travailler avec des cartes en ligne: https://riptutorial.com/fr/xamarin-forms/topic/3917/travailler-avec-des-cartes

Chapitre 41: Utilisation de ListViews

Introduction

Cette documentation explique comment utiliser les différents composants de Xamarin Forms ListView

Examples

Tirez pour actualiser dans XAML et code derrière

Pour activer l'option Pull to Refresh dans `ListView` dans Xamarin, vous devez d'abord spécifier qu'il s'agit de `PullToRefresh` activé, puis spécifier le nom de la commande à appeler lors de l' `ListView` :

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

La même chose peut être obtenue dans le code derrière:

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

Ensuite, vous devez spécifier ce que la commande d' `Refresh` fait dans votre code derrière:

```
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;
    }
}
```

Lire Utilisation de ListViews en ligne: https://riptutorial.com/fr/xamarin-forms/topic/9487/utilisation-de-listviews

Chapitre 42: Vues Xamarin.Forms

Examples

Bouton

Le **bouton** est probablement le contrôle le plus courant non seulement dans les applications mobiles, mais aussi dans toutes les applications disposant d'une interface utilisateur. Le concept d'un bouton a trop d'objectifs à énumérer ici. En règle générale cependant, vous utiliserez un bouton pour permettre aux utilisateurs de lancer une action ou une opération dans votre application. Cette opération peut inclure tout, de la navigation de base dans votre application, à la soumission de données à un service Web quelque part sur Internet.

XAML

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

XAML Code-Behind

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

Code

```
var button = new Button( )
{
    Text = "Hello, Forms !",
    VerticalOptions = LayoutOptions.CenterAndExpand,
    HorizontalOptions = LayoutOptions.CenterAndExpand,
    TextColor = Color.Red,
    BorderColor = Color.Blue,
};

button.Clicked += ( sender, args ) =>
{
    var b = (Button) sender;
    b.Text = "I've been clicked!";
};
```

Sélecteur de date

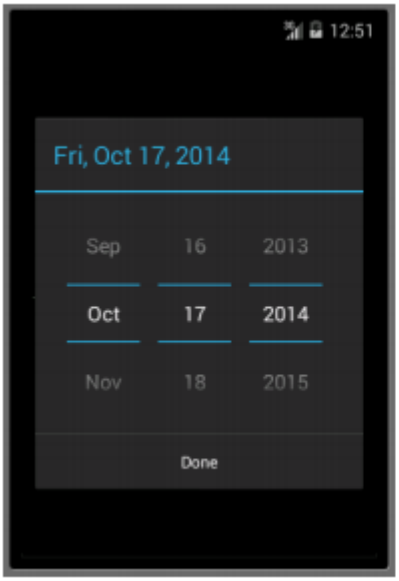
Très souvent, dans les applications mobiles, il y aura une raison de traiter les dates. Lorsque vous travaillez avec des dates, vous aurez probablement besoin d'une sorte de saisie utilisateur pour sélectionner une date. Cela peut se produire lorsque vous travaillez avec une application de planification ou de calendrier. Dans ce cas, il est préférable de fournir aux utilisateurs un contrôle spécialisé leur permettant de sélectionner une date de manière interactive, plutôt que de demander aux utilisateurs de saisir manuellement une date. C'est là que le contrôle DatePicker est vraiment utile.

