EBook Gratuito

APPRENDIMENTO Xamarin.Forms

Free unaffiliated eBook created from
Stack Overflow contributors.

#xamarin.forms

Sommario

Di

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

Capitolo 1: Iniziare con Xamarin.Forms

Osservazioni

Xamarin.Forms consente di creare app iOS, Android e Windows con grandi quantità di codice condiviso, incluso il codice dell'interfaccia utente o il markup dell'interfaccia utente XAML. Le pagine e le viste delle app sono associate ai controlli nativi su ciascuna piattaforma, ma possono essere personalizzate per fornire un'interfaccia utente specifica della piattaforma o per accedere a funzionalità specifiche della piattaforma.

Versioni

Versione	Data di rilascio
2.3.1	2016/08/03
2.3.0-Hotfix1	2016/06/29
2.3.0	2016/06/16
2.2.0-Hotfix1	2016/05/30
2.2.0	2016/04/27
2.1.0	2016/03/13
2.0.1	2016/01/20
2.0.0	2015/11/17
1.5.1	2016/10/20
1.5.0	2016/09/25
1.4.4	2015/07/27
1.4.3	2015/06/30
1.4.2	2015/04/21
1.4.1	2015/03/30
1.4.0	2015/03/09
1.3.5	2015/03/02
1.3.4	2015/02/17

Versione	Data di rilascio
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

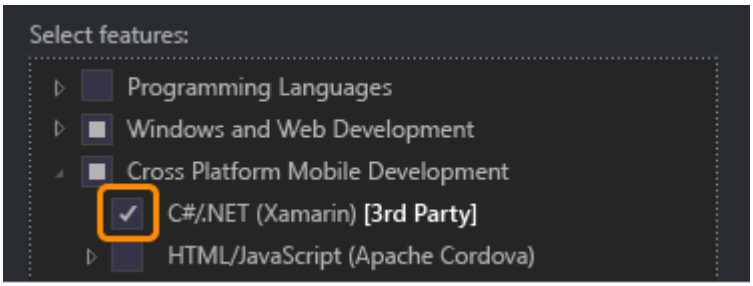
Installazione (Visual Studio)

Xamarin.Forms è un'astrazione dell'interfaccia utente dell'interfaccia utente nativamente supportata da più piattaforme che consente agli sviluppatori di creare facilmente interfacce utente che possono essere condivise su Android, iOS, Windows e Windows Phone. Le interfacce utente sono renderizzate utilizzando i controlli nativi della piattaforma di destinazione, consentendo alle applicazioni Xamarin.Form di mantenere l'aspetto e il feel appropriati per ciascuna piattaforma.

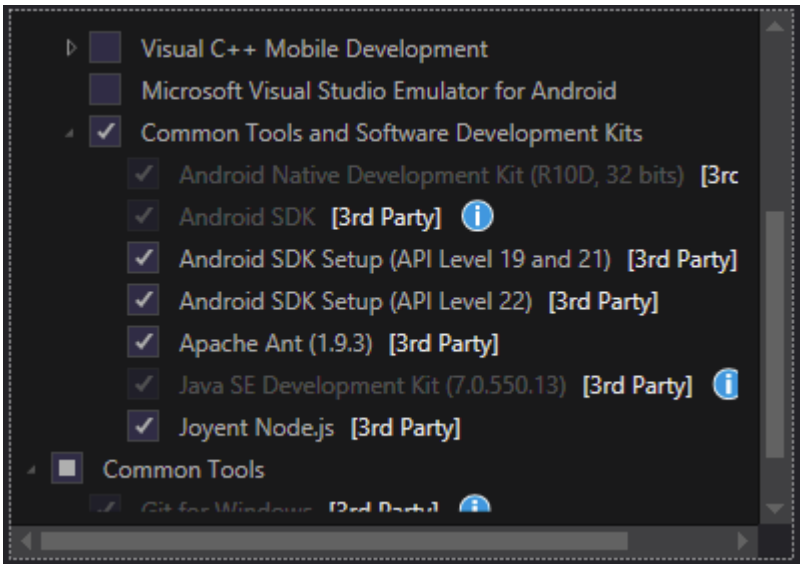
Xamarin Plugin per Visual Studio

Per iniziare con Xamarin.Forms per Visual Studio è necessario avere il plugin Xamarin stesso. Il modo più semplice per installarlo è scaricare e installare l'ultima versione di Visual Studio.

Se hai già installato Visual Studio più recente, vai a Pannello di controllo> Programmi e funzionalità, fai clic con il tasto destro su Visual Studio e fai clic su Cambia. Quando si apre il programma di installazione, fai clic su Modifica e seleziona gli strumenti di sviluppo mobile multipiattaforma:



Puoi anche scegliere di installare l'SDK di Android:



Deselezionalo se hai già installato l'SDK. Sarai in grado di configurare Xamarin per utilizzare l'SDK Android esistente in seguito.

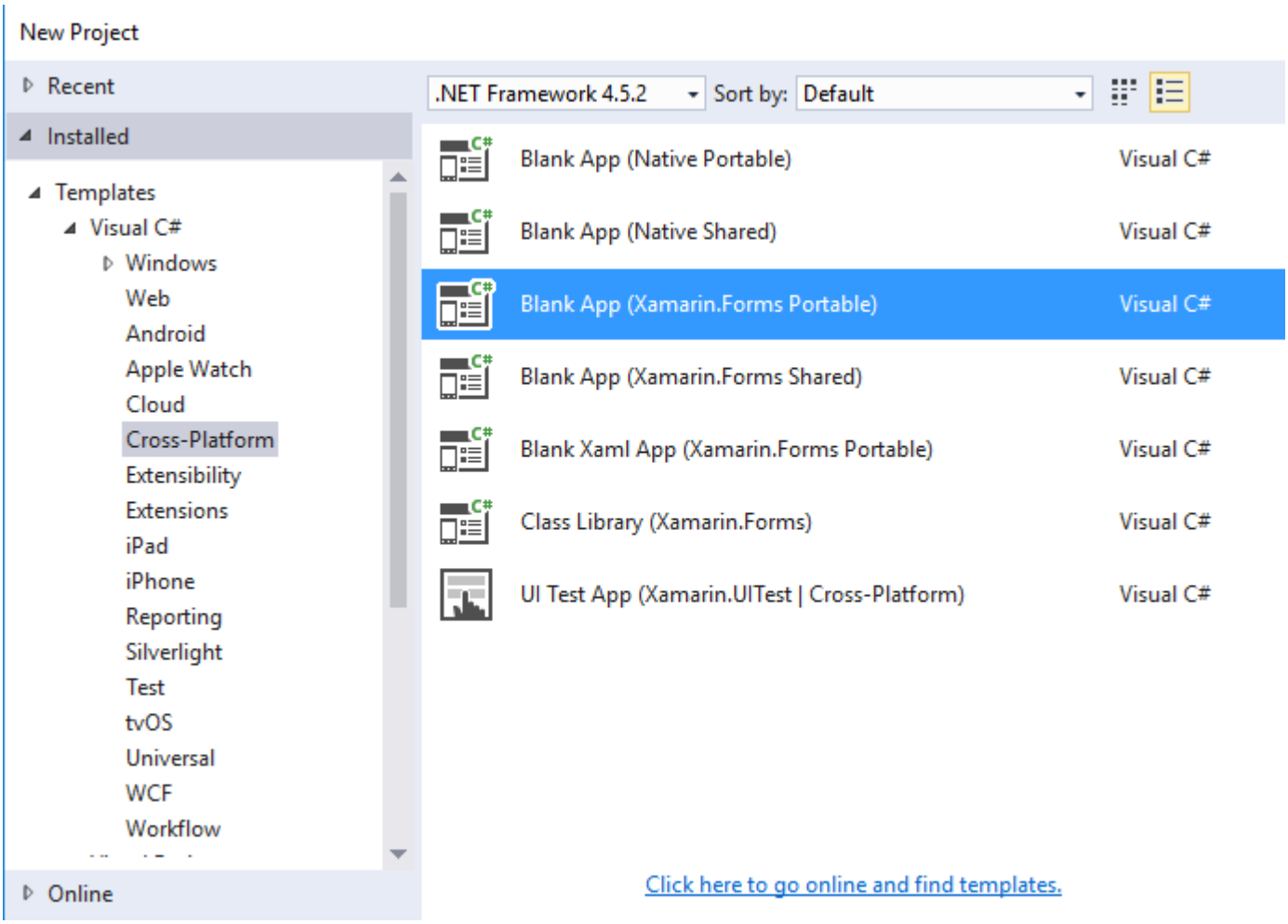
Xamarin.Forms

Xamarin.Forms è un set di librerie per la libreria di classi portatili e gli assembly nativi. La stessa libreria Xamarin.Forms è disponibile come pacchetto NuGet. Per aggiungerlo al tuo progetto basta usare il normale comando `Install-Package` della Console di Gestione pacchetti:

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

per tutti gli assembly iniziali (ad esempio MyProject, MyProject.Droid e MyProject.iOS).

Il modo più semplice per iniziare con Xamarin.Forms è creare un progetto vuoto in Visual Studio:



Come puoi vedere ci sono 2 opzioni disponibili per creare l'app vuota - Portatile e condiviso. Vi consiglio di iniziare con Portable one perché è il più usato nel mondo reale (differenze e ulteriori spiegazioni da aggiungere).

Dopo aver creato il progetto, assicurati di utilizzare l'ultima versione di Xamarin.Forms poiché il modello iniziale potrebbe contenere quello vecchio. Usa la tua Console Gestione pacchetti o Gestisci l'opzione Pacchetti NuGet per eseguire l'aggiornamento agli ultimi Xamarin.Forms (ricorda che è solo un pacchetto NuGet).

Mentre i modelli di Visual Studio Xamarin.Forms creeranno per te un progetto di piattaforma iOS, dovrai connettere Xamarin a un host di build Mac per poter eseguire questi progetti su iOS Simulator o dispositivi fisici.

Hello World Xamarin Forms: Visual Studio

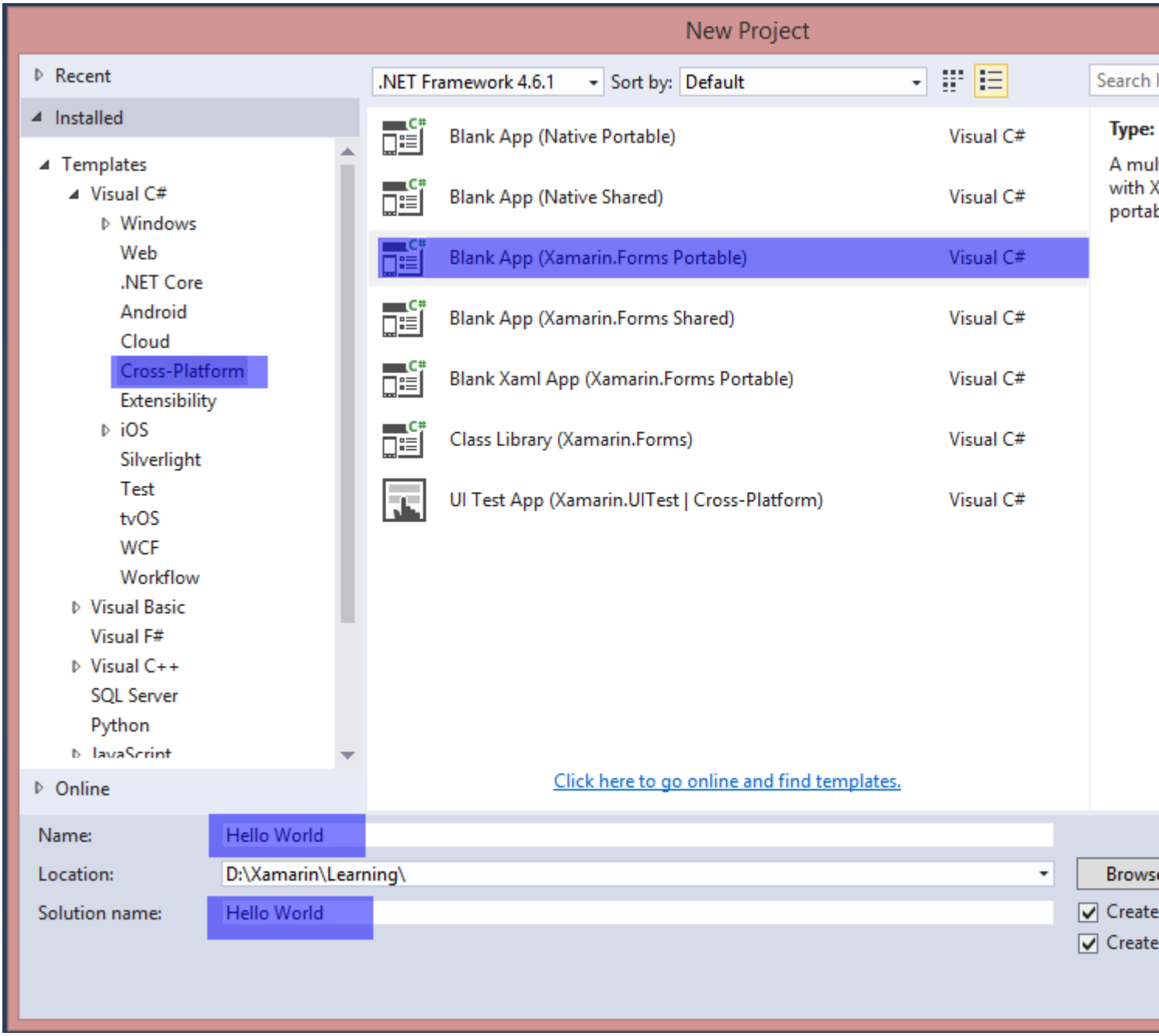
Dopo aver installato Xamarin come descritto nel primo esempio, è ora di lanciare la prima applicazione di esempio.

Passaggio 1: creazione di un nuovo progetto.

In Visual Studio, scegli Nuovo -> Progetto -> Visual C # -> Cross-Platform -> Blank App (Xamarin.Forms Portable)

Denominare l'app "Hello World" e selezionare la posizione per creare il progetto e fare clic su OK. Questo creerà una soluzione per te che contiene tre progetti:

1. HelloWorld (qui è posizionata la logica e le viste, ovvero il progetto portatile)
2. HelloWorld.Droid (il progetto Android)
3. HelloWorld.iOS (il progetto iOS)



Passaggio 2: analisi del campione

Dopo aver creato la soluzione, un'applicazione di esempio sarà pronta per essere distribuita. Apri `App.cs` trova nella radice del progetto portatile e investiga il codice. Come visto sotto, il `Content` del campione è uno `StackLayout` che contiene `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
        }
    }
}
```

Passaggio 3: avvio dell'applicazione

Ora fai semplicemente clic con il pulsante destro del mouse sul progetto che desideri avviare (`HelloWorld.Droid` o `HelloWorld.iOS`) e fai clic su `Set as StartUp Project` di avvio. Quindi, nella barra degli strumenti di Visual Studio, fare clic sul pulsante `Start` (il pulsante triangolare verde che assomiglia a un pulsante Riproduci) per avviare l'applicazione sul simulatore / emulatore di destinazione.

Leggi Iniziare con Xamarin.Forms online: https://riptutorial.com/it/xamarin-forms/topic/908/iniziare-con-xamarin-forms

Capitolo 2: Accesso alle funzionalità native con DependencyService

Osservazioni

Se non vuoi che il tuo codice si interrompa quando non viene trovata alcuna implementazione, controlla prima `DependencyService` se ha un'implementazione disponibile.

Puoi farlo con un semplice controllo se non è `null` .

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

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

oppure, se il tuo IDE supporta C # 6, con operatore null-condizionale:

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

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

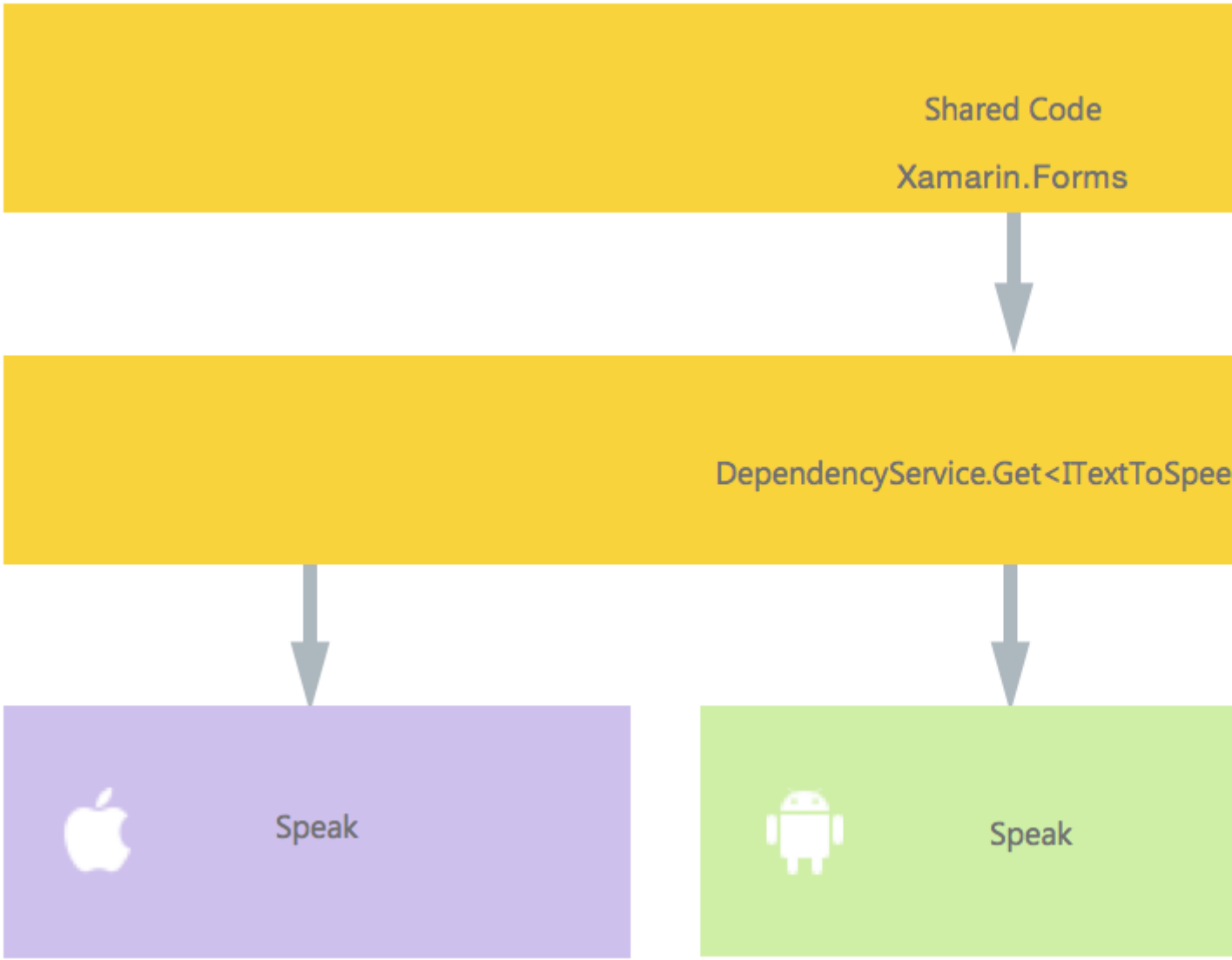
Se non lo fai e non viene trovata alcuna implementazione in fase di runtime, il tuo codice genererà un'eccezione.

Examples

Implementazione del text-to-speech

Un buon esempio di una funzione che richiede il codice specifico della piattaforma è quando si desidera implementare la sintesi vocale (tts). Questo esempio presuppone che si stia lavorando con codice condiviso in una libreria PCL.

Una panoramica schematica della nostra soluzione apparirebbe come l'immagine sottostante.



Nel nostro codice condiviso definiamo un'interfaccia che è registrata con `DependencyService` . Qui è dove faremo le nostre chiamate. Definisci un'interfaccia come sotto.

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

Ora in ogni piattaforma specifica, abbiamo bisogno di creare un'implementazione di questa interfaccia. Iniziamo con l'implementazione iOS.

Implementazione 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);
    }
}
```

Nell'esempio di codice sopra riportato si nota che esiste un codice specifico per iOS. Ti piace tipi come `AVSpeechSynthesizer` . Questi non funzionerebbero nel codice condiviso.

Per registrare questa implementazione con Xamarin `DependencyService` aggiungere questo attributo sopra la dichiarazione dello spazio dei nomi.

```
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
```

Ora quando fai una chiamata come questa nel tuo codice condiviso, viene iniettata l'implementazione corretta per la piattaforma su cui stai eseguendo la tua app.

`DependencyService.Get<ITextToSpeech>()` . Maggiori informazioni su questo in seguito.

Implementazione Android

L'implementazione di Android di questo codice potrebbe apparire come sotto.

```
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
}
```

Ancora una volta non dimenticare di registrarlo con `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
```

Implementazione di Windows Phone

Infine, per Windows Phone è possibile utilizzare questo codice.

```
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);
    }
}
```

E ancora una volta non dimenticare di registrarlo.

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

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

Implementazione in codice condiviso

Ora è tutto a posto per farlo funzionare! Infine, nel tuo codice condiviso ora puoi chiamare questa funzione usando l'interfaccia. In fase di esecuzione verrà iniettata l'implementazione che corrisponde alla piattaforma corrente su cui è in esecuzione.

In questo codice vedrai una pagina che potrebbe essere in un progetto Xamarin Forms. Crea un pulsante che richiama il metodo `Speak()` utilizzando `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;
}
```

Il risultato sarà che quando l'app viene eseguita e si fa clic sul pulsante, verrà pronunciato il testo fornito.

Tutto questo senza dover fare cose difficili come suggerimenti per il compilatore e così via. Ora hai un modo uniforme per accedere alle funzionalità specifiche della piattaforma attraverso un codice indipendente dalla piattaforma.

Ottenere numeri di versione del sistema operativo dell'applicazione e del dispositivo - Android e iOS - PCL

L'esempio seguente raccoglierà il numero di versione del sistema operativo del dispositivo e la versione dell'applicazione (definita nelle proprietà di ciascun progetto) immessa nel **nome della versione** su Android e **versione** su iOS.

Per prima cosa crea un'interfaccia nel tuo progetto 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();
}
```

Ora implementiamo l'interfaccia nei progetti Android e 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; }
    }
```

```
}
```

Ora per usare il codice in un metodo:

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

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

Leggi Accesso alle funzionalità native con DependencyService online: https://riptutorial.com/it/xamarin-forms/topic/2409/accesso-alle-funzionalita-native-con-dependencyservice

Capitolo 3: AppSettings Reader in Xamarin.Forms

Examples

Lettura del file app.config in un progetto Xaml Xamarin.Forms

Mentre ogni piattaforma mobile offre la propria API di gestione delle impostazioni, non ci sono modi per leggere le impostazioni da un buon vecchio file .net di stile app.config xml; Ciò è dovuto a una serie di buone ragioni, in particolare la gestione della rete di configurazione. NET che si trova sul lato dei pesi massimi, e ciascuna piattaforma ha una propria API di file system.

Così abbiamo creato una semplice libreria PCLAppConfig , ben nuget impacchettata per il tuo consumo immediato.