XAML

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

Code

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



Entrée

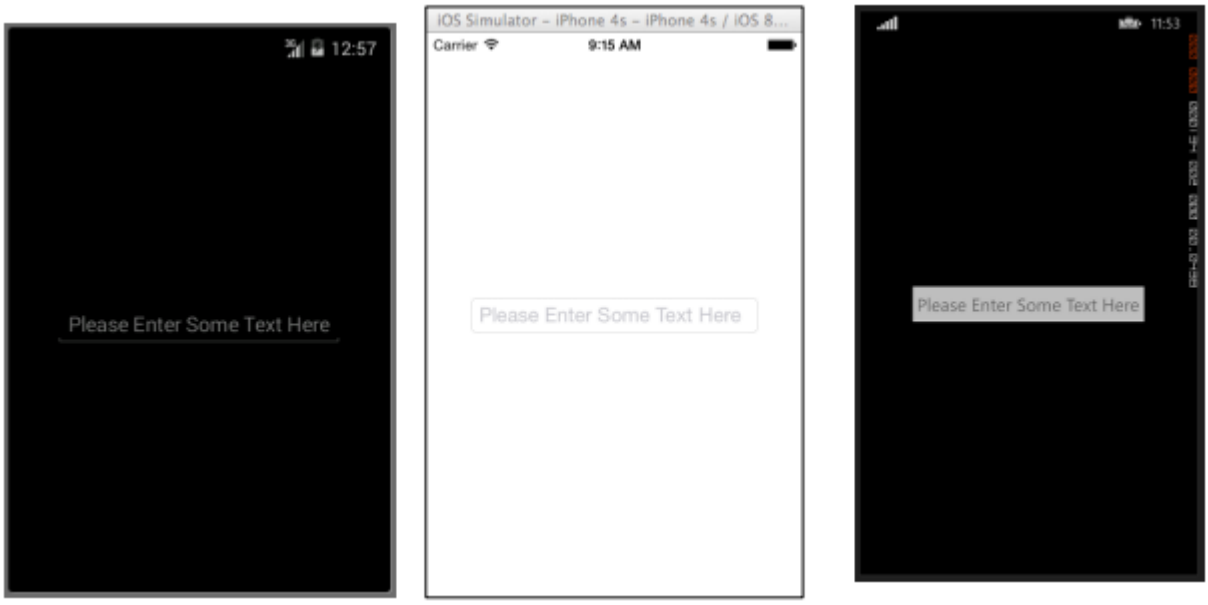
La vue d'entrée permet aux utilisateurs de saisir une seule ligne de texte. Cette seule ligne de texte peut être utilisée à des fins multiples, y compris la saisie de notes de base, d'informations d'identification, d'URL, etc. Cette vue est une vue polyvalente, ce qui signifie que si vous avez besoin de taper du texte normal ou que vous souhaitez masquer un mot de passe, tout se fait via ce seul contrôle.

XAML

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

Code

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



Éditeur

L'éditeur est très similaire à l'entrée en ce sens qu'il permet aux utilisateurs d'entrer du texte libre. La différence est que l'éditeur permet une entrée multi-lignes alors que l'entrée n'est utilisée que pour une entrée sur une seule ligne. L'entrée fournit également quelques propriétés supplémentaires par rapport à l'éditeur pour permettre une personnalisation supplémentaire de la vue.

XAML

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

Code

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

Image

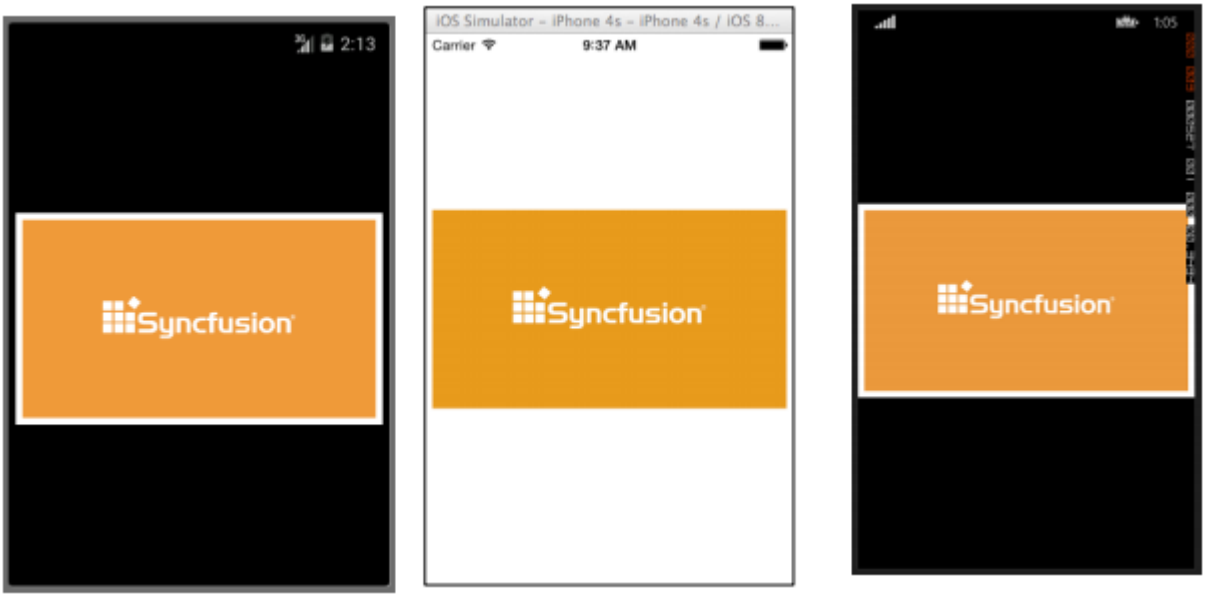
Les images sont des parties très importantes de toute application. Ils permettent d'injecter des éléments visuels supplémentaires ainsi que des marques dans votre application. Sans oublier que les images sont généralement plus intéressantes à regarder que du texte ou des boutons. Vous pouvez utiliser une image en tant qu'élément autonome dans votre application, mais un élément Image peut également être ajouté à d'autres éléments d'affichage, tels qu'un bouton.

XAML

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

Code

```
var image = new Image {
Aspect = Aspect.AspectFit,
Source = ImageSource.FromUri(new Uri("http://d2g29cya9iq7ip.cloudfron
t.net/content/images/company/aboutus-video-bg.png?v=25072014072745"))
};
```



Étiquette

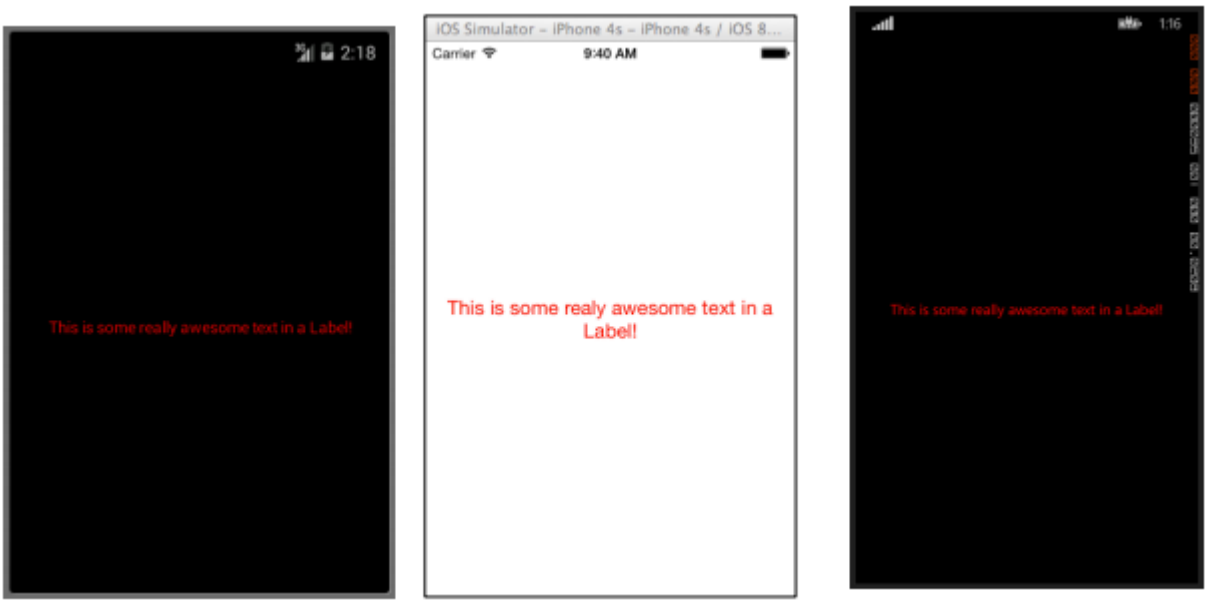
Croyez-le ou non, le Label est l'une des classes View les plus cruciales, mais encore sous-estimées, non seulement dans Xamarin.Forms, mais dans le développement de l'interface utilisateur en général. Il est considéré comme une ligne de texte plutôt ennuyeuse, mais sans cette ligne de texte, il serait très difficile de transmettre certaines idées à l'utilisateur. Les contrôles d'étiquette peuvent être utilisés pour décrire ce que l'utilisateur doit entrer dans un contrôle d'édition ou de saisie. Ils peuvent décrire une section de l'interface utilisateur et lui donner un contexte. Ils peuvent être utilisés pour afficher le total dans une application de calculatrice. Oui, l'étiquette est vraiment le contrôle le plus polyvalent de votre trousse à outils, qui ne suscite pas toujours beaucoup d'attention, mais c'est le premier à être remarqué s'il n'y en a pas.