Questa libreria utilizza la deliziosa libreria PCLStorage

In questo esempio si presuppone che si stia sviluppando un progetto Xaml Xamarin.Forms, in cui sarà necessario accedere alle impostazioni dal proprio viewmodel condiviso.

1. Inizializza ConfigurationManager.AppSettings su ciascun progetto della piattaforma, subito dopo la dichiarazione 'Xamarin.Forms.Forms.Init', come di seguito:

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. Aggiungi un file app.config al tuo progetto PCL condiviso e aggiungi le voci appSettings, come faresti con qualsiasi file app.config

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

```
</configuration>
```

3. Aggiungi questo file app.config PCL **come file collegato** su tutti i tuoi progetti di piattaforma. Per Android, assicurati di impostare l'azione di compilazione su **"AndroidAsset"** , per UWP imposta l'azione di creazione su **"Contenuto"**

4. Accedi alle tue impostazioni: `ConfigurationManager.AppSettings["config.text"];`

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

Capitolo 4: Associazione dati

Osservazioni

Possibili eccezioni

System.ArrayTypeMismatchException: Tentativo di accedere a un elemento come tipo incompatibile con l'array.

Questa eccezione può verificarsi quando si tenta di associare una raccolta a una proprietà non associabile quando la pre-compilazione XAML è abilitata. Un esempio comune è il tentativo di associare `Picker.Items` . Vedi sotto.

System.ArgumentException: oggetto di tipo 'Xamarin.Forms.Binding' non può essere convertito in tipo 'System.String'.

Questa eccezione può verificarsi quando si tenta di associare una raccolta a una proprietà non associabile quando la pre-compilazione XAML è disabilitata. Un esempio comune è il tentativo di associare `Picker.Items` . Vedi sotto.

Il `Picker.Items` proprietà non è Bindable

Questo codice causerà un errore:

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

L'eccezione può essere una delle seguenti:

> System.ArrayTypeMismatchException: Tentativo di accedere a un elemento come tipo incompatibile con l'array.

o

> System.ArgumentException: oggetto di tipo 'Xamarin.Forms.Binding' non può essere convertito in tipo 'System.String'.

In particolare, la proprietà `Items` non è associabile. Le soluzioni includono la creazione di un controllo personalizzato o l'utilizzo di un controllo di terze parti, come `BindablePicker` di FreshEssentials . Dopo aver installato il pacchetto FreshEssentials NuGet nel progetto, del

pacchetto di `BindablePicker` controllo con un associabile `ItemsSource` proprietà è disponibile:

```
<?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

Collegamento di base a ViewModel

EntryPage.xaml:

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

MyViewModel.cs:

```
using System;
using System.ComponentModel;

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

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

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

        public ICommand SaveCommand { get; private set; }

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

        private void SaveCommandExecute()
        {

        }

        public event PropertyChangedEventHandler PropertyChanged;

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

Leggi Associazione dati online: https://riptutorial.com/it/xamarin-forms/topic/3915/associazione-dati

Capitolo 5: Avviso display

Examples

DisplayAlert

Una finestra di avviso può essere visualizzata su una `Xamarin.Forms Page` con il metodo `DisplayAlert` . Possiamo fornire un titolo, un corpo (testo da avvisare) e uno / due pulsanti di azione. `Page` offre due sostituzioni del metodo `DisplayAlert` .

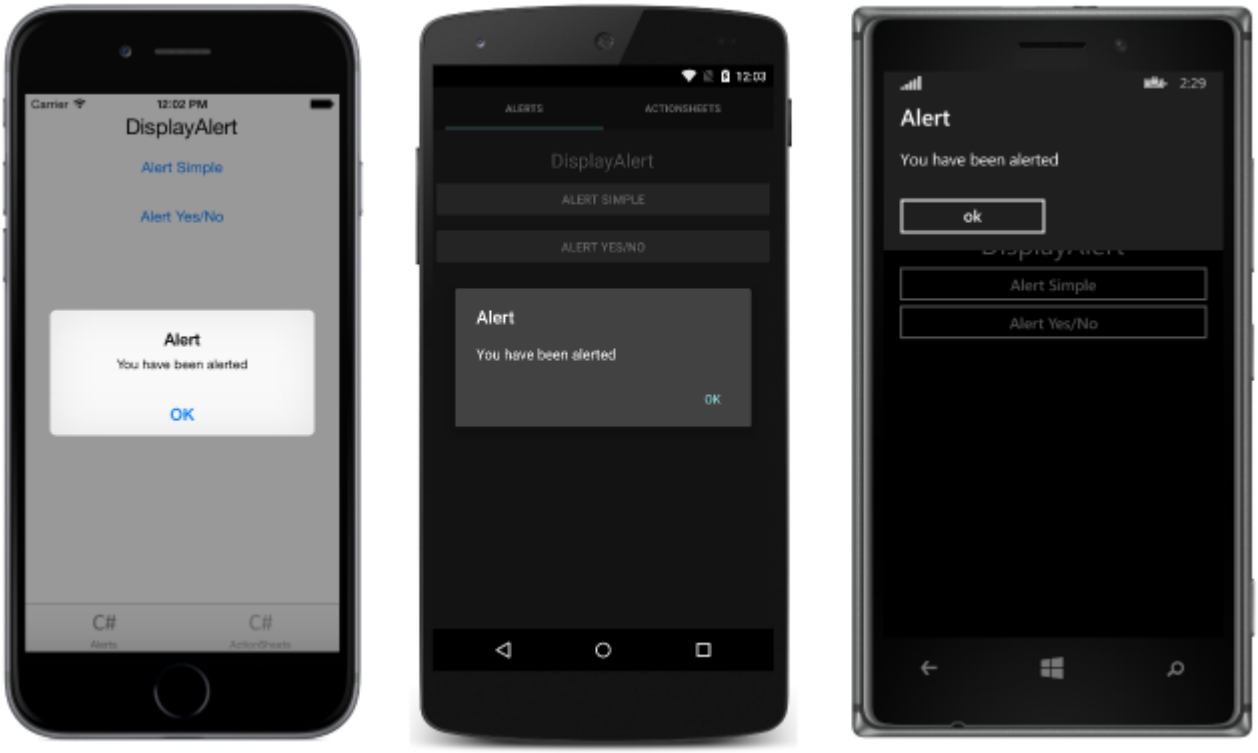
1. `public Task DisplayAlert (String title, String message, String cancel)`

Questa sovrascrittura presenta una finestra di avviso per l'utente dell'applicazione con un singolo pulsante di annullamento. L'avviso viene visualizzato in modo modale e, una volta chiuso, l'utente continua a interagire con l'applicazione.

Esempio :

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

Sopra snippet presenterà un'implementazione nativa di avvisi in ogni piattaforma (`AlertDialog` in Android, `UIAlertView` in iOS, `MessageDialog` in Windows) come di seguito.



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

Questa sostituzione esegue una finestra di avviso per l'utente dell'applicazione con un pulsante Accetta e Annulla. Cattura la risposta di un utente presentando due pulsanti e restituendo un

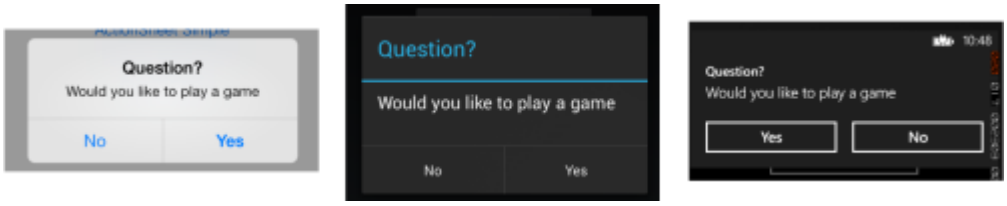
valore `boolean` . Per ottenere una risposta da un avviso, fornire il testo per entrambi i pulsanti e attendere il metodo. Dopo che l'utente ha selezionato una delle opzioni, la risposta verrà restituita al codice.

Esempio :

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

Esempio 2: (se Condizione true o false check to alert procedere)

```
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
        }
     }
```



Esempio di avviso con un solo pulsante e azione

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

Qui otterremo l'azione clic Ok.

Leggi Avviso display online: https://riptutorial.com/it/xamarin-forms/topic/4883/avviso-display

Capitolo 6: caching

Examples

Memorizzazione nella cache con Akavache

Informazioni su Akavache

Akavache è una libreria incredibilmente utile che fornisce funzionalità di copertura per la memorizzazione nella cache dei dati. Akavache fornisce un'interfaccia di archiviazione a valore chiave e funziona sulla parte superiore di SQLite3. Non è necessario mantenere sincronizzati gli schemi poiché è in realtà una soluzione No-SQL che lo rende perfetto per la maggior parte delle applicazioni mobili, specialmente se è necessario aggiornare l'app spesso senza perdita di dati.

Raccomandazioni per Xamarin

Akavache è sicuramente la migliore libreria di caching per l'applicazione Xamarin se solo tu non hai bisogno di operare con dati fortemente relativi, quantità binarie o veramente grandi di dati. Utilizzare Akavache nei seguenti casi:

- È necessario che l'app memorizzi nella cache i dati per un determinato periodo di tempo (è possibile configurare il timeout di scadenza per ciascuna entità che viene salvata;
- Vuoi che la tua app funzioni offline;
- È difficile determinare e congelare lo schema dei dati. Ad esempio, hai elenchi contenenti diversi oggetti tipizzati;
- È sufficiente avere un accesso semplice ai valori-chiave per i dati e non è necessario creare query complesse.

Akavache non è un "punto d'argento" per l'archiviazione dei dati, quindi pensaci due volte a usarlo nei seguenti casi:

- Le tue entità di dati hanno molte relazioni tra loro;
- Non hai davvero bisogno che la tua app funzioni offline;
- Hai un'enorme quantità di dati da salvare localmente;
- È necessario migrare i dati da una versione all'altra;
- È necessario eseguire query complesse tipiche per il raggruppamento, le proiezioni, ecc. Di SQL.

  In realtà è possibile migrare manualmente i dati semplicemente leggendo e scrivendo di nuovo con i campi aggiornati.

Semplice esempio

Interagire con Akavache avviene principalmente attraverso un oggetto chiamato `BlobCache` .

La maggior parte dei metodi di Akavache restituisce osservabili reattivi, ma puoi anche solo aspettarli grazie ai metodi di estensione.

```
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!"));
```

Gestione degli errori

```
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()));
```

Leggi caching online: https://riptutorial.com/it/xamarin-forms/topic/3644/caching

Capitolo 7: Caratteri personalizzati negli stili

Osservazioni

Risorse da guardare:

- Stili di Xamarin
- Utilizzo di caratteri personalizzati su iOS e Android con Xamarin.Forms
- Renderizzatori personalizzati
- Dizionari delle risorse
- Proprietà allegate

Examples

Accesso a caratteri personalizzati in Syles

Xamarin.Forms fornisce un ottimo meccanismo per lo styling dell'applicazione cross-platform con stili globali.

Nel mondo mobile la tua applicazione deve essere carina e distinguersi dalle altre applicazioni. Uno di questi caratteri è Caratteri personalizzati utilizzati nell'applicazione.

Con il potente supporto di XAML Styling in Xamarin.Forms hai appena creato lo stile di base per tutte le etichette con i tuoi font personalizzati.

Per includere caratteri personalizzati nei tuoi progetti iOS e Android segui la guida in Uso dei caratteri personalizzati su iOS e Android con Xamarin.Form post scritto da Gerald.

Dichiarare lo stile nella sezione delle risorse del file App.xaml. Questo rende tutti gli stili visibili a livello globale.

Da Gerald post sopra dobbiamo usare la proprietà StyleId ma non è una proprietà associabile, quindi per utilizzarla in Style Setter dobbiamo creare la proprietà Attachable per esso:

```
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);
        }
    }
}
```

Quindi aggiungi lo stile nella risorsa 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>
```

In base al post precedente, dobbiamo creare un Renderer personalizzato per Label che erediti dalla piattaforma Android di LabelRenderer.

```
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;
        }
    }
}
```

Per la piattaforma iOS non sono richiesti riproduttori personalizzati.

Ora puoi ottenere lo stile nel markup della pagina:

Per etichetta specifica

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

Oppure applica lo stile a tutte le etichette sulla pagina creando Stile basato su 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>
```

Leggi Caratteri personalizzati negli stili online: https://riptutorial.com/it/xamarin-forms/topic/4854/caratteri-personalizzati-negli-stili

Capitolo 8: CarouselView - Versione pre-release

Osservazioni

CarouselView è un controllo Xamarin che può contenere qualsiasi tipo di vista. Questo controllo preliminare può essere utilizzato solo nei progetti Xamarin Forms.

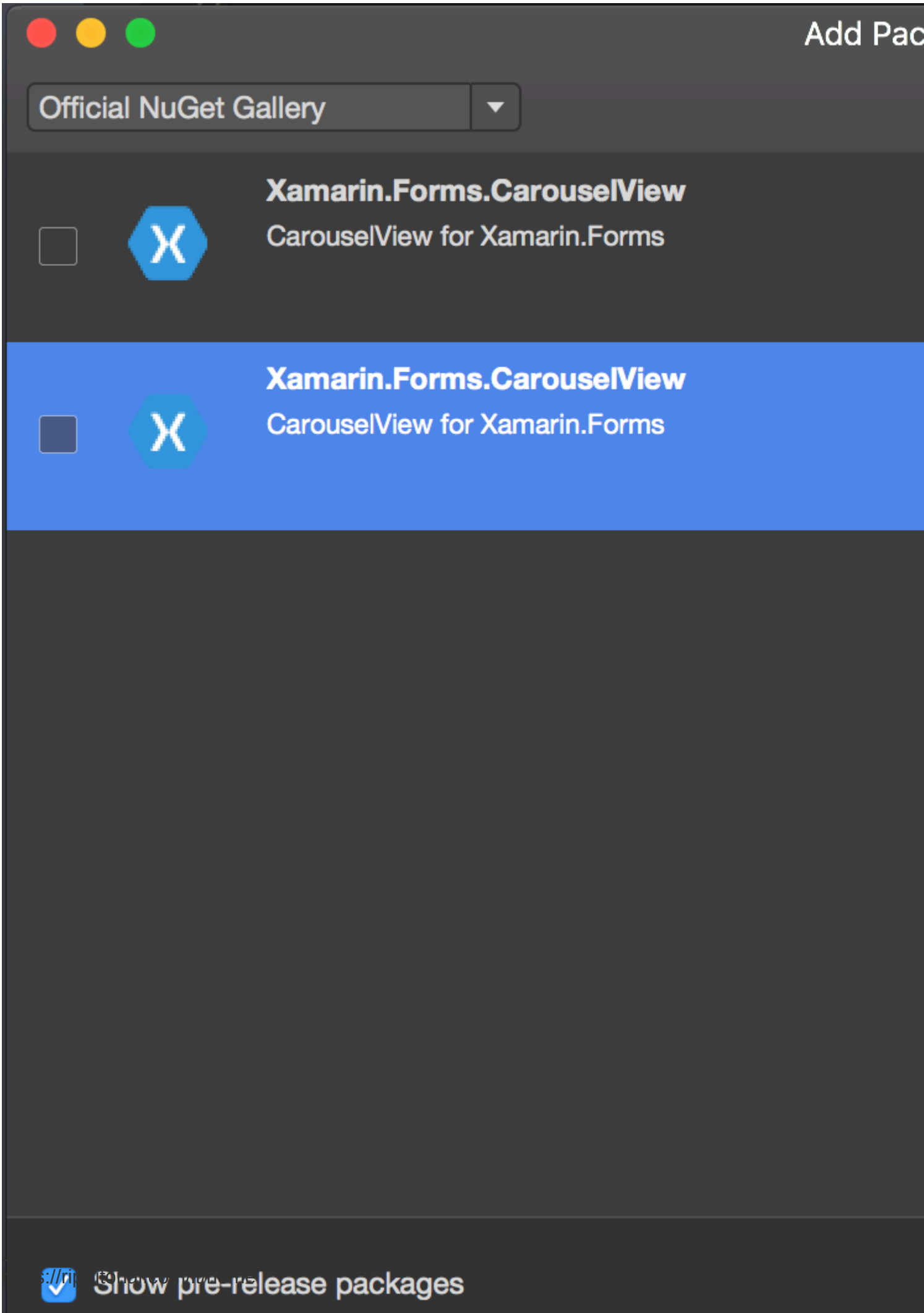
Nell'esempio fornito da James Montemagno , sul blog di Xamarin, CarouselView viene utilizzato per visualizzare le immagini.

In questo momento CarouselView non è integrato in Xamarin.Forms. Per utilizzarlo nei tuoi progetti, dovrai aggiungere il pacchetto NuGet (vedi esempio sopra).

Examples

Importa CarouselView

Il modo più semplice per importare CarouselView è utilizzare il Gestore di pacchetti NuGet in Xamarin / Visual Studio:


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

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

Capitolo 9: Cellule Xamarin.Forms

Examples

EntryCell

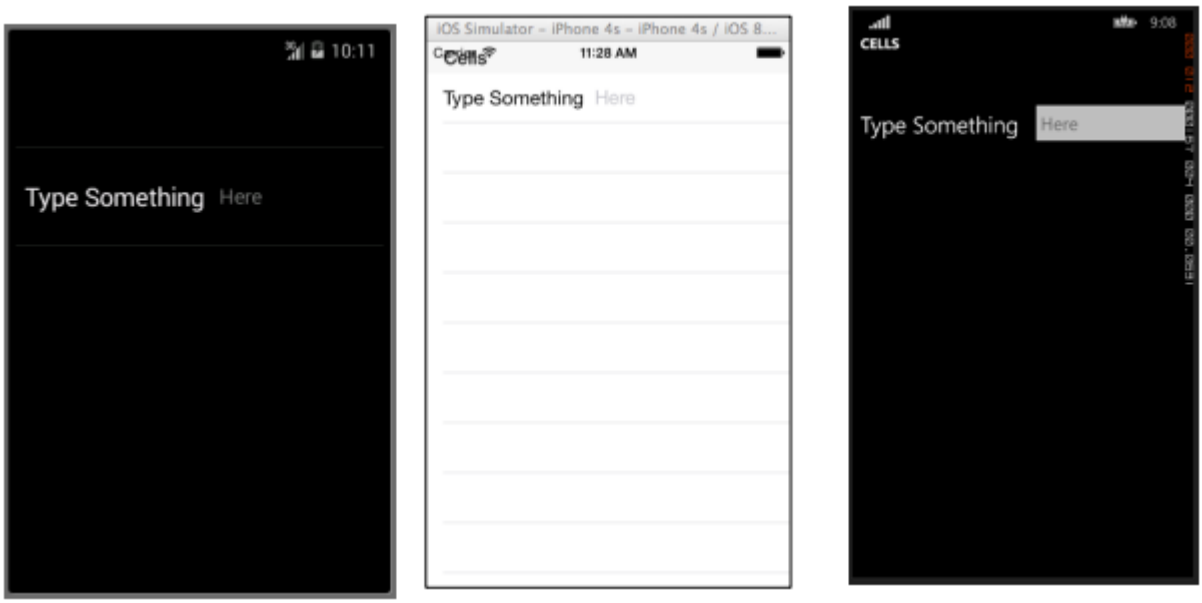
Una EntryCell è una cella che combina le capacità di un'etichetta e di una voce. EntryCell può essere utile in scenari quando si creano alcune funzionalità all'interno dell'applicazione per raccogliere dati dall'utente. Possono essere facilmente inseriti in un TableView e trattati come un semplice modulo.

XAML

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

Codice

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



SwitchCell

Una SwitchCell è una cella che combina le capacità di un'etichetta e di un interruttore on-off. Una SwitchCell può essere utile per attivare o disattivare la funzionalità o anche le preferenze dell'utente o le opzioni di configurazione.

XAML

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

Codice

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

TextCell

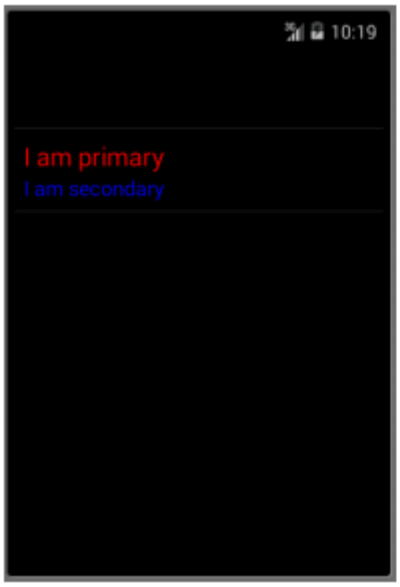
Un TextCell è una cella che ha due aree di testo separate per la visualizzazione dei dati. Un TextCell viene in genere utilizzato a scopo informativo in entrambi i controlli TableView e ListView. Le due aree di testo sono allineate verticalmente per massimizzare lo spazio all'interno della cella. Questo tipo di cella viene anche comunemente utilizzato per visualizzare dati gerarchici, quindi quando l'utente tocca questa cella, passerà a un'altra pagina.

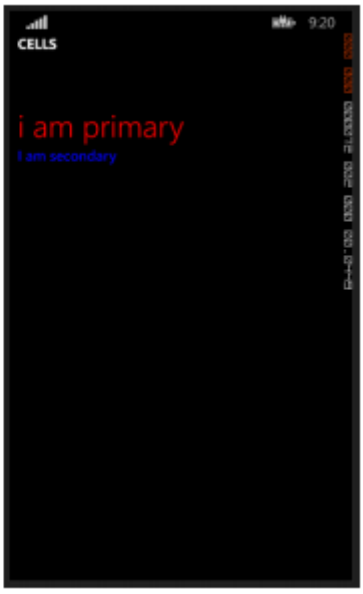
XAML

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

Codice

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





ImageCell

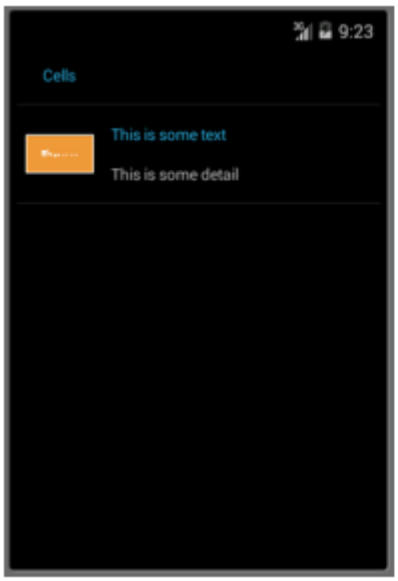
Un ImageCell è esattamente come sembra. È una semplice cella che contiene solo un'immagine. Questo controllo funziona in modo molto simile a un normale controllo Immagine, ma con un numero di campanelli e fischietti molto inferiore.