XAML

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

Code

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



Lire Vues Xamarin.Forms en ligne: https://riptutorial.com/fr/xamarin-forms/topic/7369/vues-xamarin-forms

Chapitre 43: Xamarin Plugin

Examples

Share Plugin

Un moyen simple de partager un message ou un lien, de copier du texte dans le presse-papiers ou d'ouvrir un navigateur dans n'importe quelle application Xamarin ou Windows.

Disponible sur NuGet: https://www.nuget.org/packages/Plugin.Share/

XAML

```
<StackLayout Padding="20" Spacing="20">
      <Button StyleId="Text" Text="Share Text" Clicked="Button_OnClicked"/>
      <Button StyleId="Link" Text="Share Link" Clicked="Button_OnClicked"/>
      <Button StyleId="Browser" Text="Open Browser" Clicked="Button_OnClicked"/>
      <Label Text=""/>

 </StackLayout>
```

C #

```
 async void Button_OnClicked(object sender, EventArgs e)
        {
            switch (((Button)sender).StyleId)
            {
                case "Text":
                    await CrossShare.Current.Share("Follow @JamesMontemagno on Twitter",
"Share");
                    break;
                case "Link":
                    await CrossShare.Current.ShareLink("http://motzcod.es", "Checkout my
blog", "MotzCod.es");
                    break;
                case "Browser":
                    await CrossShare.Current.OpenBrowser("http://motzcod.es");
                    break;
            }
        }
```

Cartes externes

Plug-ins de cartes externes Ouvrez des cartes externes pour accéder à une géolocalisation ou une adresse spécifique. Option pour lancer avec l'option de navigation sur iOS aussi.

Disponible sur NuGet: [https://www.nuget.org/packages/Xam.Plugin.ExternalMaps/][1]

XAML

```
  <StackLayout Spacing="10" Padding="10">
```

```
      <Button x:Name="navigateAddress" Text="Navigate to Address"/>
      <Button x:Name="navigateLatLong" Text="Navigate to Lat|Long"/>
      <Label Text=""/>

    </StackLayout>
```

Code

```
namespace PluginDemo
{
    public partial class ExternalMaps : ContentPage
    {
        public ExternalMaps()
        {
            InitializeComponent();
            navigateLatLong.Clicked += (sender, args) =>
            {
                CrossExternalMaps.Current.NavigateTo("Space Needle", 47.6204, -122.3491);
            };

            navigateAddress.Clicked += (sender, args) =>
            {
                CrossExternalMaps.Current.NavigateTo("Xamarin", "394 pacific ave.", "San
Francisco", "CA", "94111", "USA", "USA");
            };
        }
    }
}
```

Plugin Geolocator

Accédez facilement à la géolocalisation via Xamarin.iOS, Xamarin.Android et 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>
```

Code

```
namespace PluginDemo
{
    public partial class GeolocatorPage : ContentPage
    {
        public GeolocatorPage()
        {
            InitializeComponent();
            buttonGetGPS.Clicked += async (sender, args) =>
```

```
            {
                try
                {
                    var locator = CrossGeolocator.Current;
                    locator.DesiredAccuracy = 1000;
                    labelGPS.Text = "Getting gps";

                    var position = await locator.GetPositionAsync(timeoutMilliseconds: 10000);

                    if (position == null)
                    {
                        labelGPS.Text = "null gps :(";
                        return;
                    }
                    labelGPS.Text = string.Format("Time: {0} \nLat: {1} \nLong: {2}
\nAltitude: {3} \nAltitude Accuracy: {4} \nAccuracy: {5} \nHeading: {6} \nSpeed: {7}",
                        position.Timestamp, position.Latitude, position.Longitude,
                        position.Altitude, position.AltitudeAccuracy, position.Accuracy,
position.Heading, position.Speed);

                }
                catch //(Exception ex)
                {
                   // Xamarin.Insights.Report(ex);
                   // await DisplayAlert("Uh oh", "Something went wrong, but don't worry we
captured it in Xamarin Insights! Thanks.", "OK");
                }
            };

            buttonTrack.Clicked += async (object sender, EventArgs e) =>
            {
                try
                {
                    if (CrossGeolocator.Current.IsListening)
                    {
                        await CrossGeolocator.Current.StopListeningAsync();
                        labelGPSTrack.Text = "Stopped tracking";
                        buttonTrack.Text = "Stop Tracking";
                    }
                    else
                    {
                        if (await CrossGeolocator.Current.StartListeningAsync(30000, 0))
                        {
                            labelGPSTrack.Text = "Started tracking";
                            buttonTrack.Text = "Track Movements";
                        }
                    }
                }
                catch //(Exception ex)
                {
                    //Xamarin.Insights.Report(ex);
                   // await DisplayAlert("Uh oh", "Something went wrong, but don't worry we
captured it in Xamarin Insights! Thanks.", "OK");
                }
            };
        }

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

```
            {
                CrossGeolocator.Current.PositionChanged +=
CrossGeolocator_Current_PositionChanged;
                CrossGeolocator.Current.PositionError +=
CrossGeolocator_Current_PositionError;
            }
            catch
            {
            }
        }

        void CrossGeolocator_Current_PositionError(object sender,
Plugin.Geolocator.Abstractions.PositionErrorEventArgs e)
        {

            labelGPSTrack.Text = "Location error: " + e.Error.ToString();
        }

        void CrossGeolocator_Current_PositionChanged(object sender,
Plugin.Geolocator.Abstractions.PositionEventArgs e)
        {
            var position = e.Position;
            labelGPSTrack.Text = string.Format("Time: {0} \nLat: {1} \nLong: {2} \nAltitude:
{3} \nAltitude Accuracy: {4} \nAccuracy: {5} \nHeading: {6} \nSpeed: {7}",
                position.Timestamp, position.Latitude, position.Longitude,
                position.Altitude, position.AltitudeAccuracy, position.Accuracy,
position.Heading, position.Speed);


        }

        protected override void OnDisappearing()
        {
            base.OnDisappearing();
            try
            {
                CrossGeolocator.Current.PositionChanged -=
CrossGeolocator_Current_PositionChanged;
                CrossGeolocator.Current.PositionError -=
CrossGeolocator_Current_PositionError;
            }
            catch
            {
            }
        }
    }
}
```

Plugin Média

Prenez ou choisissez des photos et des vidéos à partir d'une API multi-plateforme.

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>
```

Code

```
namespace PluginDemo
{
    public partial class MediaPage : ContentPage
    {
        public MediaPage()
        {
            InitializeComponent();
            takePhoto.Clicked += async (sender, args) =>
            {

                if (!CrossMedia.Current.IsCameraAvailable ||
!CrossMedia.Current.IsTakePhotoSupported)
                {
                    await DisplayAlert("No Camera", ":( No camera avaialble.", "OK");
                    return;
                }
                try
                {
                    var file = await CrossMedia.Current.TakePhotoAsync(new
Plugin.Media.Abstractions.StoreCameraMediaOptions
                    {
                        Directory = "Sample",
                        Name = "test.jpg",
                        SaveToAlbum = saveToGallery.IsToggled
                    });

                    if (file == null)
                        return;

                    await DisplayAlert("File Location", (saveToGallery.IsToggled ?
file.AlbumPath : file.Path), "OK");

                    image.Source = ImageSource.FromStream(() =>
                    {
                        var stream = file.GetStream();
                        file.Dispose();
                        return stream;
                    });
                }
                catch //(Exception ex)
                {
                   // Xamarin.Insights.Report(ex);
                   // await DisplayAlert("Uh oh", "Something went wrong, but don't worry we
captured it in Xamarin Insights! Thanks.", "OK");
                }
            };
```