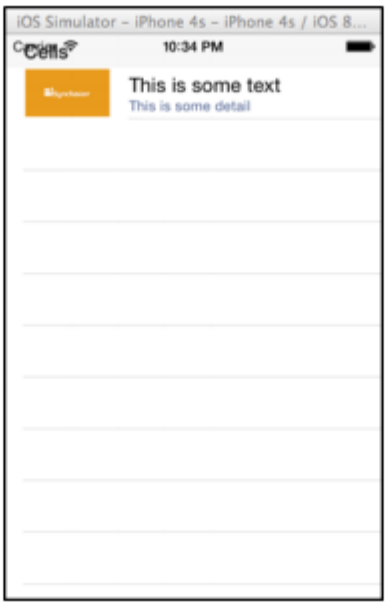
XAML

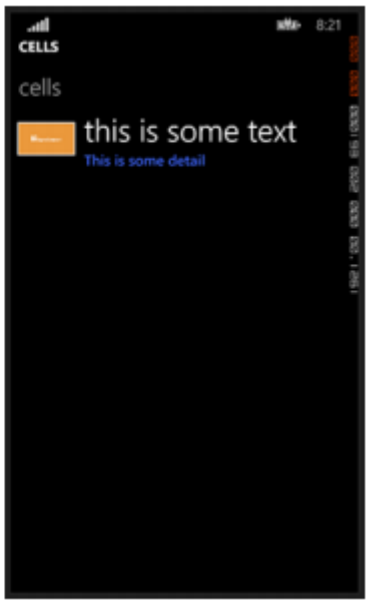
```
<ImageCell ImageSource="http://d2g29cya9iq7ip.cloudfront.net/content/imag
es/company/aboutus-video-bg.png?v=25072014072745")),
 Text="This is some text"
 Detail="This is some detail" />
```

Codice

```
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

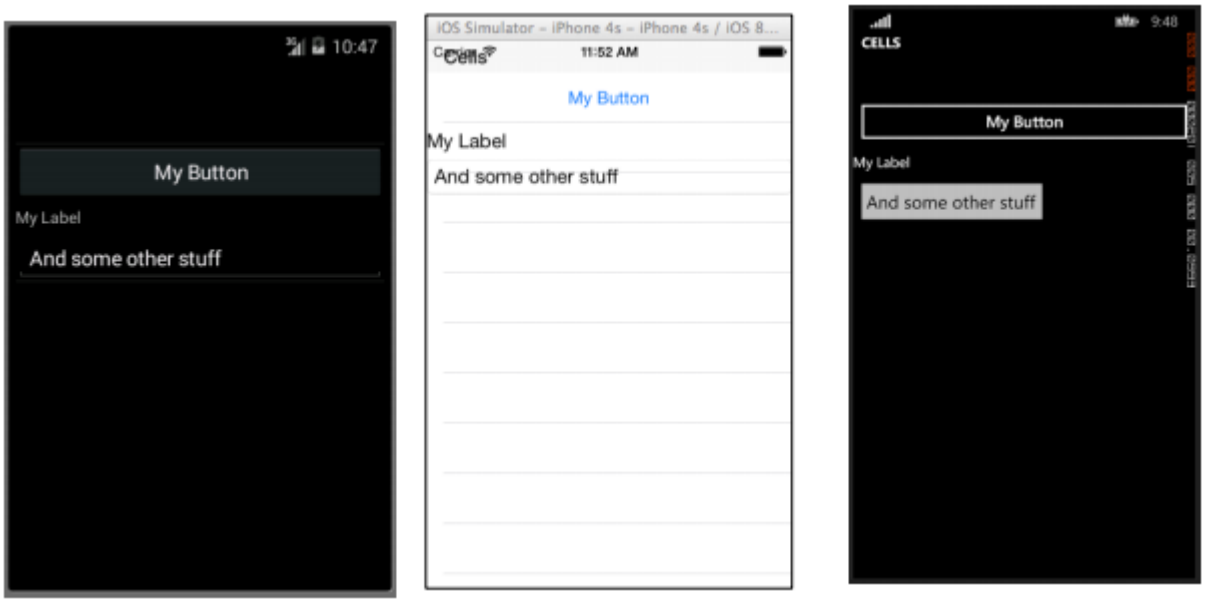
Puoi considerare un ViewCell una lavagna vuota. È la tua tela personale per creare una cella che sembra esattamente come la vuoi tu. Puoi anche comporlo di istanze di più altri oggetti View messi insieme con i controlli Layout. Sei solo limitato dalla tua immaginazione. E forse la dimensione dello schermo.

XAML

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

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

Codice

```
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 }
}
};
```



Leggi Cellule Xamarin.Forms online: https://riptutorial.com/it/xamarin-forms/topic/7370/cellule-xamarin-forms

Capitolo 10: Ciclo di vita generico dell'app Xamarin.Forms? Platform-dipendente!

Examples

Il ciclo di vita di Xamarin.Forms non è l'effettivo ciclo di vita dell'app ma una rappresentazione multipiattaforma di esso.

Diamo un'occhiata ai metodi nativi per il ciclo di vita delle app per diverse piattaforme.

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);
```

Finestre.

```
//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;
```

E ora i metodi del ciclo di vita delle app **Xamarin.Forms** :

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

Ciò che si può facilmente capire dal semplice osservare gli elenchi, la prospettiva del ciclo di vita delle app multipiattaforma di Xamarin.Forms è notevolmente semplificata. Ti dà l'indizio generico su quale sia lo stato della tua app, ma nella maggior parte dei casi di produzione dovrai costruire una logica dipendente dalla piattaforma.

Leggi Ciclo di vita generico dell'app Xamarin.Forms? Platform-dipendente! online: https://riptutorial.com/it/xamarin-forms/topic/8329/ciclo-di-vita-generico-dell-app-xamarin-forms--platform-dipendente-

Capitolo 11: Comportamento specifico della piattaforma

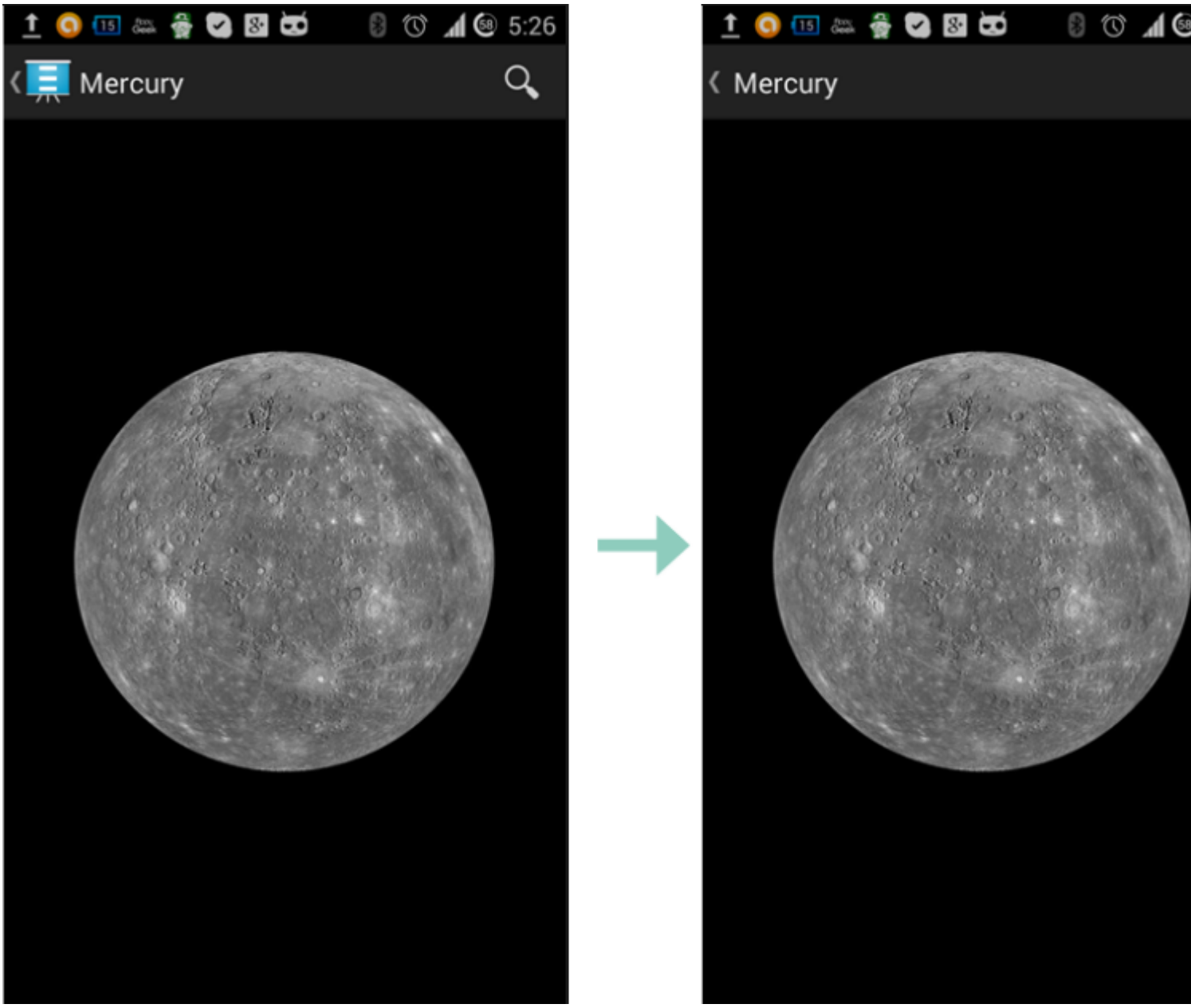
Osservazioni

Piattaforme di destinazione

```
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

Rimozione dell'icona nell'intestazione di navigazione in Anroid



Usando una piccola immagine trasparente chiamata empty.png

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

Riduci le dimensioni del carattere dell'etichetta in iOS

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

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

Leggi Comportamento specifico della piattaforma online: https://riptutorial.com/it/xamarin-forms/topic/6636/comportamento-specifico-della-piattaforma

Capitolo 12: Creazione di controlli personalizzati

Examples

Creare un controllo di input personalizzato di Xamarin Forms (non è richiesto alcun nativo)

Di seguito è riportato un esempio di controllo personalizzato di Xamarin Forms puro. Per questo non viene eseguito alcun rendering personalizzato, ma potrebbe essere facilmente implementato, infatti, nel mio codice personale, utilizzo questo stesso controllo insieme a un renderer personalizzato sia per `Label` che per `Entry` .

Il controllo personalizzato è un `ContentView` con `Label` , una `Entry` e un `BoxView` suo interno, tenuti in posizione utilizzando 2 `StackLayout` s. Definiamo anche più proprietà associabili e un evento `TextChanged` .

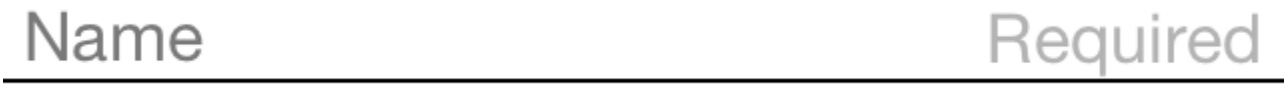
Le proprietà bindable personalizzate funzionano in base alla loro definizione e al fatto che gli elementi all'interno del controllo (in questo caso `Label` e una `Entry`) sono associati alle proprietà bindable personalizzate. Alcune proprietà `BindingPropertyChangedDelegate` devono anche implementare un `BindingPropertyChangedDelegate` per fare in modo che gli elementi delimitati modifichino i loro valori.

```
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;
    }
}
```

Ed ecco un'immagine del prodotto finale su iOS (l'immagine mostra come appare quando si usa un renderizzatore personalizzato per `Label` and `Entry` che viene usato per rimuovere il bordo su iOS e per specificare un font personalizzato per entrambi gli elementi):



Un problema che ho incontrato è stato ottenere `BoxView.BackgroundColor` per cambiare quando `UnderlineColor` cambiato. Anche dopo aver vincolato la proprietà `BackgroundColor` `BoxView` , non sarebbe cambiato fino a quando non ho aggiunto il delegato `UnderlineColorChanged` .

Etichetta con raccolta cumulabile di span

Ho creato un'etichetta personalizzata con wrapper attorno alla proprietà `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;
    }
}
```

Ho aggiunto la raccolta di Custom `TextComponent` :

```
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); }
    }
}
```

E quando la collezione di componenti di testo cambia o la proprietà `Text` di componenti separati

cambia, ricostruisco la proprietà `FormattedText Label` di base.

E come l'ho usato in `XAML` :

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

Codebehind of page:

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

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

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

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

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

Creazione di un controllo voce personalizzato con una proprietà MaxLength

Il controllo della `Entry` Form Xamarin non ha una proprietà `MaxLength` . Per ottenere ciò è possibile estendere `Entry` come sotto, aggiungendo una proprietà MaxLength `MaxLength` . Quindi devi solo iscriverti all'evento `TextChanged` su `Entry` e convalidare la lunghezza del `Text` quando viene

chiamato:

```
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;
    }
}
```

Utilizzo in XAML:

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

Leggi Creazione di controlli personalizzati online: https://riptutorial.com/it/xamarin-forms/topic/3913/creazione-di-controlli-personalizzati

Capitolo 13: Creazione di controlli personalizzati

introduzione

Ogni vista `Xamarin.Forms` ha un renderer di accompagnamento per ogni piattaforma che crea un'istanza di un controllo nativo. Quando una vista viene renderizzata sulla piattaforma specifica, la classe `ViewRenderer` viene istanziata.

Il processo per farlo è il seguente:

Creare un controllo personalizzato Xamarin.Forms.

Consuma il controllo personalizzato da Xamarin.Forms.

Creare il renderizzatore personalizzato per il controllo su ciascuna piattaforma.

Examples

Implementazione di un controllo CheckBox

In questo esempio implementeremo una casella di controllo personalizzata per Android e iOS.

Creazione del controllo personalizzato

```
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);
        }
    }
}
```

Iniziamo con il rendering personalizzato di Android creando una nuova classe (`CheckboxCustomRenderer`) nella parte `Android` della nostra soluzione.

Alcuni dettagli importanti da notare:

- Dobbiamo contrassegnare la parte superiore della nostra classe con l'attributo ExportRenderer in modo che il renderer sia registrato con `Xamarin.Forms` . In questo modo, `Xamarin.Forms` utilizzerà questo renderer quando sta tentando di creare il nostro oggetto `Checkbox` su `Android` .
- Stiamo facendo la maggior parte del nostro lavoro nel metodo `OnElementChanged` , dove istanziamo e impostiamo il nostro controllo nativo.

Consumare il controllo personalizzato

```
<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>
```

Creazione del renderizzatore personalizzato su ciascuna piattaforma

La procedura per la creazione della classe di rendering personalizzata è la seguente:

1. Creare una sottoclasse della classe `ViewRenderer<T1,T2>` che esegue il rendering del controllo personalizzato. Il primo argomento di tipo dovrebbe essere il controllo personalizzato per il renderer, in questo caso `CheckBox` . Il secondo argomento di tipo dovrebbe essere il controllo

nativo che implementerà il controllo personalizzato.
2. Sovrascrivi il metodo `OnElementChanged` che `OnElementChanged` rendering del controllo personalizzato e della logica di scrittura per personalizzarlo. Questo metodo viene chiamato quando viene creato il controllo `Xamarin.Forms` corrispondente.
3. Aggiungi un attributo `ExportRenderer` alla classe di rendering personalizzata per specificare che verrà utilizzato per il rendering del controllo personalizzato `Xamarin.Forms` . Questo attributo viene utilizzato per registrare il renderizzatore personalizzato con `Xamarin.Forms` .

Creazione del renderizzatore personalizzato per 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;
            }
        }
    }
}
```

Creazione del renderizzatore personalizzato per iOS

Poiché in iOS la casella di controllo non è integrata, creeremo prima un `CheckBoxView` e poi creeremo un renderer per la nostra casella di controllo Xamarin.Forms.

`CheckBoxView` è basato su due immagini: checked_checkbox.png e unchecked_checkbox.png, quindi la proprietà `Color` verrà ignorata.

La vista CheckBox:

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

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

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

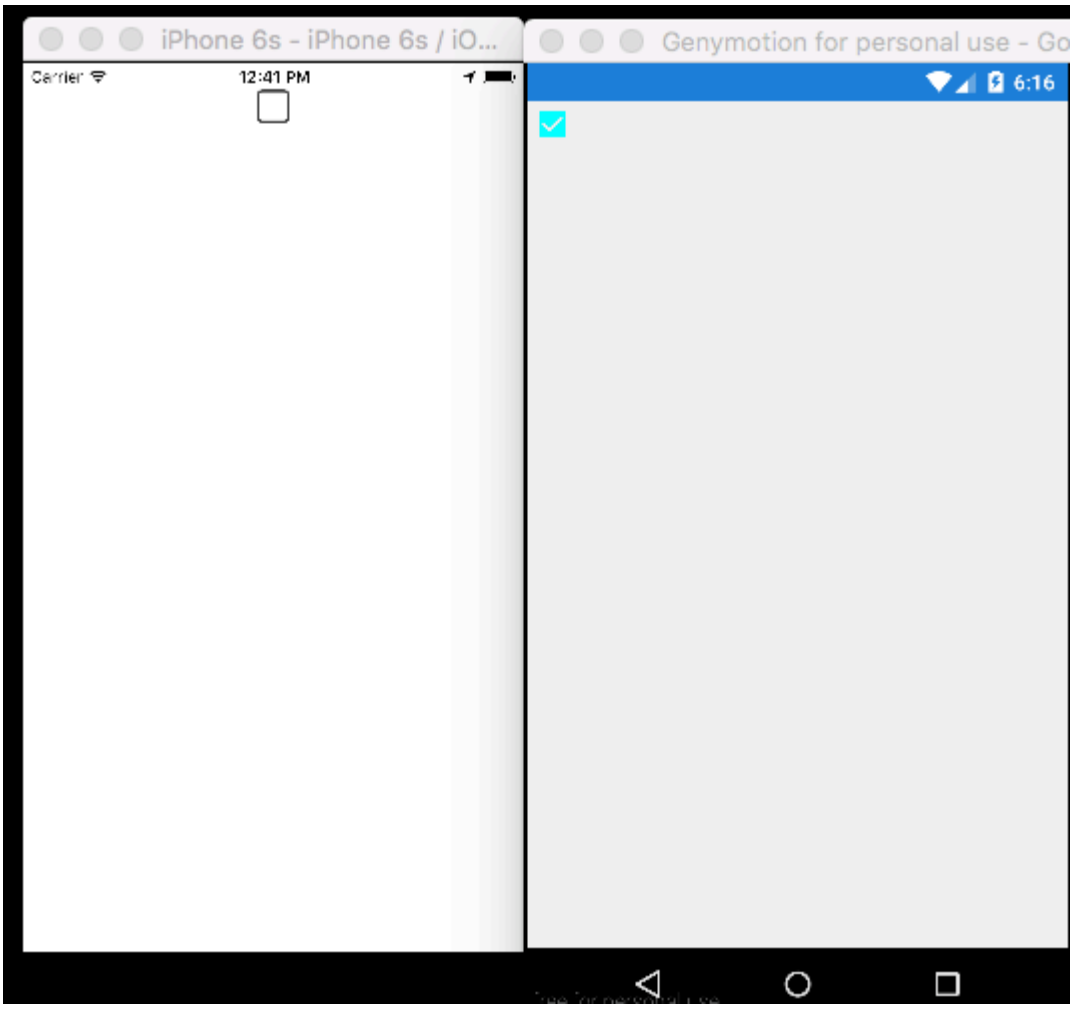
```
        }

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

        public bool Checked
        {
            set { Selected = value; }
            get { return Selected; }
        }

        void Initialize()
        {
            ApplyStyle();

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

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

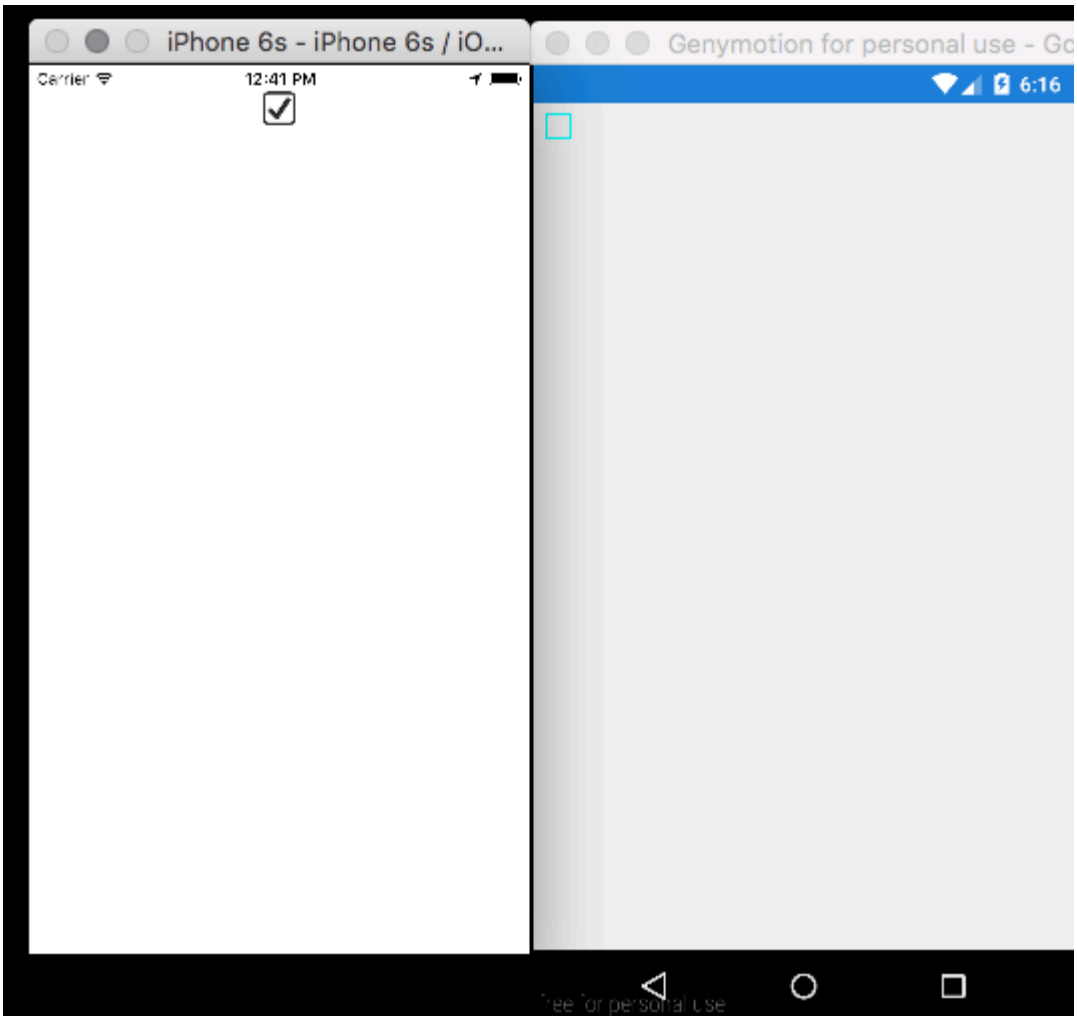
Il renderer personalizzato CheckBox:

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

Risultato:


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



Leggi Creazione di controlli personalizzati online: https://riptutorial.com/it/xamarin-forms/topic/5975/creazione-di-controlli-personalizzati

Capitolo 14: Creazione di controlli personalizzati

Examples

Creazione di un pulsante personalizzato

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

Utilizzo in XAML:

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

Ora puoi utilizzare le tue proprietà in C #:

```
exampleControl.IntData
```

Nota che devi specificare da solo dove si trova la classe TurboButton nel tuo progetto. L'ho fatto in questa linea:

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

Puoi cambiare liberamente "customControls" con un altro nome. Sta a te come lo chiamerai.