```
            pickPhoto.Clicked += async (sender, args) =>
            {
                if (!CrossMedia.Current.IsPickPhotoSupported)
                {
                    await DisplayAlert("Photos Not Supported", ":( Permission not granted to
photos.", "OK");
                    return;
                }
                try
                {
                    Stream stream = null;
                    var file = await CrossMedia.Current.PickPhotoAsync().ConfigureAwait(true);


                    if (file == null)
                        return;

                    stream = file.GetStream();
                    file.Dispose();

                    image.Source = ImageSource.FromStream(() => stream);

                }
                catch //(Exception ex)
                {
                   // Xamarin.Insights.Report(ex);
                   // await DisplayAlert("Uh oh", "Something went wrong, but don't worry we
captured it in Xamarin Insights! Thanks.", "OK");
                }
            };

            takeVideo.Clicked += async (sender, args) =>
            {
                if (!CrossMedia.Current.IsCameraAvailable ||
!CrossMedia.Current.IsTakeVideoSupported)
                {
                    await DisplayAlert("No Camera", ":( No camera avaialble.", "OK");
                    return;
                }

                try
                {
                    var file = await CrossMedia.Current.TakeVideoAsync(new
Plugin.Media.Abstractions.StoreVideoOptions
                    {
                        Name = "video.mp4",
                        Directory = "DefaultVideos",
                        SaveToAlbum = saveToGallery.IsToggled
                    });

                    if (file == null)
                        return;

                    await DisplayAlert("Video Recorded", "Location: " +
(saveToGallery.IsToggled ? file.AlbumPath : file.Path), "OK");

                    file.Dispose();

                }
                catch //(Exception ex)
                {
```

```
                    // Xamarin.Insights.Report(ex);
                    // await DisplayAlert("Uh oh", "Something went wrong, but don't worry we
captured it in Xamarin Insights! Thanks.", "OK");
                }
            };

            pickVideo.Clicked += async (sender, args) =>
            {
                if (!CrossMedia.Current.IsPickVideoSupported)
                {
                    await DisplayAlert("Videos Not Supported", ":( Permission not granted to
videos.", "OK");
                    return;
                }
                try
                {
                    var file = await CrossMedia.Current.PickVideoAsync();

                    if (file == null)
                        return;

                    await DisplayAlert("Video Selected", "Location: " + file.Path, "OK");
                    file.Dispose();

                }
                catch //(Exception ex)
                {
                    //Xamarin.Insights.Report(ex);
                    //await DisplayAlert("Uh oh", "Something went wrong, but don't worry we
captured it in Xamarin Insights! Thanks.", "OK");
                }
            };
        }
    }
}
```

Plugin de messagerie

Plug-in de messagerie pour Xamarin et Windows pour passer un appel, envoyer un SMS ou envoyer un courrier électronique en utilisant les applications de messagerie par défaut sur les différentes plates-formes mobiles.

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>
```

Code

```
namespace PluginDemo
{
    public partial class MessagingPage : ContentPage
    {
        public MessagingPage()
        {
            InitializeComponent();
            buttonCall.Clicked += async (sender, e) =>
            {
                try
                {
                    // Make Phone Call
                    var phoneCallTask = MessagingPlugin.PhoneDialer;
                    if (phoneCallTask.CanMakePhoneCall)
                        phoneCallTask.MakePhoneCall(phone.Text);
                    else
                        await DisplayAlert("Error", "This device can't place calls", "OK");
                }
                catch
                {
                   // await DisplayAlert("Error", "Unable to perform action", "OK");
                }
            };

            buttonSms.Clicked += async (sender, e) =>
            {
                try
                {

                    var smsTask = MessagingPlugin.SmsMessenger;
                    if (smsTask.CanSendSms)
                        smsTask.SendSms(phone.Text, "Hello World");
                    else
                        await DisplayAlert("Error", "This device can't send sms", "OK");
                }
                catch
                {
                   // await DisplayAlert("Error", "Unable to perform action", "OK");
                }
            };

            buttonEmail.Clicked += async (sender, e) =>
            {
                try
                {
                    var emailTask = MessagingPlugin.EmailMessenger;
                    if (emailTask.CanSendEmail)
                        emailTask.SendEmail(email.Text, "Hello there!", "This was sent from
the Xamrain Messaging Plugin from shared code!");
                    else
                        await DisplayAlert("Error", "This device can't send emails", "OK");
                }
                catch
                {
//await DisplayAlert("Error", "Unable to perform action", "OK");
                }
            };
        }
    }
}
```

Plugin d'autorisations

Vérifiez si vos utilisateurs ont accordé ou refusé des autorisations pour les groupes de permissions communs sur iOS et Android.