Leggi Creazione di controlli personalizzati online: https://riptutorial.com/it/xamarin-forms/topic/6592/creazione-di-controlli-personalizzati

Capitolo 15: Database SQL e API in Xamarin Forms.

Osservazioni

Crea la tua API con il database Microsoft SQL e implementale nell'applicazione Xamarin.

Examples

Creare API utilizzando il database SQL e implementare nei moduli Xamarin,

Blog del codice sorgente

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

Capitolo 16: DependencyService

Osservazioni

Quando si utilizza `DependencyService` genere sono necessarie 3 parti:

- **Interfaccia** : definisce le funzioni che desideri astrarre.
- **Implementazione della piattaforma** : una classe all'interno di ciascun progetto specifico della piattaforma che implementa l'interfaccia definita in precedenza.
- **Registrazione** - Ogni classe di implementazione specifica della piattaforma deve essere registrata con `DependencyService` tramite un attributo metadata. Ciò consente a `DependencyService` di trovare l'implementazione in fase di esecuzione.

Quando si utilizza `DependencyService` è necessario fornire un'implementazione per ciascuna piattaforma scelta come target. Quando non viene fornita un'implementazione, l'applicazione non riuscirà in fase di esecuzione.

Examples

Interfaccia

L'interfaccia definisce il comportamento che si desidera esporre tramite `DependencyService` . Un esempio di utilizzo di `DependencyService` è un servizio Da testo a voce. Al momento non ci sono astrazioni per questa funzione in Xamarin.Forms, quindi è necessario crearne di proprie. Inizia definendo un'interfaccia per il comportamento:

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

Poiché definiamo la nostra interfaccia possiamo codificarci contro di essa dal nostro codice condiviso.

Nota: le classi che implementano l'interfaccia devono avere un costruttore senza parametri per lavorare con `DependencyService` .

implementazione iOS

L'interfaccia che hai definito deve essere implementata in ogni piattaforma mirata. Per iOS questo viene fatto attraverso il framework `AVFoundation` . La seguente implementazione dell'interfaccia `ITextToSpeech` gestisce la pronuncia di un determinato testo in inglese.

```
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);
    }
}
```

Una volta creato il tuo corso, devi abilitare `DependencyService` per scoprirlo in fase di esecuzione. Questo viene fatto aggiungendo un attributo `[assembly]` sopra la definizione della classe e al di fuori di qualsiasi definizione di spazio dei nomi.

```
using AVFoundation;
using DependencyServiceSample.iOS;

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

Questo attributo registra la classe con `DependencyService` modo che possa essere utilizzata quando è necessaria `ITextToSpeech` dell'interfaccia `ITextToSpeech` .

Codice condiviso

Dopo aver creato e registrato le tue classi specifiche per piattaforma puoi iniziare ad agganciarle al tuo codice condiviso. La pagina seguente contiene un pulsante che attiva la funzionalità di sintesi vocale utilizzando una frase predefinita. Utilizza `DependencyService` per recuperare un'implementazione specifica della `ITextToSpeech` di `ITextToSpeech` in fase di esecuzione utilizzando gli SDK nativi.

```
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;
```

```
}
```

Quando si esegue questa applicazione su un dispositivo iOS o Android e si tocca il pulsante, si sentirà l'applicazione pronunciare la frase data.

Implementazione Android

L'implementazione specifica per Android è un po 'più complessa perché ti costringe ad ereditare da un `Java.Lang.Object` nativo e ti costringe ad implementare l'interfaccia `IOnInitListener` . Android richiede di fornire un contesto Android valido per molti dei metodi SDK che espone. Xamarin.Forms espone un oggetto `Forms.Context` che ti fornisce un contesto Android che puoi utilizzare in questi casi.

```
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
}
```

Una volta creato il tuo corso, devi abilitare `DependencyService` per scoprirlo in fase di esecuzione. Questo viene fatto aggiungendo un attributo `[assembly]` sopra la definizione della classe e al di

fuori di qualsiasi definizione di spazio dei nomi.

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


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

Questo attributo registra la classe con `DependencyService` modo che possa essere utilizzata quando è necessaria `ITextToSpeech` dell'interfaccia `ITextToSpeech` .

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

Capitolo 17: effetti

introduzione

Gli effetti semplificano le personalizzazioni specifiche della piattaforma. Quando è necessario modificare le proprietà di Xamarin Forms Control, è possibile utilizzare gli effetti. Quando è necessario sovrascrivere i metodi di Xamarin Forms Control, è possibile utilizzare i renderer personalizzati

Examples

Aggiunta di effetti specifici per piattaforma per un controllo di entrata

1. Crea una nuova app Xamarin Forms utilizzando il file PCL -> Nuova soluzione -> App multipiattaforma -> Moduli Xamarin -> App moduli; Assegna un nome al progetto come `EffectsDemo`
2. Sotto il progetto iOS, aggiungi una nuova classe `Effect` che eredita dalla classe `PlatformEffect` e sovrascrive i metodi `OnAttached` , `OnDetached` e `OnElementPropertyChanged` Notare i due attributi `ResolutionGroupName` ed `ExportEffect` , necessari per consumare questo effetto dal progetto PCL / condiviso.

- `OnAttached` è il metodo in cui entra la logica per la personalizzazione
- `OnDetached` è il metodo in cui avviene la pulizia e la `OnDetached`
- `OnElementPropertyChanged` è il metodo che viene attivato in base alle modifiche alle proprietà di diversi elementi. Per identificare la proprietà giusta, controlla la modifica esatta della proprietà e aggiungi la tua logica. In questo esempio, `OnFocus` darà il colore `Blue` e `OutofFocus` darà Colore `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);
        }
    }
```

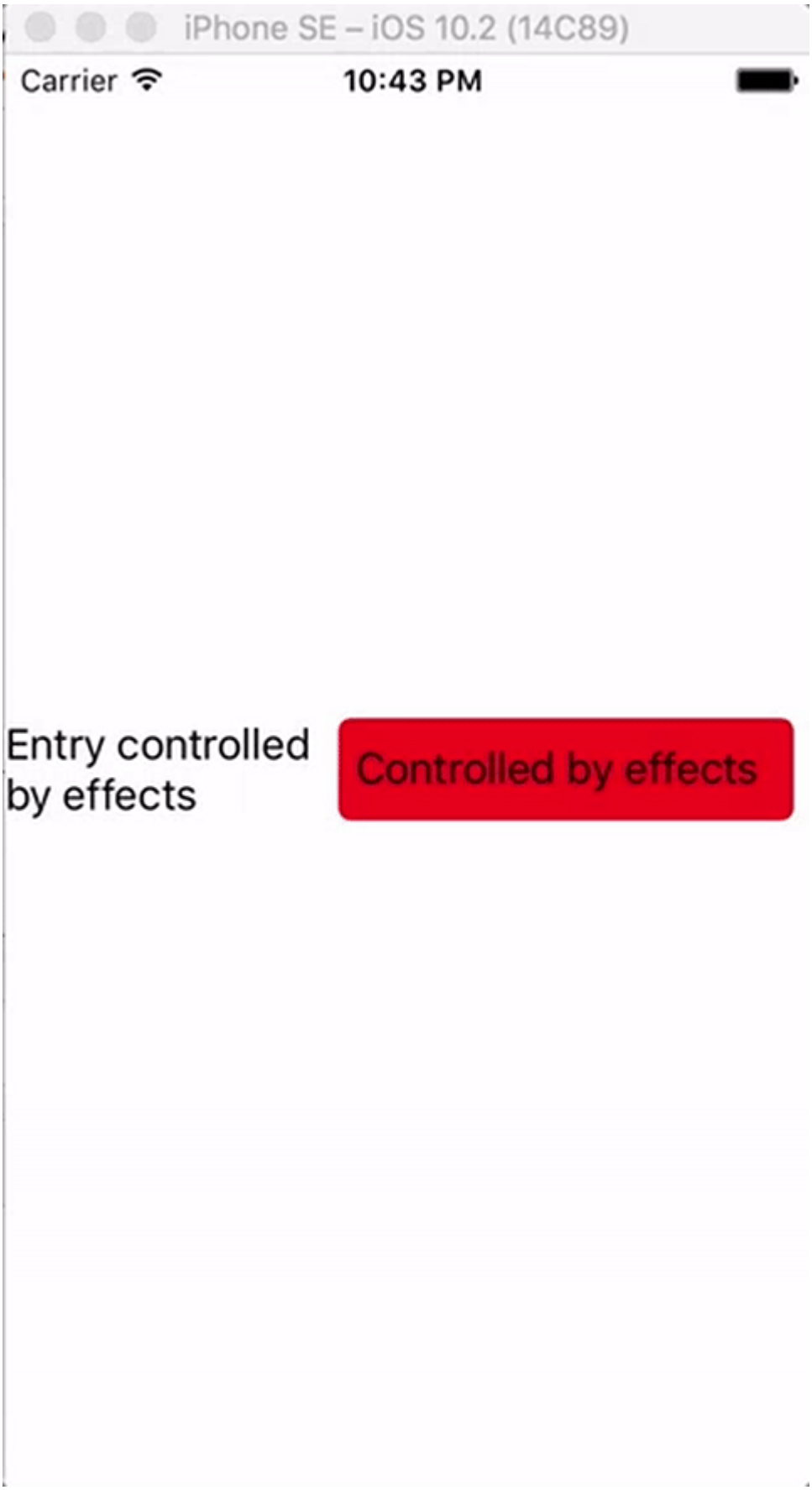
}}

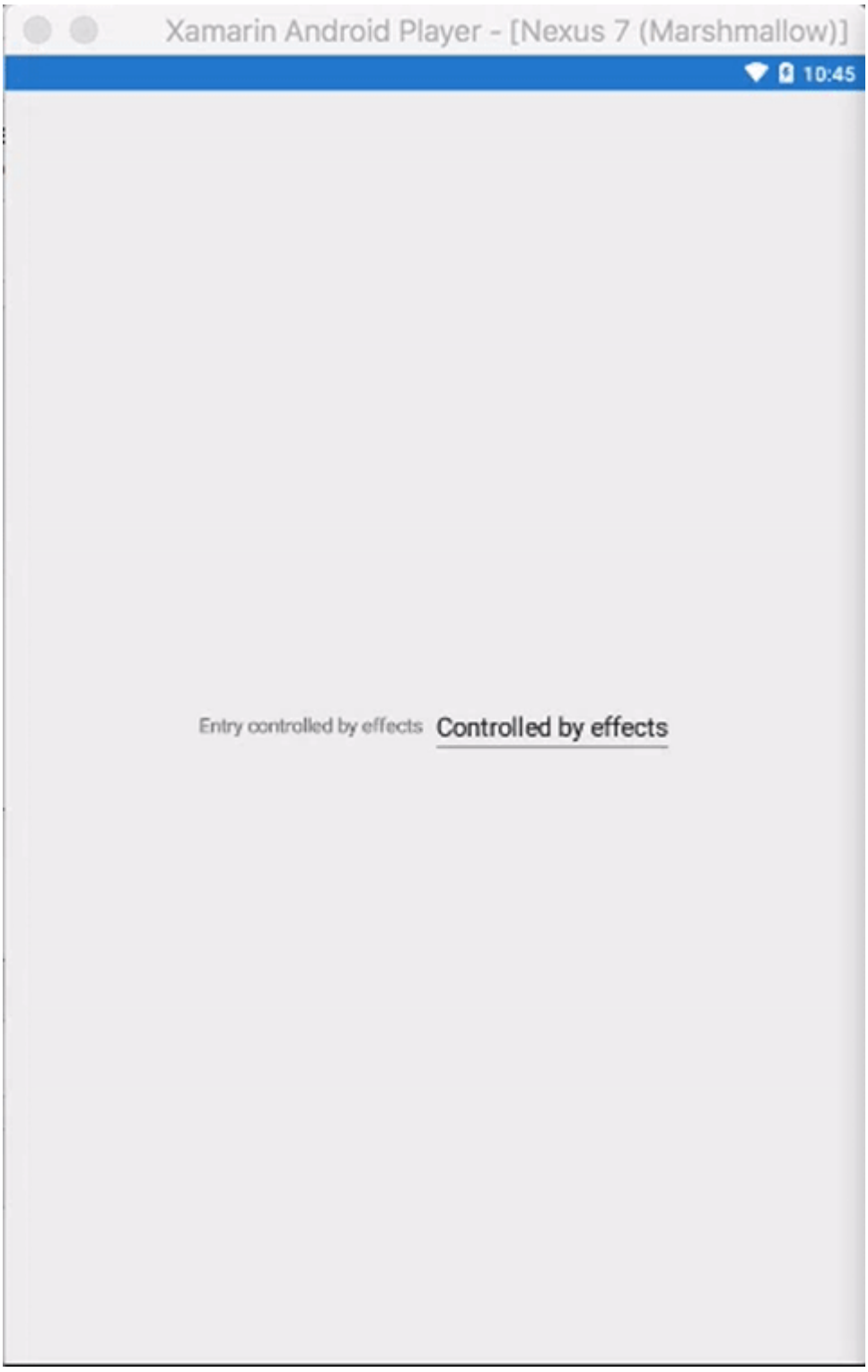
3. Per consumare questo effetto nell'applicazione, sotto il progetto `PCL` , creare una nuova classe denominata `FocusEffect` che eredita da `RoutingEffect` . Questo è essenziale per far sì che il PCL installi l'implementazione specifica della piattaforma dell'effetto. Esempio di codice qui sotto:

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

4. Aggiungi l'effetto al controllo `Entry` in XAML

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



Poiché l'effetto è stato implementato solo nella versione iOS, quando l'app viene eseguita in `iOS Simulator` dopo aver focalizzato le modifiche al colore di sfondo della `Entry` e non accade nulla in `Android Emulator` poiché l' `Effect` non è stato creato nel progetto `Droid`

Leggi effetti online: https://riptutorial.com/it/xamarin-forms/topic/9252/effetti

Capitolo 18: Gesti

Examples

Crea un'immagine toccabile aggiungendo un TapGestureRecognizer

Sono disponibili un paio di riconoscitori predefiniti in Xamarin.Forms, uno dei quali è il `TapGestureRecognizer` .

Puoi aggiungerli praticamente a qualsiasi elemento visivo. Dai un'occhiata a una semplice implementazione che si lega a `Image` . Ecco come farlo in codice.

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

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

O in XAML:

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

Qui il comando viene impostato utilizzando l'associazione dati. Come puoi vedere puoi anche impostare `NumberOfTapsRequired` per abilitarlo per più tap prima che entri in azione. Il valore predefinito è 1 tocco.

Altri gesti sono Pizzico e Pan.

Ingrandisci un'immagine con il gesto di pizzica

Per rendere zoomabile `Image` (o qualsiasi altro elemento visivo) dobbiamo aggiungere un `PinchGestureRecognizer` ad esso. Ecco come farlo nel codice:

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

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

Ma può anche essere fatto da XAML:

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

Nel gestore di eventi accompagnato dovresti fornire il codice per ingrandire l'immagine. Ovviamente anche altri usi possono essere implementati.

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

Altri gesti sono Tap e Pan.

Mostra tutto il contenuto dell'immagine ingrandito con PanGestureRecognizer

Quando si ha una zoomata `Image` (o altri contenuti) si consiglia di trascinare tutto `Image` per mostrare tutti i suoi contenuti nello zoom Stato.

Questo può essere ottenuto implementando il PanGestureRecognizer. Dal codice questo sembra così:

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

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

Questo può essere fatto anche da XAML:

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

Nell'evento code-behind ora puoi gestire il panning di conseguenza. Utilizzare questa firma del metodo per gestirla:

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

Posiziona un segnaposto dove l'utente ha toccato lo schermo con MR.Gestures

Gli Xamarin integrati nei riconoscitori di gesti forniscono solo una gestione tattile molto semplice.

Ad esempio, non c'è modo di ottenere la posizione di un dito commovente. MR.Gestures è un componente che aggiunge 14 diversi eventi di gestione del tocco. La posizione delle dita toccando fa parte delle `EventArgs` passati a tutti gli eventi MR.Gestures.

Se si desidera posizionare un pin in qualsiasi punto dello schermo, il modo più semplice è utilizzare `MR.Gestures.AbsoluteLayout` che gestisce l'evento `Tapping` .

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

Come puoi vedere il `Tapping="OnTapping"` sembra più simile a .NET rispetto alla sintassi Xamarins con `GestureRecognizers` nidificati. Quella sintassi è stata copiata da iOS e ha un odore un po 'per gli sviluppatori .NET.

Nel tuo codice `OnTapping` potresti aggiungere il gestore di `OnTapping` questo modo:

```
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);
    }
}
```

Invece dell'evento `Tapping` , è possibile utilizzare anche il `TappingCommand` e collegarlo al ViewModel, ma ciò complicherebbe le cose in questo semplice esempio.

Altri esempi di MR.Gestures possono essere trovati nell'app GestureSample su GitHub e sul sito Web MR.Gestures . Questi mostrano anche come utilizzare tutti gli altri eventi touch con gestori di eventi, comandi, MVVM, ...

Leggi Gesti online: https://riptutorial.com/it/xamarin-forms/topic/3914/gesti

Capitolo 19: Gesto Xamarin

Examples

Tocca Gesto

Con il gesto del tocco, puoi rendere selezionabile qualsiasi elemento dell'interfaccia utente (immagini, pulsanti, impilati, ...):

(1) Nel codice, utilizzando l'evento:

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

(2) Nel codice, usando `ICommand` (con MVVM-Pattern , ad esempio):

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

(3) O in Xaml (con evento e `ICommand` , solo uno è necessario):

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

Leggi Gesto Xamarin online: https://riptutorial.com/it/xamarin-forms/topic/7994/gesto-xamarin

Capitolo 20: Gesto Xamarin

Examples

Evento di gesto

Quando mettiamo il controllo di Label, l'etichetta non fornisce alcun evento. <Etichetta x: Nome = "lblSignUp Testo =" Non hai un account? "/> Come mostrato, l'etichetta mostra solo lo scopo.

Quando l'utente vuole sostituire Button with Label, allora diamo l'evento per Label. Come mostrato di seguito:

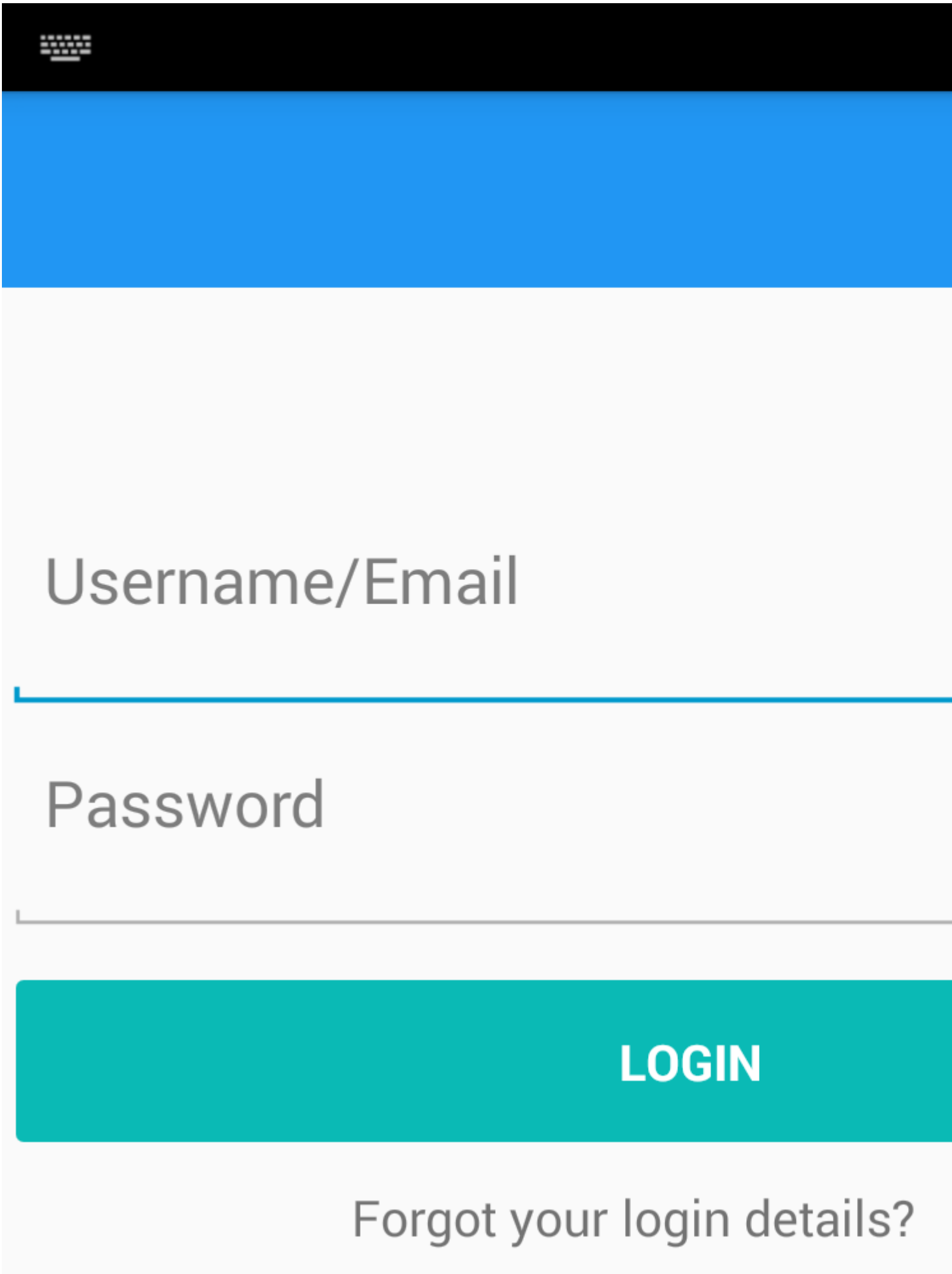
XAML

```
<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);
```

La schermata sotto mostra l'evento etichetta. Schermata 1: L'etichetta "Non hai un account?" come mostrato in fondo.


Username/Email
Password
LOGIN
Forgot your login details?

Capitolo 21: La gestione delle eccezioni

Examples

Un modo per segnalare le eccezioni su iOS

Vai al file `Main.cs` nel **progetto iOS** e modifica il codice esistente, come presentato di seguito:

```
    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` interfaccia di `ILittleWatson` , utilizzata nel codice portatile, potrebbe avere il seguente aspetto:

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

    void SaveReport(Exception ex);
}
```

Implementazione per il **progetto 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);
        }
    }
}
```

E poi, da qualche parte, dove inizia l'app, metti:

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

Leggi La gestione delle eccezioni online: https://riptutorial.com/it/xamarin-forms/topic/6428/la-gestione-delle-eccezioni

Capitolo 22: Lavorare con database locali

Examples

Utilizzo di SQLite.NET in un progetto condiviso

SQLite.NET è una libreria open source che consente di aggiungere il supporto dei database locali utilizzando `SQLite` versione 3 in un progetto `Xamarin.Forms` .

I passaggi seguenti mostrano come includere questo componente in un progetto condiviso `Xamarin.Forms` :

1. Scarica l'ultima versione della classe SQLite.cs e aggiungila al Progetto condiviso.

2. Ogni tabella che verrà inclusa nel database deve essere modellata come classe nel Progetto condiviso. Una tabella viene definita aggiungendo almeno due attributi nella classe: `Table` (per la classe) e `PrimaryKey` (per una proprietà).

Per questo esempio, una nuova classe denominata `Song` viene aggiunta al progetto condiviso, definita come segue:

```
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. Successivamente, aggiungi una nuova classe chiamata `Database` , che eredita dalla classe `SQLiteConnection` (inclusa in SQLite.cs). In questa nuova classe, viene definito il codice per l'accesso al database, la creazione di tabelle e le operazioni CRUD per ogni tabella. Il codice di esempio è mostrato di seguito:

```
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. Come si può vedere nel passaggio precedente, il costruttore della nostra classe `Database` include un parametro `path` , che rappresenta la posizione del file che archivia il file di database SQLite. Un oggetto `Database` statico può essere dichiarato in `App.cs` Il `path` è specifico per la piattaforma:

```
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. Ora chiama semplicemente l'oggetto `DB` tramite la classe `App` ogni volta che devi eseguire un'operazione CRUD alla tabella `Songs` . Ad esempio, per inserire una nuova `Song` dopo che l'utente ha fatto clic su un pulsante, è possibile utilizzare il seguente codice:

```
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);
}
```

Lavorare con database locali utilizzando xamarin.forms in visual studio 2015

Esempio SQlite Passo dopo passo Spiegazione

1. I passaggi seguenti mostrano come includere questo componente in un progetto condiviso Xamarin.Form: aggiungere pacchetti in (pcl, Andriod, Windows, Ios) Aggiungi riferimenti Fare clic su **Gestisci pacchetti Nuget** -> fare clic su Sfoglia per installare **SQLite.Net.Core-PCL** , **SQLite Net Extensions** dopo aver completato l'installazione, controllalo una volta nei riferimenti quindi

2. Per aggiungere Class **Employee.cs** sotto il codice

```
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. Per aggiungere un'interfaccia ISQLite

```
using SQLite.Net;
```

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

4. È possibile creare una classe per le logiche del database e seguire i seguenti metodi.

utilizzando SQLite.Net; using System.Collections.Generic; using System.Linq; usando Xamarin.Forms; namespace DatabaseEmployeeCreation.SqlLite.ViewModel {public class DatabaseLogic {static object locker = new object (); Database 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. creare 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. per visualizzare EmployeeDetails sotto il codice sottostante

```
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. Per implementare il metodo nel metodo Android e ios GetConnection ()

```
    using System;
    using Xamarin.Forms;
    using System.IO;
    using DatabaseEmployeeCreation.Droid;
    using DatabaseEmployeeCreation.SqlLite.ViewModel;
    using SQLite;
    using SQLite.Net;

    [assembly: Dependency(typeof(SQLiteEmployee_Andriod))]
    namespace DatabaseEmployeeCreation.Droid
    {
        public class SQLiteEmployee_Andriod : ISQLite
        {
            public SQLiteEmployee_Andriod()
            {
            }

            #region ISQLite implementation
            public SQLiteConnection GetConnection()
            {
                //var sqliteFilename = "EmployeeSQLite.db3";
                //string documentsPath =
System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal); // Documents
folder
                //var path = Path.Combine(documentsPath, sqliteFilename);

                //// This is where we copy in the prepopulated database
                //Console.WriteLine(path);
                //if (!File.Exists(path))
                //{
                //    var s =
Forms.Context.Resources.OpenRawResource(Resource.Raw.EmployeeSQLite);  // RESOURCE NAME ###

                //    // create a write stream
                //    FileStream writeStream = new FileStream(path, FileMode.OpenOrCreate,
```

```
FileAccess.Write);
                //    // write to the stream
                //    ReadWriteStream(s, writeStream);
                //}

                //var conn = new SQLiteConnection(path);

                //// Return the database connection
                //return conn;
                var filename = "DatabaseEmployeeCreationSQLite.db3";
                var documentspath =
Environment.GetFolderPath(Environment.SpecialFolder.Personal);
                var path = Path.Combine(documentspath, filename);
                var platform = new SQLite.Net.Platform.XamarinAndroid.SQLitePlatformAndroid();
                var connection = new SQLite.Net.SQLiteConnection(platform, path);
                return connection;
            }

            //public  SQLiteConnection GetConnection()
            //{
            //    var filename = "EmployeeSQLite.db3";
            //    var documentspath =
Environment.GetFolderPath(Environment.SpecialFolder.Personal);
            //    var path = Path.Combine(documentspath, filename);

            //    var platform = new
SQLite.Net.Platform.XamarinAndroid.SQLitePlatformAndroid();
            //    var connection = new SQLite.Net.SQLiteConnection(platform, path);
            //    return connection;
            //}
            #endregion

            /// <summary>
            /// helper method to get the database out of /raw/ and into the user filesystem
            /// </summary>
            void ReadWriteStream(Stream readStream, Stream writeStream)
            {
                int Length = 256;
                Byte[] buffer = new Byte[Length];
                int bytesRead = readStream.Read(buffer, 0, Length);
                // write the required bytes
                while (bytesRead > 0)
                {
                    writeStream.Write(buffer, 0, bytesRead);
                    bytesRead = readStream.Read(buffer, 0, Length);
                }
                readStream.Close();
                writeStream.Close();
            }
        }
    }
```

Spero che questo esempio sopra sia un modo molto semplice che ho spiegato

Leggi Lavorare con database locali online: https://riptutorial.com/it/xamarin-forms/topic/5997/lavorare-con-database-locali

Capitolo 23: Lavorare con Maps

Osservazioni

Se hai intenzione di eseguire il tuo progetto su un altro computer, dovrai generare una nuova chiave API per questo, poiché le impronte digitali SHA-1 non corrisponderanno per diversi computer di compilazione.

Puoi esplorare il progetto, descritto nell'esempio *Aggiungere una mappa in Xamarin.Forms* qui

Examples

Aggiunta di una mappa in Xamarin.Forms (Xamarin Studio)

Puoi semplicemente utilizzare le API native della mappa su ogni piattaforma con Xamarin Forms. Tutto ciò che serve è scaricare il pacchetto *Xamarin.Forms.Maps* da nuget e installarlo su ciascun progetto (incluso il progetto PCL).

Inizializzazione delle mappe

Prima di tutto devi aggiungere questo codice ai tuoi progetti specifici per la piattaforma. Per fare ciò devi aggiungere la chiamata al metodo `Xamarin.FormsMaps.Init` , come negli esempi qui sotto.

---

progetto iOS

File AppDelegate.cs

```
[Register("AppDelegate")]
public partial class AppDelegate : Xamarin.Forms.Platform.iOS.FormsApplicationDelegate
{
    public override bool FinishedLaunching(UIApplication app, NSDictionary options)
    {
        Xamarin.Forms.Forms.Init();
        Xamarin.FormsMaps.Init();

        LoadApplication(new App());

        return base.FinishedLaunching(app, options);
    }
}
```

Progetto Android

File MainActivity.cs

```
[Activity(Label = "MapExample.Droid", Icon = "@drawable/icon", Theme = "@style/MyTheme",
MainLauncher = true, ConfigurationChanges = ConfigChanges.ScreenSize |
ConfigChanges.Orientation)]
public class MainActivity : Xamarin.Forms.Platform.Android.FormsAppCompatActivity
{
    protected override void OnCreate(Bundle bundle)
    {
        TabLayoutResource = Resource.Layout.Tabbar;
        ToolbarResource = Resource.Layout.Toolbar;

        base.OnCreate(bundle);

        Xamarin.Forms.Forms.Init(this, bundle);
        Xamarin.FormsMaps.Init(this, bundle);

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

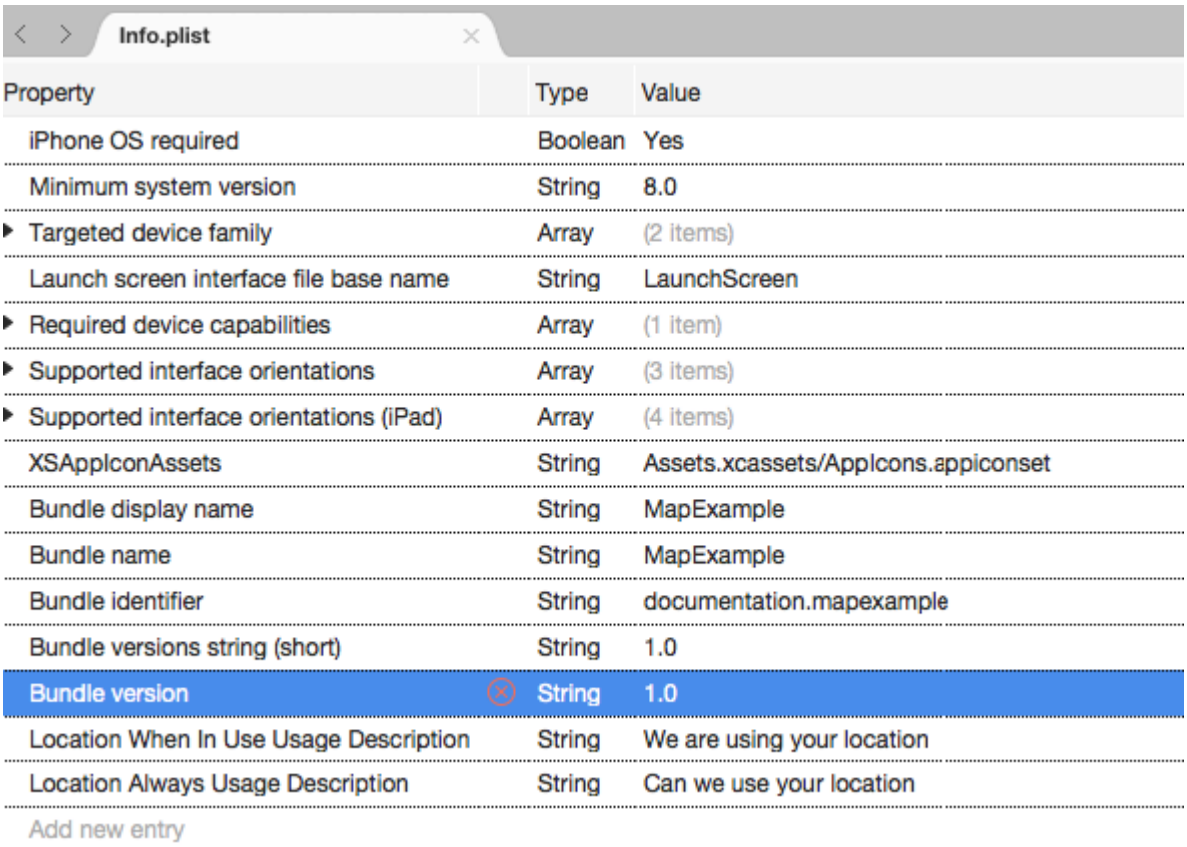
Configurazione della piattaforma

Sono necessari passaggi di configurazione aggiuntivi su alcune piattaforme prima che venga visualizzata la mappa.

progetto iOS

Nel progetto iOS devi solo aggiungere 2 voci al tuo file *Info.plist* :

- `NSLocationWhenInUseUsageDescription` *string* with value `We are using your location`
- `NSLocationAlwaysUsageDescription` *string* with value `Can we use your location`



Progetto Android

Per utilizzare Google Maps devi generare una chiave API e aggiungerla al tuo progetto. Segui le istruzioni qui sotto per ottenere questa chiave:

1. (Facoltativo) Trova la posizione dello strumento keytool (l'impostazione predefinita è `/System/Library/Frameworks/JavaVM.framework/Versions/Current/Commands`)

2. (Facoltativo) Apri il terminale e vai al tuo posto di keytool:

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

3. Esegui il seguente comando keytool:

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

   Dove [USERNAME] è, ovviamente, la tua cartella utente corrente. Dovresti ottenere qualcosa di simile a questo nell'output:

   ```
   Alias name: androiddebugkey
   Creation date: Jun 30, 2016
   Entry type: PrivateKeyEntry
   Certificate chain length: 1
   Certificate[1]:
   Owner: CN=Android Debug, O=Android, C=US
   Issuer: CN=Android Debug, O=Android, C=US
   Serial number: 4b5ac934
   ```

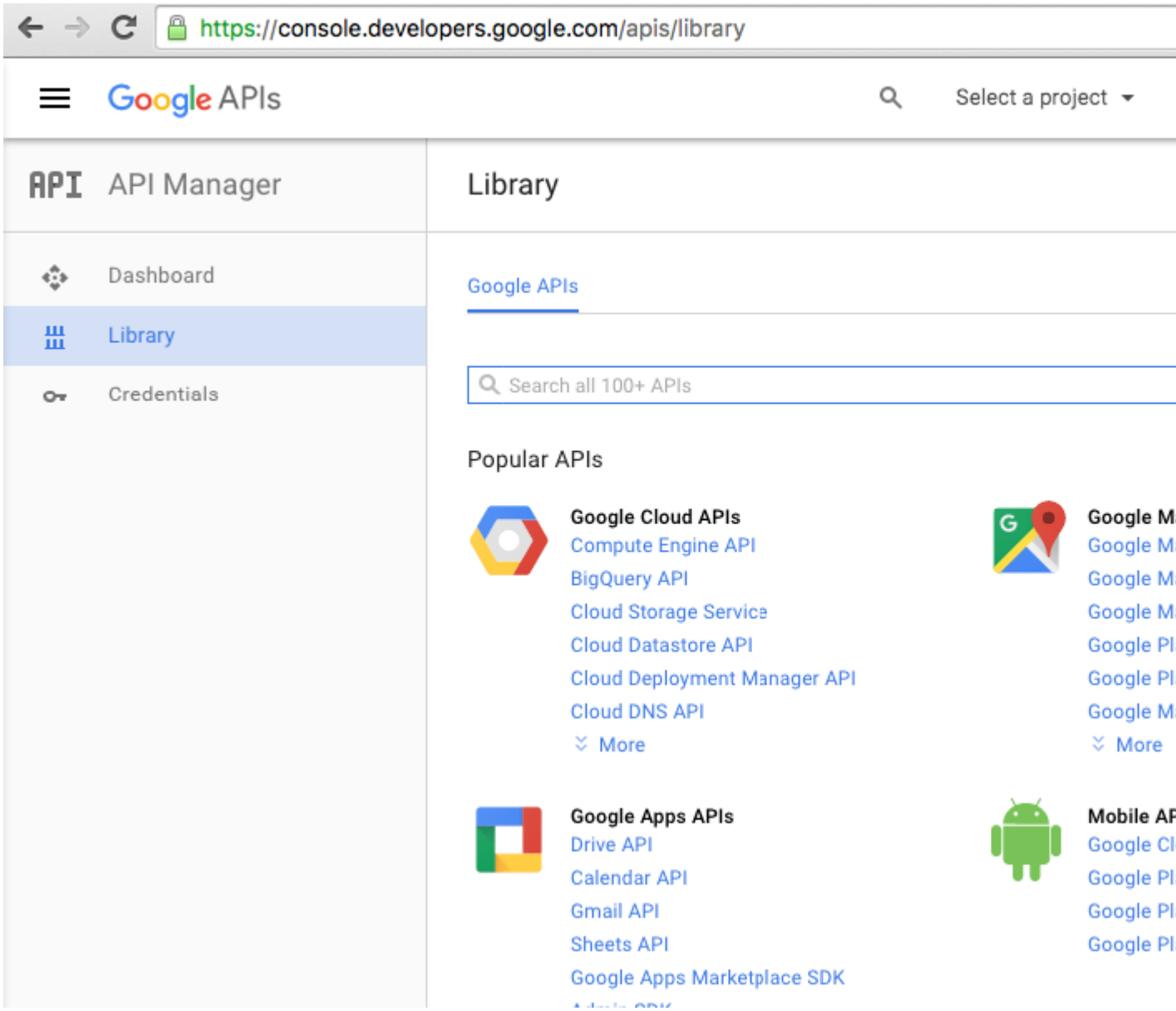
```
Valid from: Thu Jun 30 10:22:00 EEST 2016 until: Sat Jun 23 10:22:00 EEST 2046
Certificate fingerprints:
         MD5:  4E:49:A7:14:99:D6:AB:9F:AA:C7:07:E2:6A:1A:1D:CA
         SHA1: 57:A1:E5:23:CE:49:2F:17:8D:8A:EA:87:65:44:C1:DD:1C:DA:51:95
         SHA256:
70:E1:F3:5B:95:69:36:4A:82:A9:62:F3:67:B6:73:A4:DD:92:95:51:44:E3:4C:3D:9E:ED:99:03:09:9F:90:3F