De plus, vous pouvez demander des autorisations avec une simple API asynchrone / attendue multiplate-forme.

Nuget disponible: https://www.nuget.org/packages/Plugin.Permissions entrez la description du lien ici **XAML**

XAML

```
 <StackLayout Padding="30" Spacing="10">
      <Button Text="Get Location" Clicked="Button_OnClicked"></Button>
      <Label x:Name="LabelGeolocation"></Label>
      <Button Text="Calendar" StyleId="Calendar"
Clicked="ButtonPermission_OnClicked"></Button>
      <Button Text="Camera" StyleId="Camera" Clicked="ButtonPermission_OnClicked"></Button>
      <Button Text="Contacts" StyleId="Contacts"
Clicked="ButtonPermission_OnClicked"></Button>
      <Button Text="Microphone" StyleId="Microphone"
Clicked="ButtonPermission_OnClicked"></Button>
      <Button Text="Phone" StyleId="Phone" Clicked="ButtonPermission_OnClicked"></Button>
      <Button Text="Photos" StyleId="Photos" Clicked="ButtonPermission_OnClicked"></Button>
      <Button Text="Reminders" StyleId="Reminders"
Clicked="ButtonPermission_OnClicked"></Button>
      <Button Text="Sensors" StyleId="Sensors" Clicked="ButtonPermission_OnClicked"></Button>
      <Button Text="Sms" StyleId="Sms" Clicked="ButtonPermission_OnClicked"></Button>
      <Button Text="Storage" StyleId="Storage" Clicked="ButtonPermission_OnClicked"></Button>
      <Label Text=""/>

    </StackLayout>
```

Code

```
bool busy;
        async void ButtonPermission_OnClicked(object sender, EventArgs e)
        {
            if (busy)
                return;

            busy = true;
            ((Button)sender).IsEnabled = false;

            var status = PermissionStatus.Unknown;
            switch (((Button)sender).StyleId)
            {
                case "Calendar":
                    status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Calendar);
                    break;
                case "Camera":
                    status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Camera);
                    break;
```

```
                case "Contacts":
                    status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Contacts);
                    break;
                case "Microphone":
                    status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Microphone);
                    break;
                case "Phone":
                    status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Phone);
                    break;
                case "Photos":
                    status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Photos);
                    break;
                case "Reminders":
                    status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Reminders);
                    break;
                case "Sensors":
                    status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Sensors);
                    break;
                case "Sms":
                    status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Sms);
                    break;
                case "Storage":
                    status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Storage);
                    break;
            }

            await DisplayAlert("Results", status.ToString(), "OK");

            if (status != PermissionStatus.Granted)
            {
                switch (((Button)sender).StyleId)
                {
                    case "Calendar":
                        status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Calendar))[Permission.Calendar];
                        break;
                    case "Camera":
                        status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Camera))[Permission.Camera];
                        break;
                    case "Contacts":
                        status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Contacts))[Permission.Contacts];
                        break;
                    case "Microphone":
                        status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Microphone))[Permission.Microphone];

                        break;
                    case "Phone":
                        status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Phone))[Permission.Phone];
                        break;
```

```
                    case "Photos":
                        status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Photos))[Permission.Photos];
                        break;
                    case "Reminders":
                        status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Reminders))[Permission.Reminders];
                        break;
                    case "Sensors":
                        status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Sensors))[Permission.Sensors];
                        break;
                    case "Sms":
                        status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Sms))[Permission.Sms];
                        break;
                    case "Storage":
                        status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Storage))[Permission.Storage];
                        break;
                }

                await DisplayAlert("Results", status.ToString(), "OK");

            }

            busy = false;
            ((Button)sender).IsEnabled = true;
        }

        async void Button_OnClicked(object sender, EventArgs e)
        {
            if (busy)
                return;

            busy = true;
            ((Button)sender).IsEnabled = false;

            try
            {
                var status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Location);
                if (status != PermissionStatus.Granted)
                {
                    if (await
CrossPermissions.Current.ShouldShowRequestPermissionRationaleAsync(Permission.Location))
                    {
                        await DisplayAlert("Need location", "Gunna need that location", "OK");
                    }

                    var results = await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Location);
                    status = results[Permission.Location];
                }

                if (status == PermissionStatus.Granted)
                {
                    var results = await CrossGeolocator.Current.GetPositionAsync(10000);
                    LabelGeolocation.Text = "Lat: " + results.Latitude + " Long: " +
results.Longitude;
                }
```

```
                else if (status != PermissionStatus.Unknown)
                {
                    await DisplayAlert("Location Denied", "Can not continue, try again.",
"OK");
                }
            }
            catch (Exception ex)
            {

                LabelGeolocation.Text = "Error: " + ex;
            }

            ((Button)sender).IsEnabled = true;
            busy = false;
        }
```