         Signature algorithm name: SHA256withRSA
         Version: 3
```

4. Tutto ciò di cui abbiamo bisogno in questo output è l'impronta digitale del certificato SHA1. Nel nostro caso è uguale a questo:

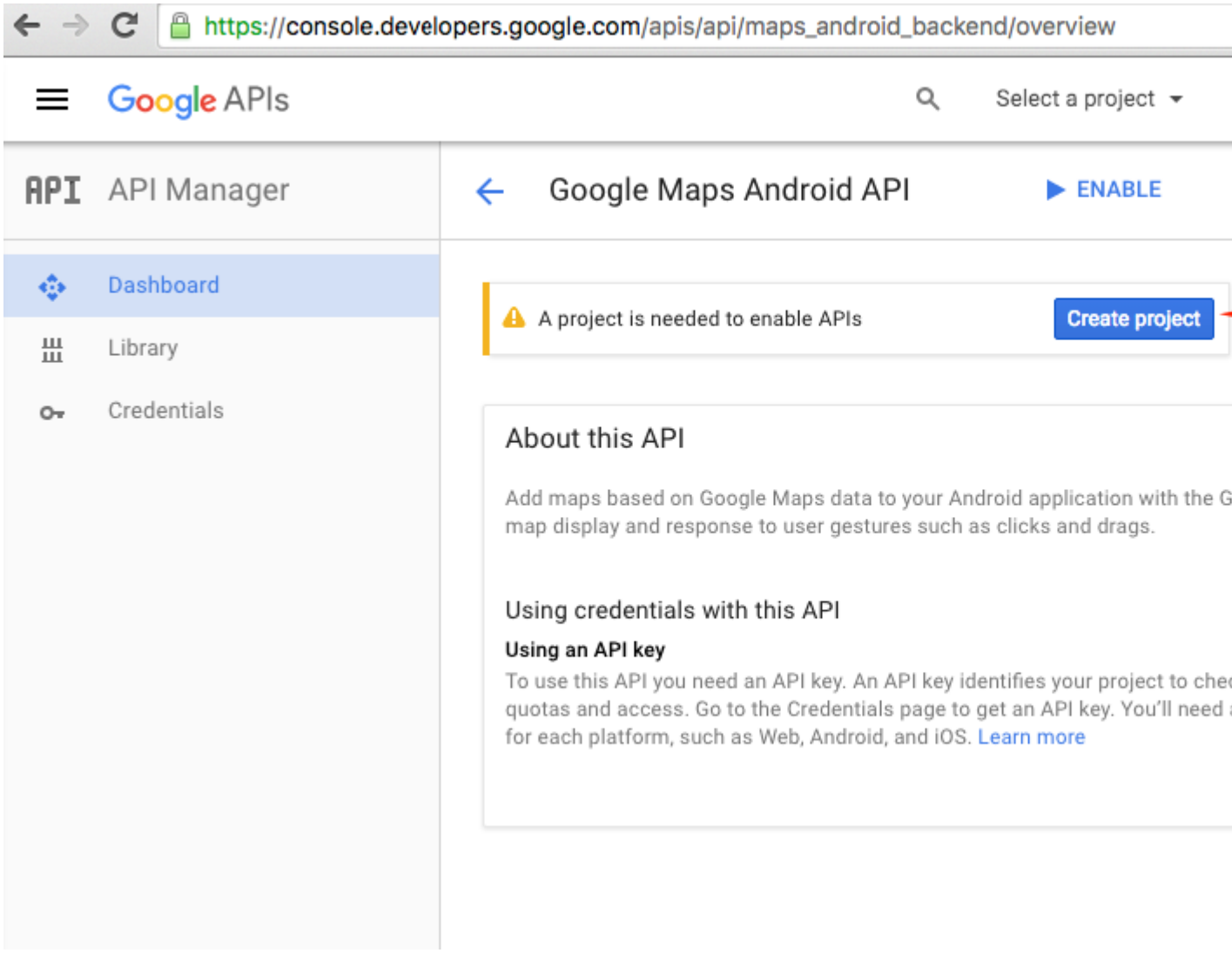
```
57:A1:E5:23:CE:49:2F:17:8D:8A:EA:87:65:44:C1:DD:1C:DA:51:95
```

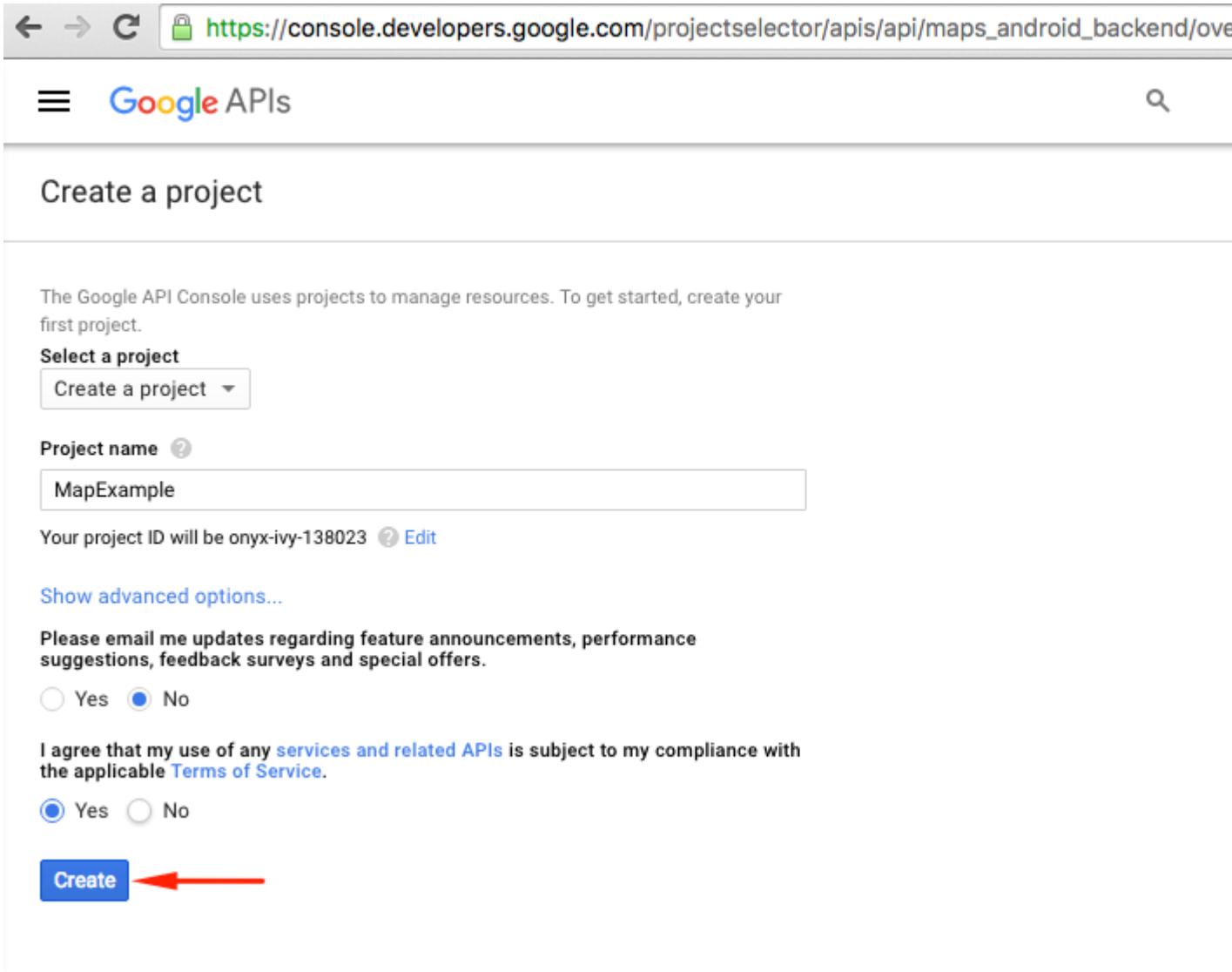
   Copia o salva da qualche parte questa chiave. Ne avremo bisogno in seguito.

5. Vai a Google Developers Console , nel nostro caso dobbiamo aggiungere l' API di Google Maps per Android , quindi sceglierlo:

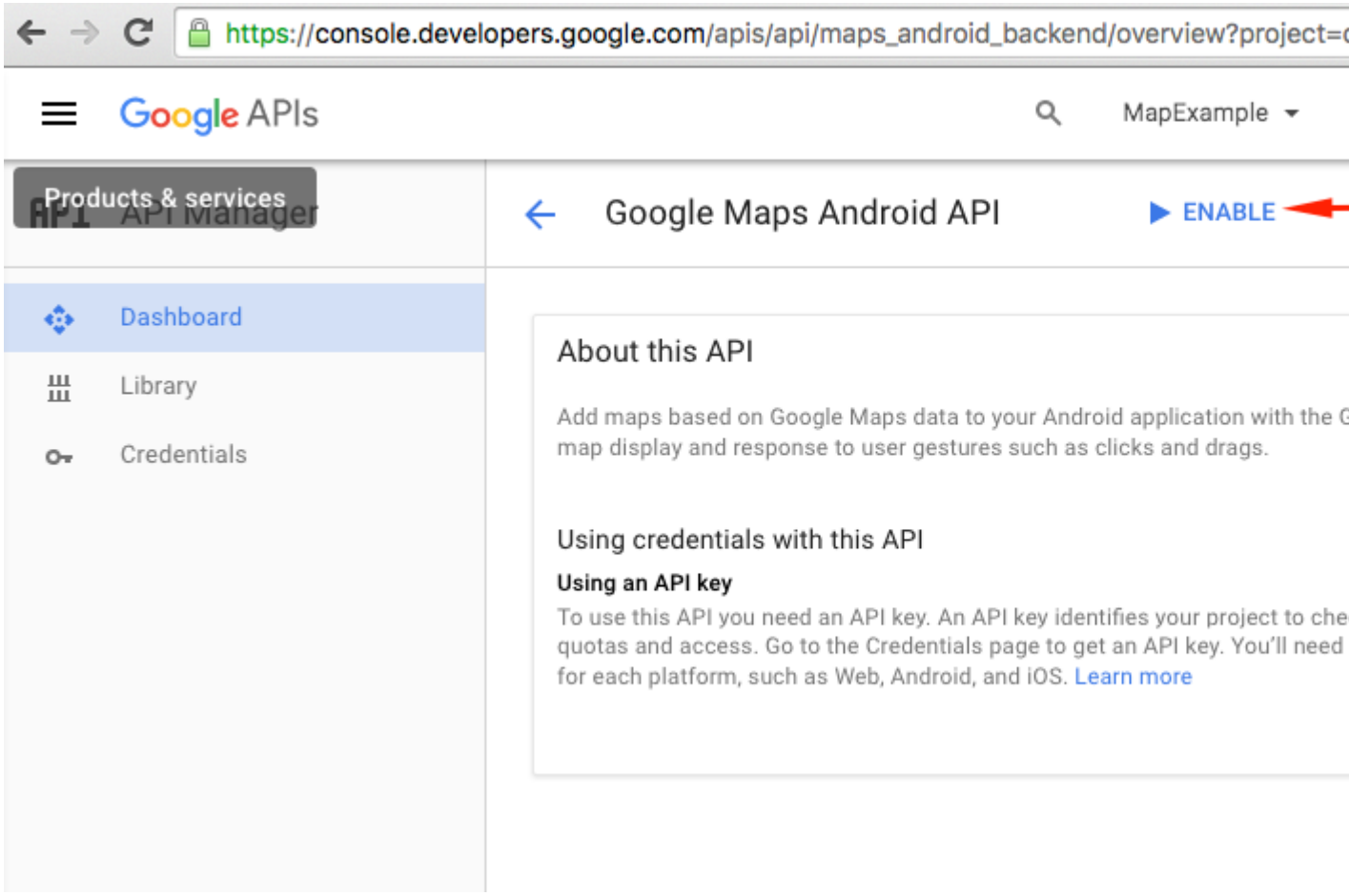


6. Google ti chiederà di creare un progetto per abilitare le API, seguire questo suggerimento e creare il progetto:

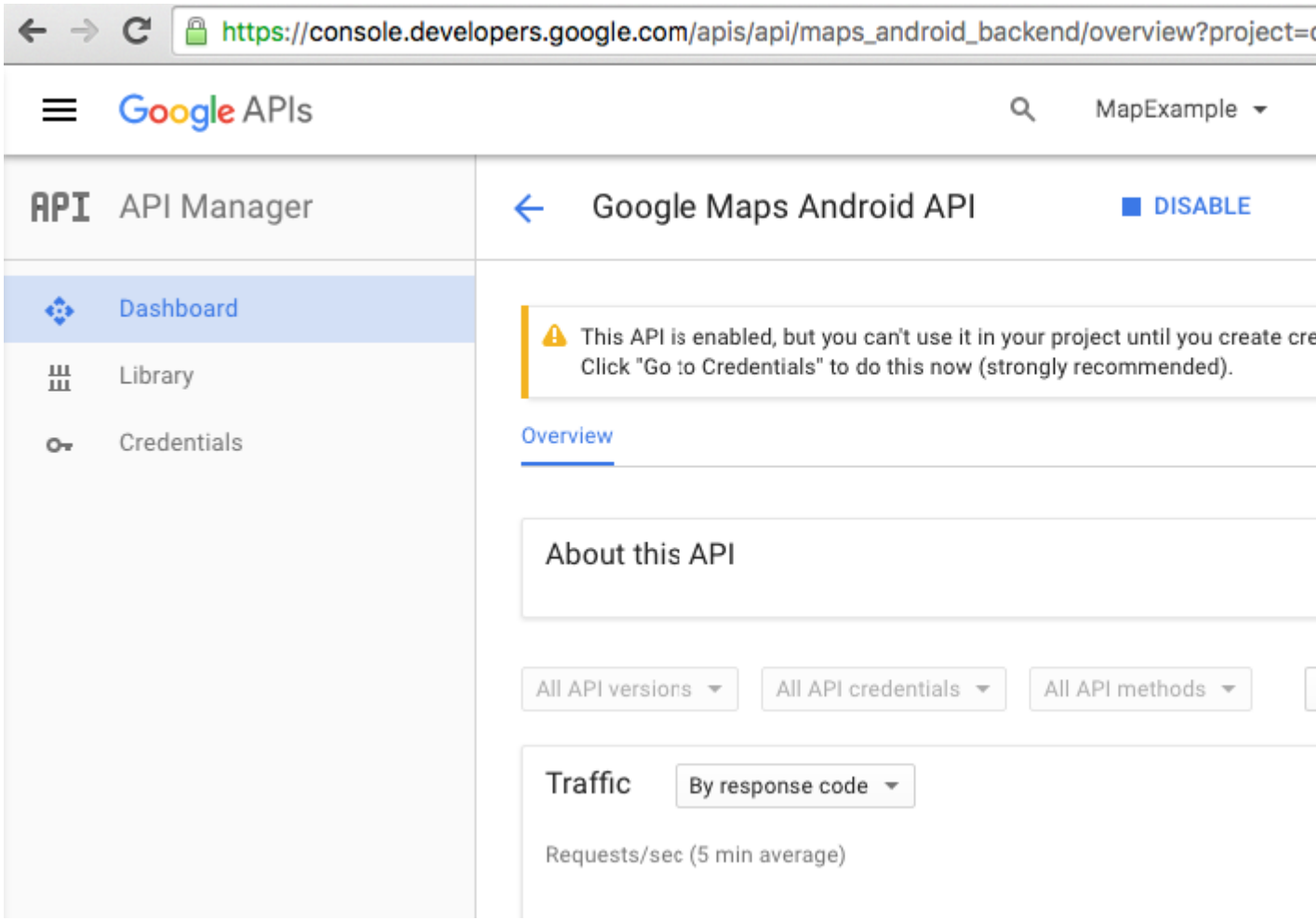




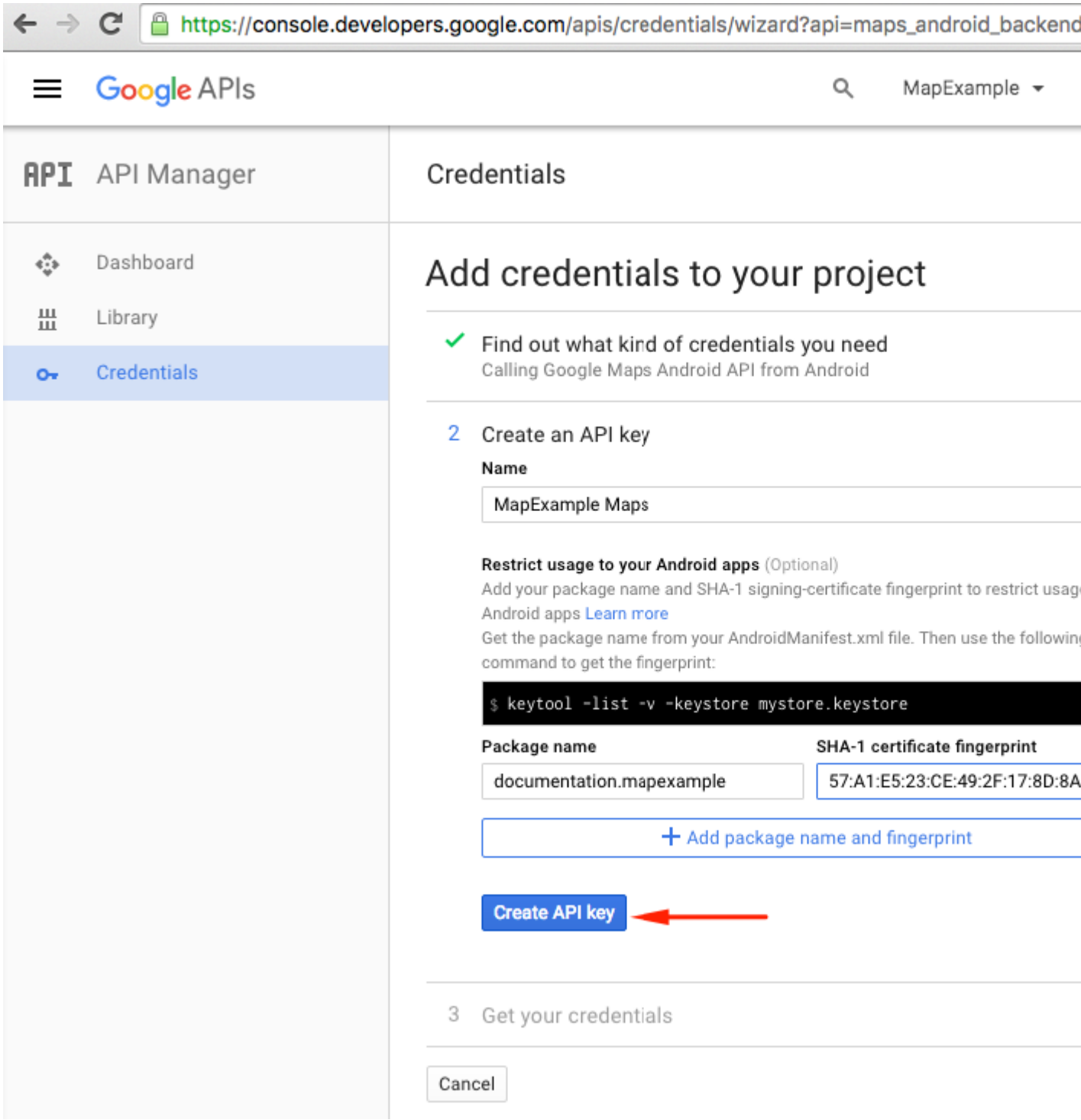
7. Abilita l'API di Google Maps per il tuo progetto:



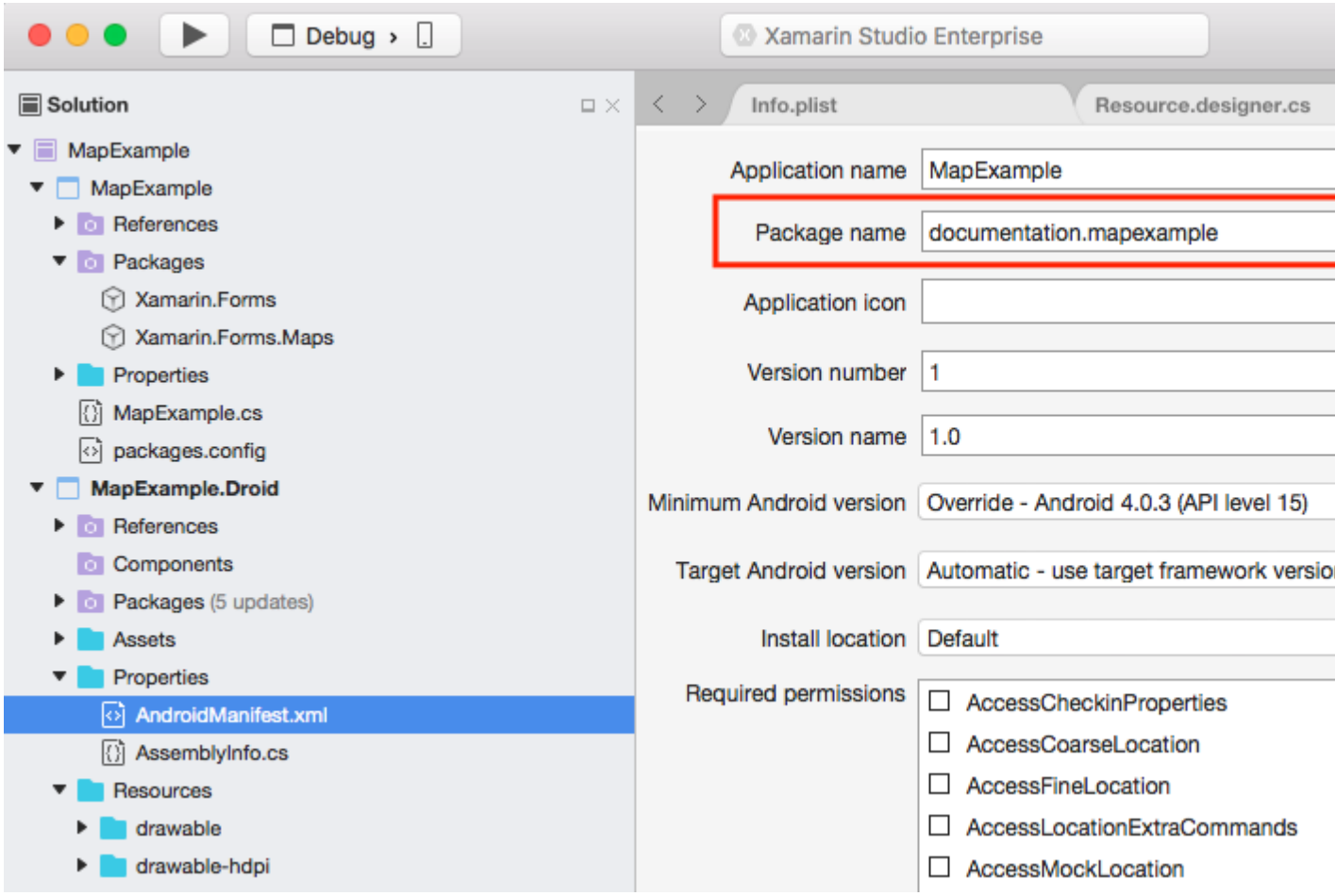
Dopo aver abilitato l'API, devi creare credenziali per la tua app. Segui questo suggerimento:



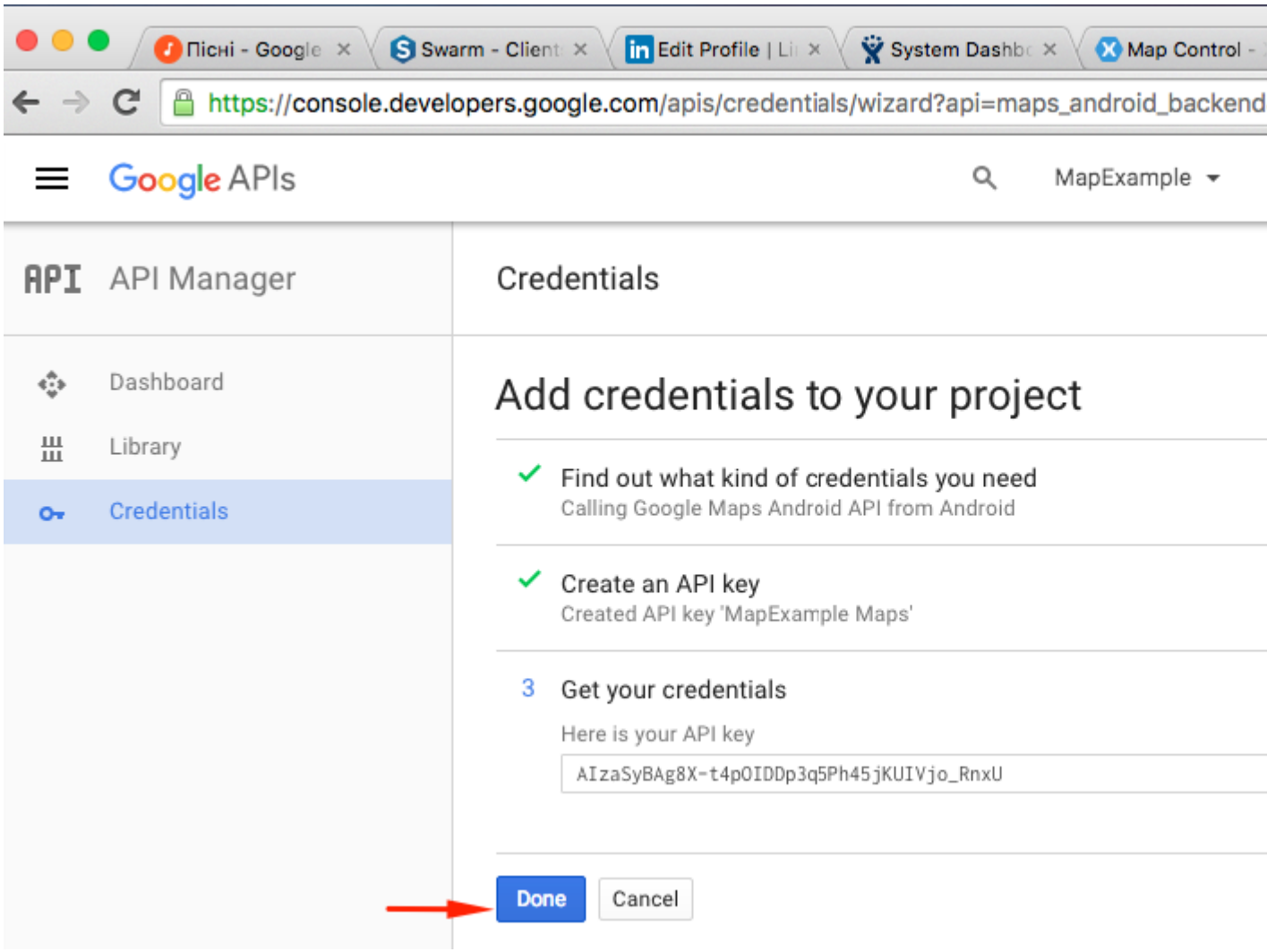
8. Nella pagina successiva scegli la piattaforma Android, tocca "Quali credenziali ho bisogno?" pulsante, crea un nome per la tua chiave API, tocca "Aggiungi nome pacchetto e impronta digitale", inserisci il nome del pacchetto e l'impronta digitale SHA1 dal punto 4 e infine crea una chiave API:



Per trovare il nome del tuo pacchetto in Xamarin Studio vai alla tua soluzione .Droid -> AndroidManifest.xml:



9. Dopo la creazione copia la nuova chiave API (non dimenticare di premere il pulsante "Fine" dopo) e incollalo nel tuo file `AndroidManifest.xml` :



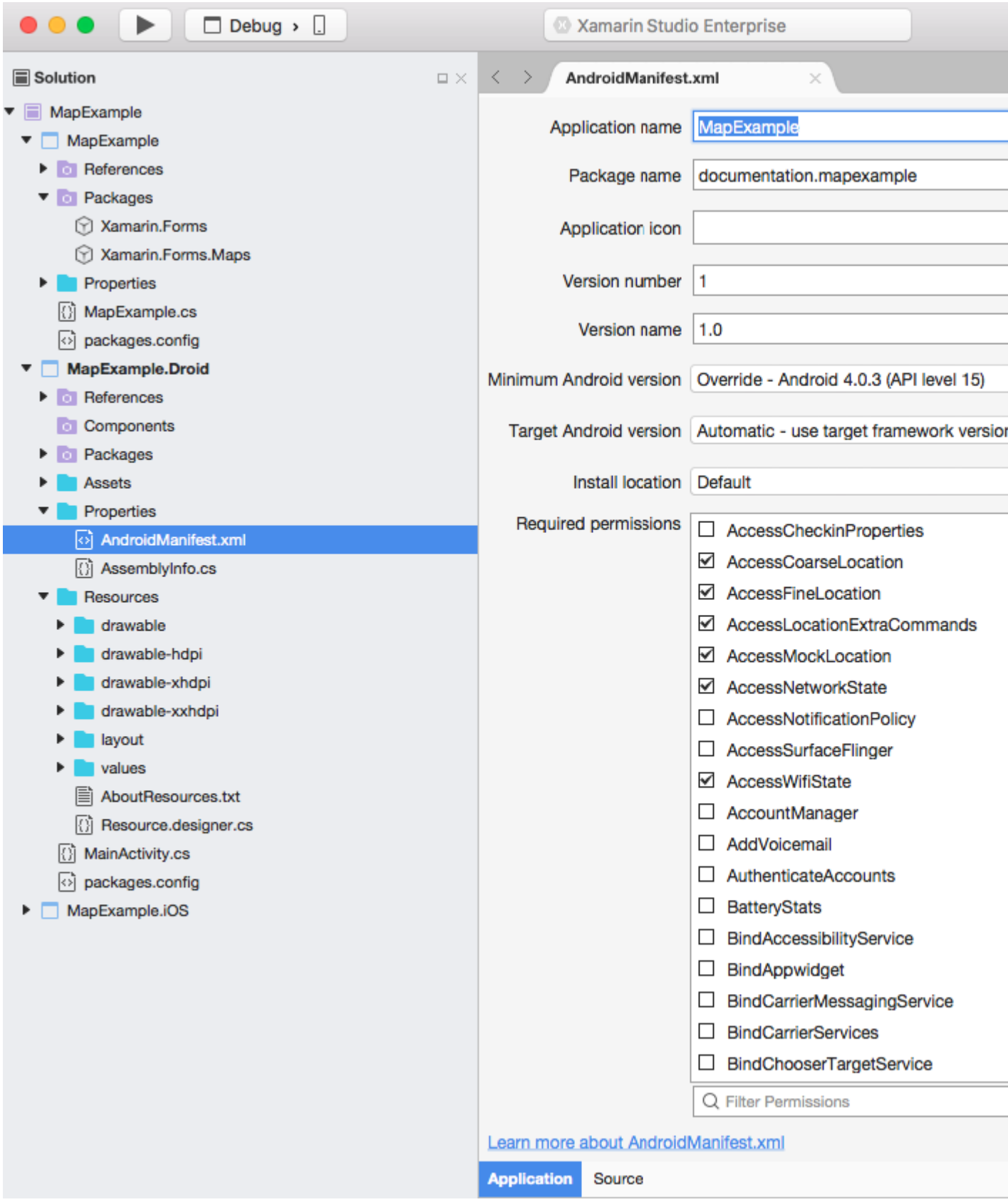
File AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="1"
    android:versionName="1.0"
    package="documentation.mapexample">
    <uses-sdk
        android:minSdkVersion="15" />
    <application
        android:label="MapExample">
        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="AIzaSyBAg8X-t4pOIDDp3q5Ph45jKUIVjo_RnxU" />
        <meta-data
            android:name="com.google.android.gms.version"
            android:value="@integer/google_play_services_version" />
    </application>
</manifest>
```

Dovrai anche abilitare alcune autorizzazioni nel tuo manifest per abilitare alcune funzionalità aggiuntive:

- Accesso alla posizione approssimativa
- Accedi a Fine posizione

- Accesso alla posizione Comandi extra
- Access Mock Location
- Accedi allo stato della rete
- Access Wifi State
- Internet



Sebbene, per scaricare i dati di Maps sono necessarie le ultime due autorizzazioni. Leggi le

autorizzazioni di Android per saperne di più. Ecco tutti i passaggi per la configurazione di Android.

> *Nota* : se vuoi eseguire la tua app su Android simulator, devi installare Google Play Services su di esso. Segui questo tutorial per installare Play Services su Xamarin Android Player. Se non riesci a trovare l'aggiornamento dei servizi di riproduzione di google dopo l'installazione del Play Store, puoi aggiornarlo direttamente dalla tua app, dove hai dipendenza dai servizi di mappe

---

Aggiungere una mappa

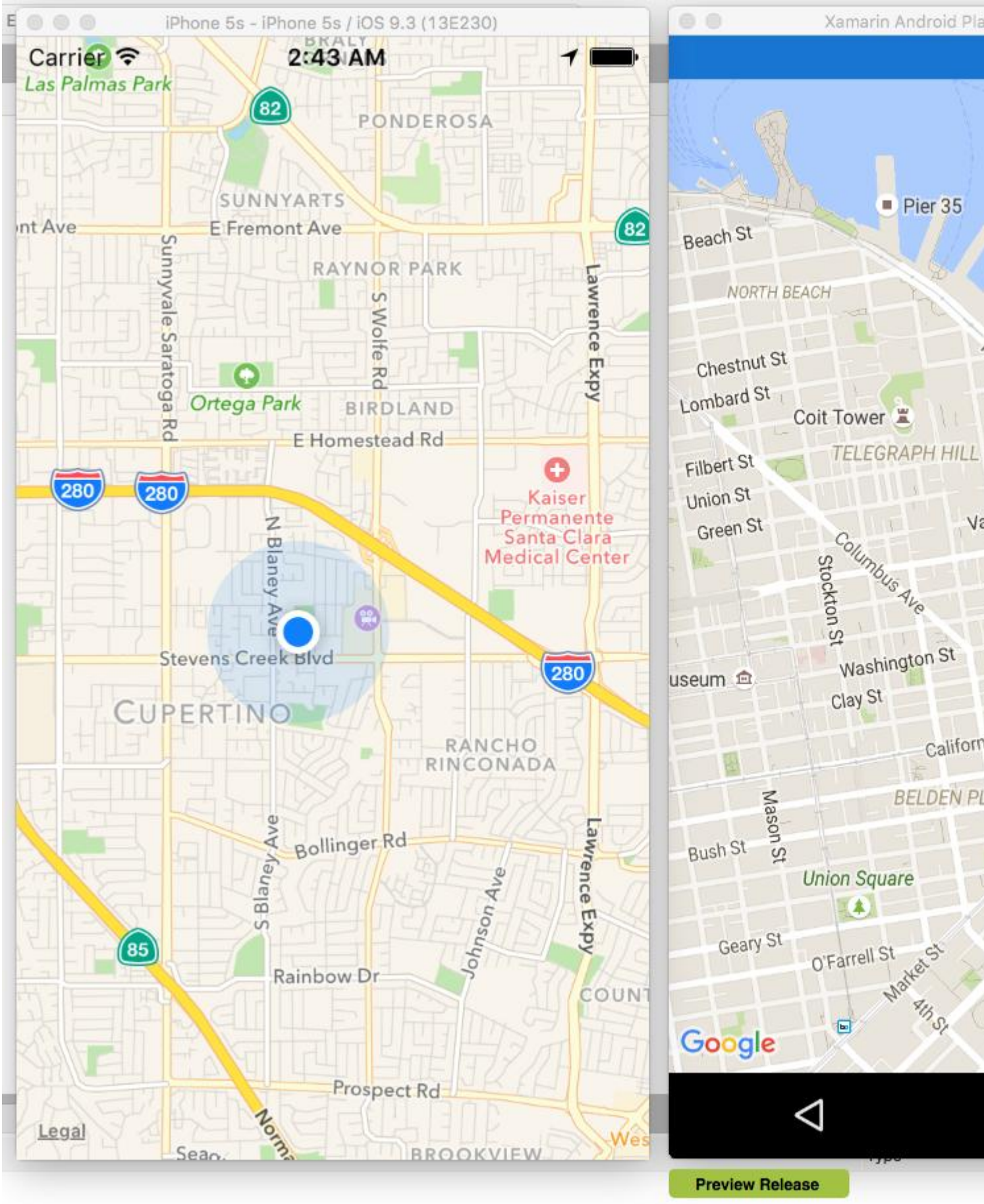
Aggiungere una vista mappa al tuo progetto crossplatform è abbastanza semplice. Ecco un esempio di come puoi farlo (sto usando un progetto PCL senza XAML).

Progetto PCL

File MapExample.cs

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

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

        MainPage = rootPage;
    }
}
```

È tutto. Ora se eseguirai la tua app su iOS o Android, ti mostrerà la vista della mappa:



Leggi Lavorare con Maps online: https://riptutorial.com/it/xamarin-forms/topic/3917/lavorare-con-maps

Capitolo 24: Le notifiche push

Osservazioni

Non esiste un modo uniforme per gestire le notifiche push in Xamarin Form poiché l'implementazione si basa in gran parte su funzioni ed eventi specifici della piattaforma. Pertanto sarà sempre necessario il codice specifico della piattaforma.

Tuttavia, utilizzando `DependencyService` è possibile condividere il maggior numero possibile di codice. Inoltre c'è un plugin progettato per questo da rdelrosario, che può essere trovato sul suo GitHub .

Il codice e gli screenshot sono tratti da una serie di blog di Gerald Versluis che spiega il processo in modo più dettagliato.

Examples

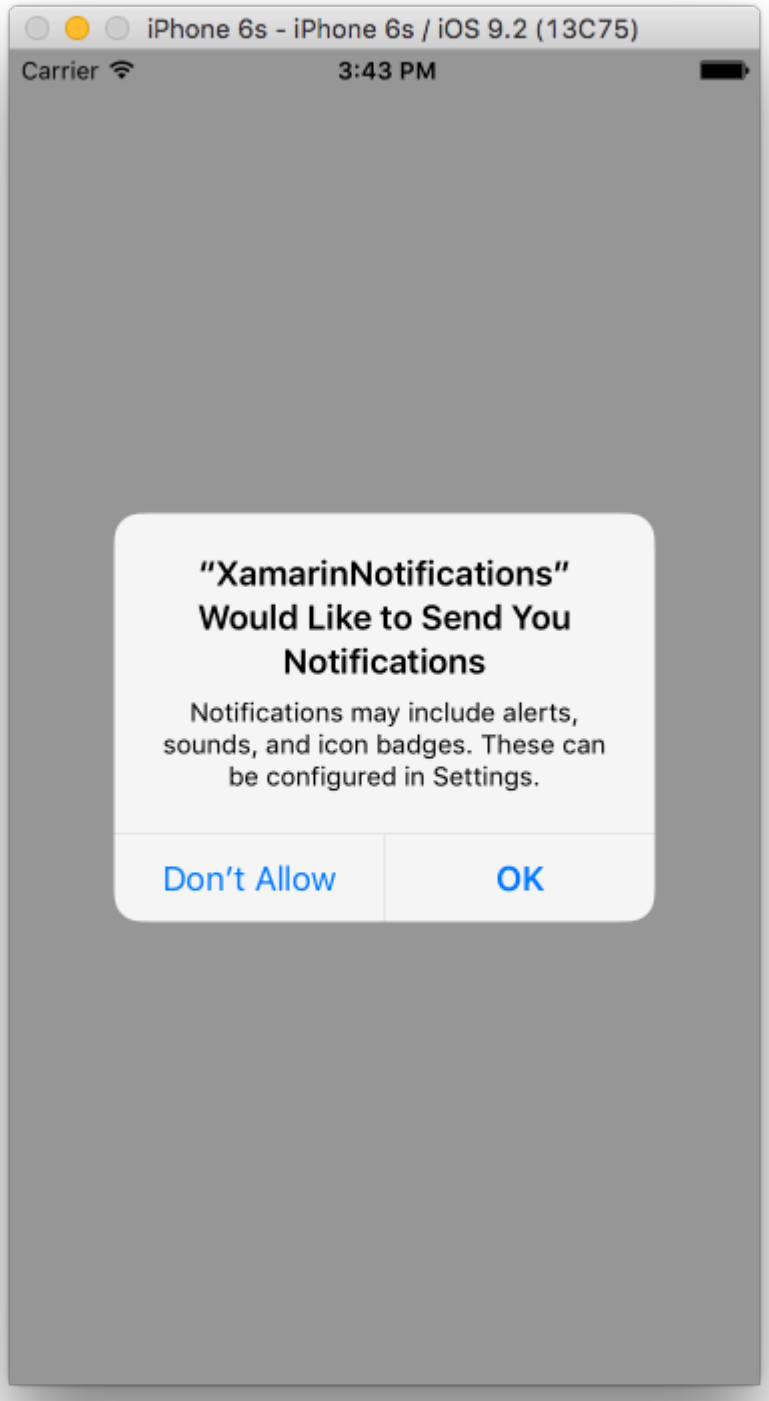
Notifiche push per iOS con Azure

Per avviare la registrazione per le notifiche push è necessario eseguire il codice seguente.

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

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

Questo codice può essere eseguito direttamente all'avvio dell'app in `FinishedLaunching` nel file `AppDelegate.cs` . Oppure puoi farlo ogni volta che un utente decide di voler abilitare le notifiche push.

L'esecuzione di questo codice attiverà un avviso per richiedere all'utente se accetterà che l'app possa inviare loro le notifiche. Quindi implementa anche uno scenario in cui l'utente lo nega!

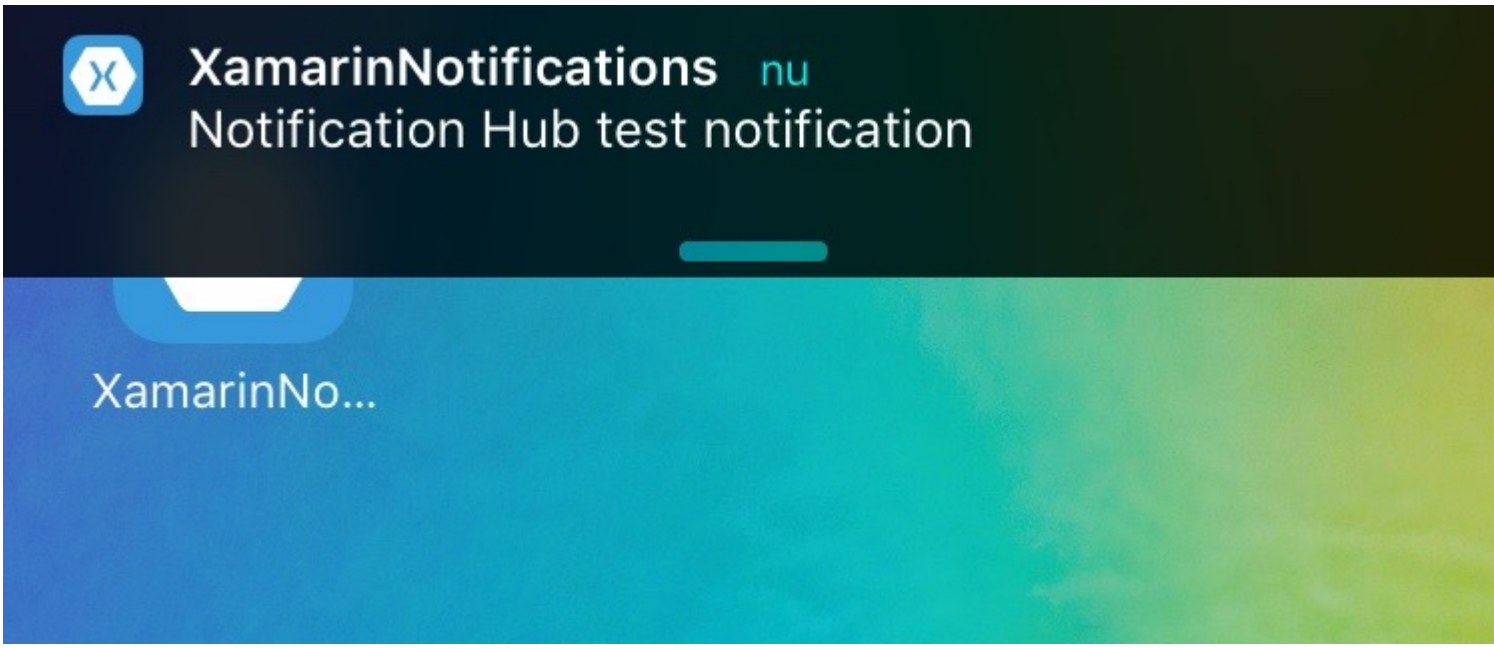


Questi sono gli eventi che necessitano di implementazione per l'implementazione di notifiche push su iOS. Li puoi trovare nel file `AppDelegate.cs` .

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

Quando viene ricevuta una notifica, questo è quello che sembra.



Notifiche push per Android con Azure

L'implementazione su Android richiede un po 'più di lavoro e richiede l'implementazione di un `Service` specifico.

Innanzitutto, controlla se il nostro dispositivo è in grado di ricevere notifiche push e, in tal caso, registrarlo con Google. Questo può essere fatto con questo codice nel nostro file `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());
}
```

I SenderIDs possono essere trovati nel codice sottostante ed è il numero di progetto che si ottiene dal dashboard degli sviluppatori di Google per poter inviare messaggi push.

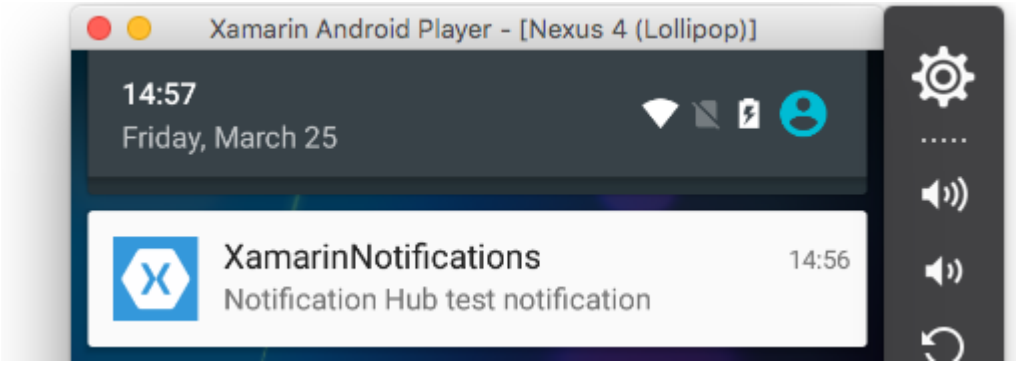
```
using Android.App;
using Android.Content;
using Gcm.Client;
using Java.Lang;
using System;
using WindowsAzure.Messaging;
using XamarinNotifications.Helpers;

// 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);
```

```
        }
    }
}
```

Una notifica di esempio su Android ha questo aspetto.

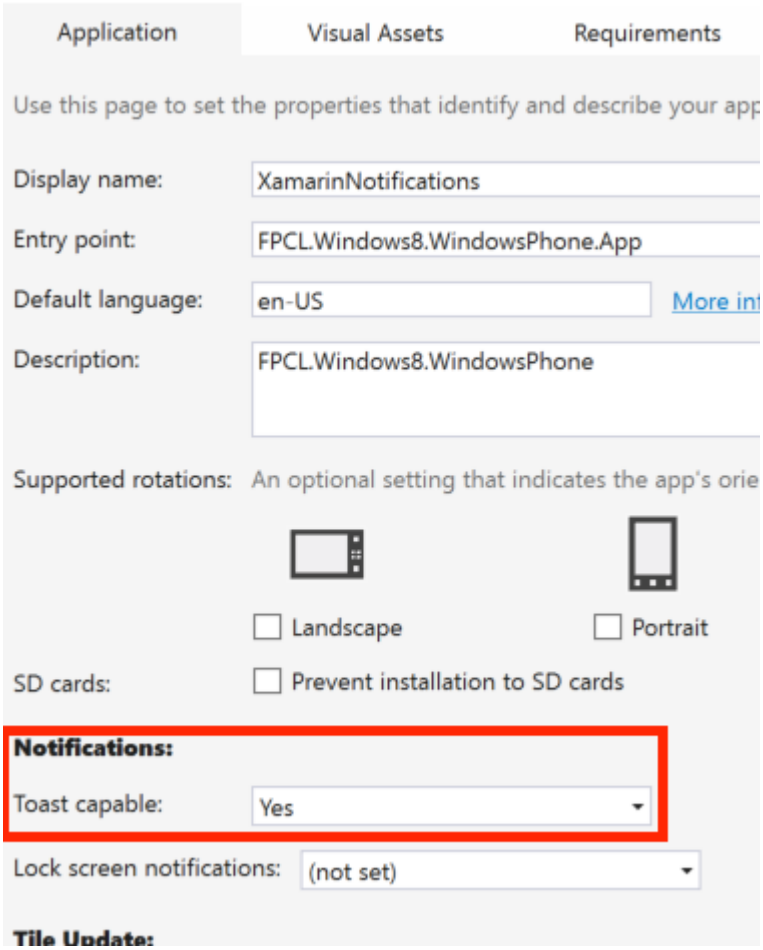


Notifiche push per Windows Phone con Azure

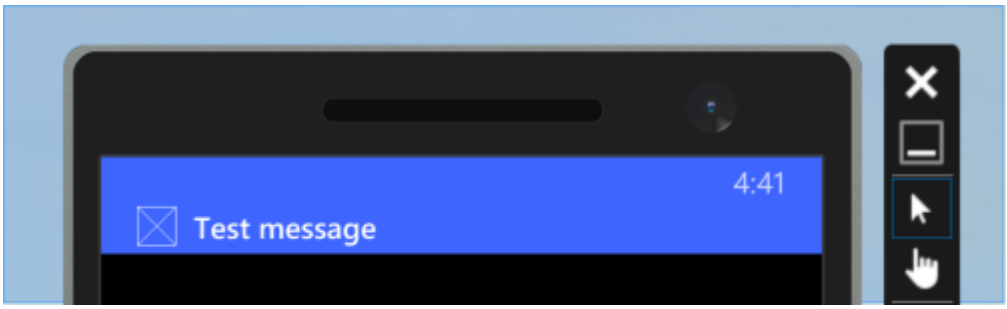
Su Windows Phone è necessario implementare qualcosa come il codice sottostante per iniziare a lavorare con le notifiche push. Questo può essere trovato nel file `App.xaml.cs`

```
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
}
```

Inoltre, non dimenticare di abilitare le funzionalità nel file `Package.appxmanifest` .



Una notifica push di esempio può apparire come questa:



Leggi Le notifiche push online: https://riptutorial.com/it/xamarin-forms/topic/5042/le-notifiche-push

Capitolo 25: Le notifiche push

Osservazioni

Lingo di notifica semplice AWS:

Endpoint : l'endpoint può essere un telefono, un indirizzo e-mail o qualsiasi altra cosa, è ciò che AWS SNS può rispondere con una notifica

Argomento - Essenzialmente un gruppo che contiene tutti i tuoi endpoint

Abbonati : registrati al tuo telefono / cliente per ricevere notifiche

Lingo di notifica push generico:

APNS - Servizio di notifica push Apple. Apple è l'unica in grado di inviare notifiche push. Questo è il motivo per cui forniamo la nostra app con il certificato appropriato. Forniamo a AWS SNS il certificato che Apple ci fornisce per autorizzare SNS a inviare una notifica a APNS per nostro conto.

GCM - Google Cloud Messaging è molto simile a APNS. Google è l'unico che può inviare direttamente notifiche push. Pertanto, per prima cosa registriamo la nostra app in GCM e consegniamo il nostro token ad AWS SNS. SNS gestisce tutte le cose complesse che riguardano GCM e inviano i dati.

Examples

Esempio di iOS

1. Avrai bisogno di un dispositivo di sviluppo
2. Vai al tuo account sviluppatore Apple e crea un profilo di provisioning con le notifiche push abilitate
3. Avrai bisogno di una sorta di modo per notificare il tuo telefono (AWS, Azure..etc) **Useremo AWS qui**

```
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"

            });

        }

    }
```

Leggi Le notifiche push online: https://riptutorial.com/it/xamarin-forms/topic/5998/le-notifiche-push

Capitolo 26: MessagingCenter

introduzione

Xamarin.Forms ha un meccanismo di messaggistica integrato per promuovere il codice disaccoppiato. In questo modo, non è necessario conoscere i modelli e gli altri componenti. Possono comunicare con un semplice contratto di messaggistica.

Ci sono fondamentalmente due ingredienti principali per l'utilizzo di `MessagingCenter` .