Lire Xamarin Plugin en ligne: https://riptutorial.com/fr/xamarin-forms/topic/7017/xamarin-plugin

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec Xamarin.Forms	Akshay Kulkarni, chrisntr, Community, Demitrian, hankide, jdstaerk, Manohar, patridge, Sergey Metlov, spaceplane
2	Accès aux fonctionnalités natives avec DependencyService	Gerald Versluis, hankide, hvaughan3, Sergey Metlov
3	Ajustements visuels spécifiques à la plate-forme	Alois, GalaxiaGuy, Paul
4	Alerte d'affichage	aboozz pallikara, GvSharma, Sreeraj, Yehor Hromadskyi
5	AppSettings Reader dans Xamarin.Forms	Ben Ishiyama-Levy, GvSharma
6	Base de données SQL et API dans les formulaires Xamarin.	RIYAZ
7	CarouselView - Version préliminaire	dpserge
8	Cellules Xamarin.Forms	Eng Soon Cheah
9	Comportement spécifique à la plate-forme	Ege Aydın
10	Contact Picker - Formulaires Xamarin (Android et iOS)	Pucho Eric
11	Création de contrôles personnalisés	hvaughan3, spaceplane, Yehor Hromadskyi
12	Cycle de vie de l'application Xamarin.Forms	Zverev Eugene

	générique? Plate-forme dépendante!	
13	Déclencheurs et comportements	hamalaiv, hvaughan3
14	DependencyService	Steven Thewissen
15	Effets	Swaminathan Vetri
16	Geste Xamarin	Joehl
17	Gestes	doerig, Gerald Versluis, Michael Rumpler
18	Gestion des exceptions	Yehor Hromadskyi
19	Liaison de données	Andrew, Matthew, Yehor Hromadskyi
20	MessagingCenter	Gerald Versluis
21	Mise en cache	Sergey Metlov
22	Mise en forme de Xamarin	Eng Soon Cheah, Gerald Versluis, Lucas Moura Veloso
23	Mise en page relative au Xamarin	Ege Aydın
24	Navigation dans Xamarin.Forms	Fernando Arreguín, jimmgarr, Lucas Moura Veloso, Paul, Sergey Metlov, Taras Shevchuk, Willian D. Andrade
25	Notifications push	Gerald Versluis, user1568891
26	OAuth2	Eng Soon Cheah
27	Page Xamarin.Forms	Eng Soon Cheah
28	Polices personnalisées dans les styles	Roma Rudyak
29	Pourquoi utiliser les formulaires Xamarin et quand utiliser les formulaires Xamarin	Daniel Krzyczkowski, mike
30	Renderers personnalisés	Bonelol, hankide, Nicolas Bodin-Ripert, Nicolas Bodin-Ripert, nishantvodoo, Yehor Hromadskyi, Zverev Eugene
31	Services de	RIYAZ

	dépendance	
32	Test d'unité	jerone, Sergey Metlov
33	Test d'unité BDD dans Xamarin.Forms	Ben Ishiyama-Levy
34	Travailler avec des bases de données locales	Luis Beltran, Manohar
35	Travailler avec des cartes	Taras Shevchuk
36	Utilisation de ListViews	cvanbeek
37	Vues Xamarin.Forms	Aaron Thompson, Eng Soon Cheah
38	Xamarin Plugin	Eng Soon Cheah