Iscriviti ; ascoltare i messaggi con una certa firma (il contratto) ed eseguire il codice quando viene ricevuto un messaggio. Un messaggio può avere più abbonati.

Invia ; inviare un messaggio per gli abbonati su cui agire.

Examples

Semplice esempio

Qui vedremo un semplice esempio di utilizzo di MessagingCenter in Xamarin.Forms.

Innanzitutto, diamo un'occhiata all'iscrizione a un messaggio. Nel modello `FooMessaging` sottoscriviamo un messaggio proveniente da `MainPage` . Il messaggio dovrebbe essere "Ciao" e quando lo riceveremo, registriamo un gestore che imposta la proprietà `Greeting` . Infine, `this` significa che l'istanza attuale di `FooMessaging` sta registrando per questo messaggio.

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

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

Per inviare un messaggio che attiva questa funzionalità, dobbiamo avere una pagina chiamata `MainPage` e implementare il codice come sotto.

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

Nella nostra `MainPage` abbiamo un pulsante con un gestore che invia un messaggio. `this` dovrebbe essere un'istanza di `MainPage` .

Passando argomenti

È anche possibile passare argomenti con un messaggio con cui lavorare.

Useremo le classi del nostro esempio precedente e le estenderemo. Nella parte ricevente, subito dietro la chiamata al metodo `Subscribe` , aggiungi il tipo di argomento che ti aspetti. Assicurati inoltre di dichiarare gli argomenti nella firma del gestore.

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

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

Quando invii un messaggio, assicurati di includere il valore dell'argomento. Inoltre, qui aggiungi il tipo subito dopo il metodo `Send` e aggiungi il valore dell'argomento.

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

In questo esempio viene utilizzata una stringa semplice, ma è anche possibile utilizzare qualsiasi altro tipo di oggetti (complessi).

Cancellazione, cancellami

Quando non hai più bisogno di ricevere messaggi, puoi semplicemente annullare l'iscrizione. Puoi farlo in questo modo:

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

Quando stai fornendo argomenti, devi annullare l'iscrizione alla firma completa, come questa:

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

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

Capitolo 27: Navigazione in Xamarin.Forms

Examples

Flusso NavigationPage

```
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;
```

```
        }
    }
}
```

Flusso NavigationPage con XAML

File App.xaml.cs (il file App.xaml è predefinito, quindi saltato)

```
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;
        }
    }
}
```

File 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>
```

In alcuni casi è necessario aprire la nuova pagina non nella navigazione corrente ma in quella globale. Ad esempio, se la pagina corrente contiene il menu in basso, sarà visibile quando si spinge la nuova pagina nella navigazione corrente. Se è necessario aprire la pagina sull'intero contenuto visibile nascondendo il menu in basso e il contenuto di un'altra pagina corrente, è necessario inserire la nuova pagina come modale nella navigazione globale. Vedi la proprietà

`App.GlobalNavigation` e l'esempio seguente.

File 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);
        }
    }
}
```

File SecondPage.xaml (il file xaml.cs è predefinito, quindi saltato)

```
<?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>
```

Navigazione gerarchica con XAML

Per impostazione predefinita, il modello di navigazione funziona come una pila di pagine, chiamando le pagine più recenti sulle pagine precedenti. Dovrai utilizzare l'oggetto NavigationPage per questo.

Spingendo nuove pagine

```
...
public class App : Application
{
    public App()
    {
```

```
        MainPage = new NavigationPage(new Page1());
    }
}
...
```

Page1.xaml

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

Page1.xaml.cs

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

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

Page2.xaml

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

Page2.xaml.cs

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

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

Popping pages

Normalmente l'utente utilizza il pulsante Indietro per restituire le pagine, ma a volte è necessario controllarlo a livello di codice, quindi è necessario chiamare il metodo **NavigationPage.PopAsync ()** per tornare alla pagina precedente o **NavigationPage.PopToRootAsync ()** per tornare all'inizio, come ...

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();
    }
}
...
```

Navigazione modale con XAML

Le pagine modali possono essere create in tre modi:

- Da oggetto **NavigationPage** per pagine a schermo intero

- Per avvisi e notifiche
- Per ActionSheets che sono menu a comparsa

Modifiche a schermo intero

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

Avvisi / conferme e notifiche

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

ActionSheets

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

Pagina principale di dettaglio principale

```
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;
    }
}
```

Navigazione dettagliata principale

Il codice seguente mostra come eseguire la navigazione asincrona quando l'app si trova in un contesto 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;
    }
```

Leggi Navigazione in Xamarin.Forms online: https://riptutorial.com/it/xamarin-forms/topic/1571/navigazione-in-xamarin-forms

Capitolo 28: Navigazione in Xamarin.Forms

Osservazioni

La navigazione su Xamarin.Forms si basa su due principali schemi di navigazione: gerarchico e modale.

Il modello gerarchico consente all'utente di spostarsi in una pila di pagine e tornare premendo il pulsante "indietro" / "su".

Il modello modale è una pagina di interruzione che richiede un'azione specifica da parte dell'utente, ma normalmente può essere annullata premendo il pulsante Annulla. Alcuni esempi sono notifiche, avvisi, finestre di dialogo e pagine registro / edizione.

Examples

Utilizzo di INavigation dal modello di visualizzazione

Il primo passo è creare un'interfaccia di navigazione che utilizzeremo sul modello di visualizzazione:

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

Nel metodo `Initialize` utilizzo il mio programma di mappatura personalizzato in cui tengo la raccolta dei tipi di pagine con le chiavi associate.

```
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 con le pagine:

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

File `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
}
```

In mapper ho associato il tipo di qualche pagina con valore enum.

Implementazione di `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.");
        }
    }
}
```

Ottengo il tipo di pagina su cui l'utente vuole navigare e crearne l'istanza usando la riflessione.

E poi potrei usare il servizio di navigazione sul modello di visualizzazione:

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

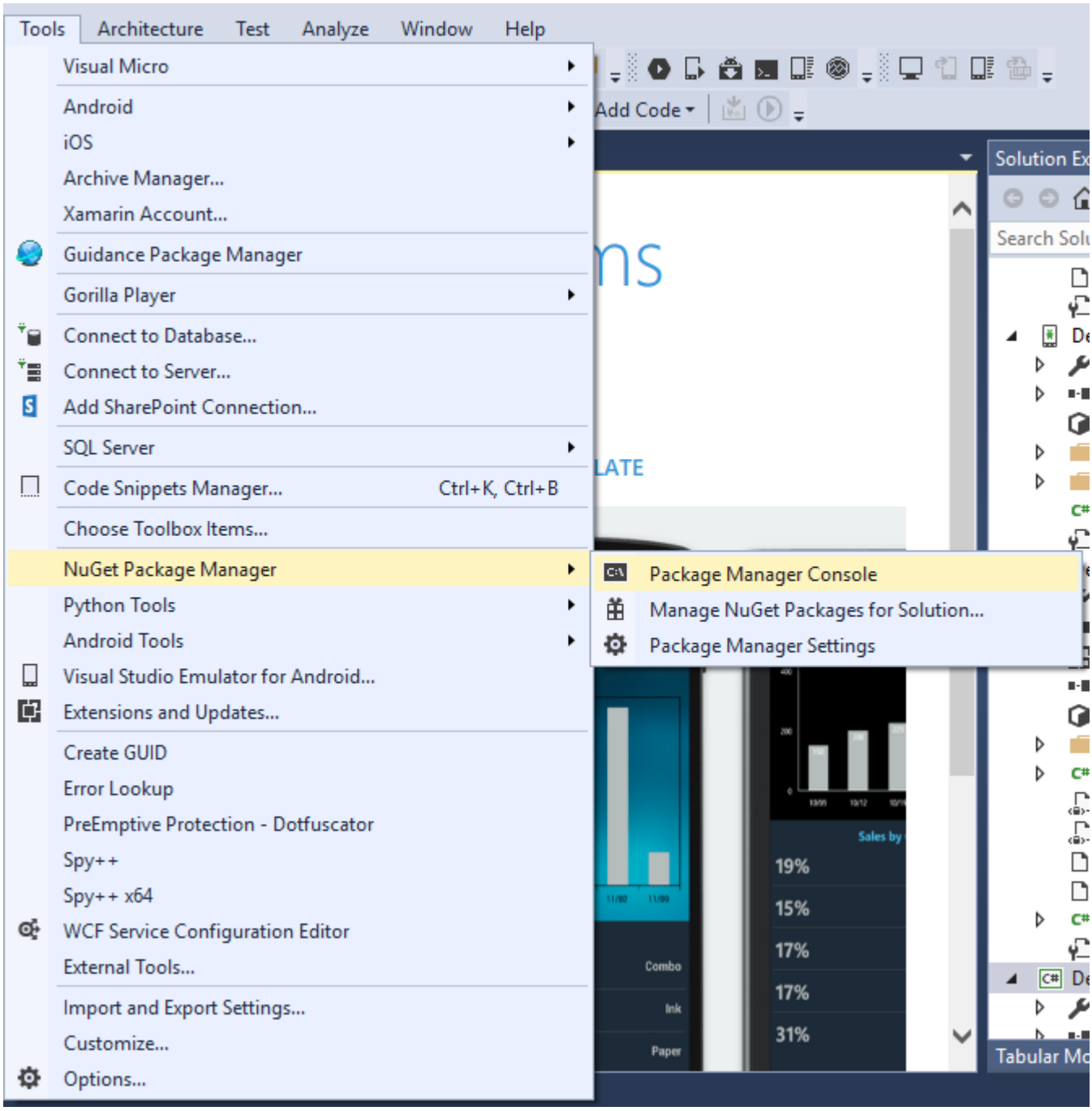
Leggi Navigazione in Xamarin.Forms online: https://riptutorial.com/it/xamarin-forms/topic/2507/navigazione-in-xamarin-forms

Capitolo 29: OAuth2

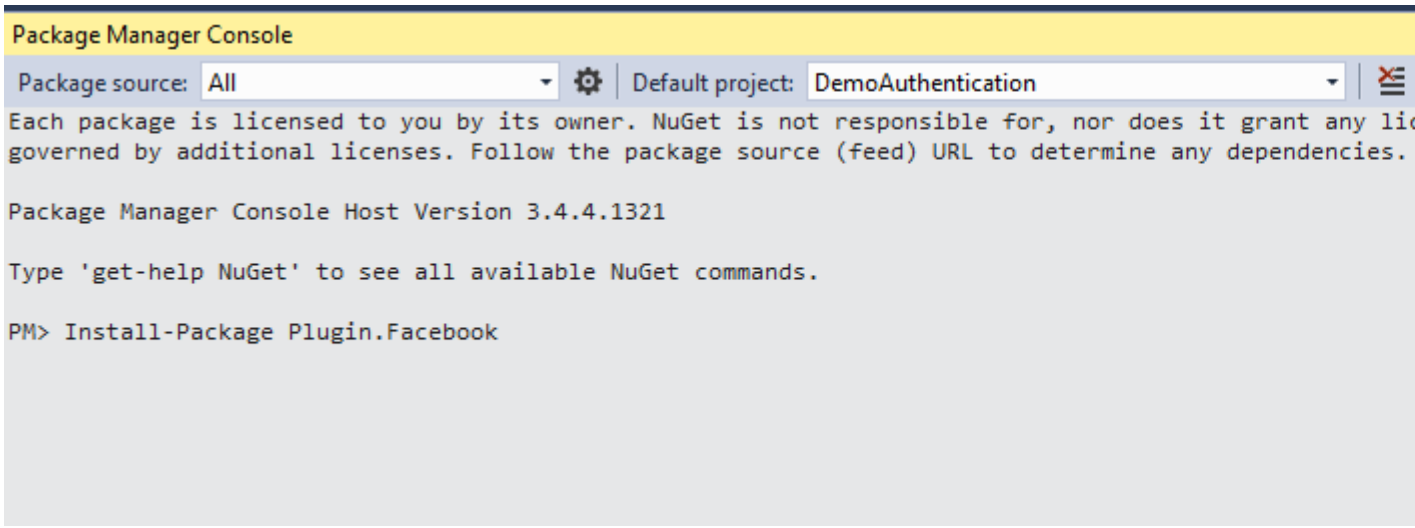
Examples

Autenticazione tramite plug-in

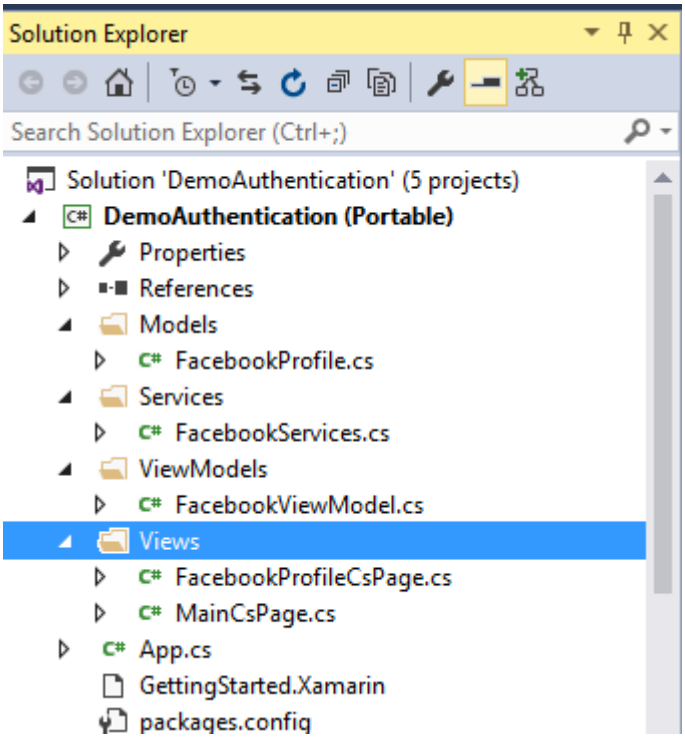
1. Per prima cosa, vai su **Strumenti** > **NuGet Package Manager** > **Console Gestione pacchetti** .



2. Immettere questo comando " **Install-Package Plugin.Facebook** " nella console di Package Manager.



3. Ora tutto il file viene creato automaticamente.



Video : accedi con Facebook in Xamarin Forms

Altre autenticazioni usando il plugin. Si prega di inserire il comando nella console di Package Manager come mostrato nel passaggio 2.

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

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

Capitolo 30: Pagina Xamarin.Forms

Examples

TabbedPage

Una TabbedPage è simile a una NavigationPage in quanto consente e gestisce la navigazione semplice tra diversi oggetti Pagina figlio. La differenza è che, in generale, ciascuna piattaforma visualizza una sorta di barra nella parte superiore o inferiore dello schermo che visualizza la maggior parte, se non tutti, degli oggetti pagina figlio disponibili. Nelle applicazioni Xamarin.Forms, TabbedPage è generalmente utile quando si dispone di un numero limitato di pagine predefinite tra gli utenti, ad esempio un menu o un semplice wizard che può essere posizionato nella parte superiore o inferiore dello schermo.

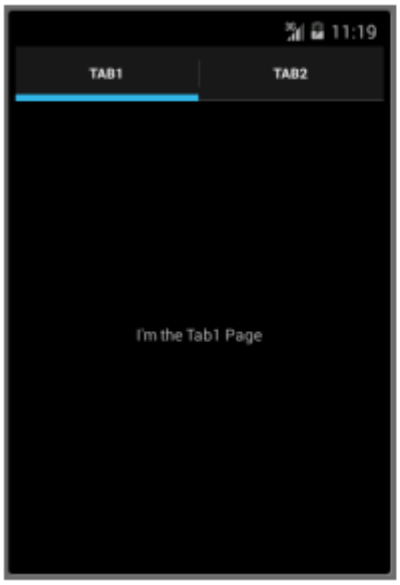
XAML

```
<?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>
```

Codice

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

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



Pagina dei contenuti

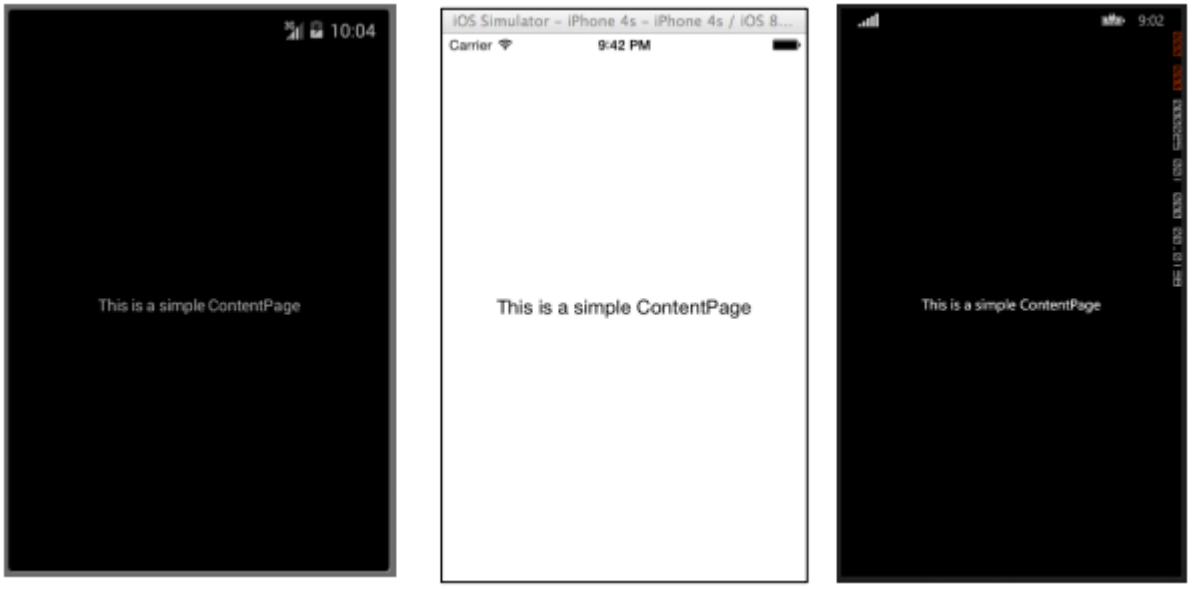
ContentPage: visualizza una singola vista.

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>
```

Codice

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



MasterDetailPage

MasterDetailPage: gestisce due pagine (riquadri) separate di informazioni.

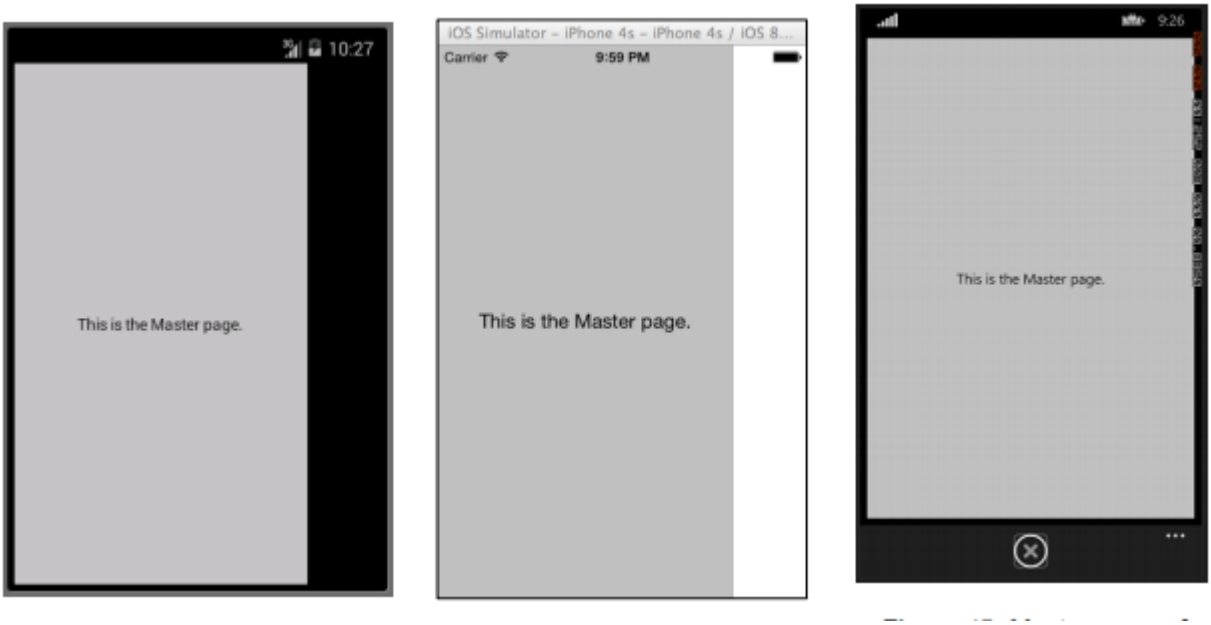
XAML

```
<?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>
```

Codice

```
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
}
}
};
```



Leggi Pagina Xamarin.Forms online: https://riptutorial.com/it/xamarin-forms/topic/7018/pagina-xamarin-forms

Capitolo 31: Perché Xamarin Forms e When to use Xamarin Forms

Osservazioni

È possibile fare riferimento alla documentazione dei moduli Xamarin ufficiale per ulteriori informazioni:

https://www.xamarin.com/forms

Examples

Perché Xamarin Forms e When to use Xamarin Forms

Xamarin sta diventando sempre più popolare: è difficile decidere quando utilizzare Xamarin.Forms e quando Xamarin.Platform (quindi Xamarin.iOS e Xamarin.Android).

Prima di tutto dovresti sapere per quale tipo di applicazioni puoi usare Xamarin.Forms:

1. Prototipi - per visualizzare come apparirà la tua applicazione sui diversi dispositivi.
2. Applicazioni che non richiedono funzionalità specifiche della piattaforma (come le API) - ma qui si ricorda che Xamarin sta lavorando attivamente per fornire la massima compatibilità multipiattaforma possibile.
3. Applicazioni in cui la condivisione del codice è fondamentale, più importante dell'interfaccia utente.
4. Applicazioni in cui i dati visualizzati sono più importanti delle funzionalità avanzate

Ci sono anche molti altri fattori:

1. Chi sarà responsabile dello sviluppo delle applicazioni: se il tuo team è composto da sviluppatori mobili esperti, sarà in grado di gestire facilmente Xamarin.Forms. Ma se hai uno sviluppatore per piattaforma (sviluppo nativo) i moduli possono essere una sfida più grande.
2. Tieni presente che con Xamarin.Forms puoi ancora riscontrare alcuni problemi a volte - la piattaforma Xamarin.Forms è ancora in fase di miglioramento.
3. Lo sviluppo rapido a volte è molto importante: per ridurre i costi e il tempo puoi decidere di utilizzare i moduli.
4. Quando si sviluppano applicazioni aziendali senza alcuna funzionalità avanzata, è meglio utilizzare Xamarin.Forms: consente di condividere il codice della modalità non evento nell'area mobile, ma in generale. Alcune porzioni di codice possono essere condivise su più piattaforme.

Non dovresti usare Xamarin.Forms quando:

1. Devi creare funzionalità personalizzate e accedere a API specifiche della piattaforma
2. Devi creare un'interfaccia utente personalizzata per l'applicazione mobile
3. Quando alcune funzionalità non sono pronte per Xamarin.Forms (come alcuni comportamenti specifici sul dispositivo mobile)
4. Il tuo team è composto da sviluppatori mobili specifici della piattaforma (sviluppo mobile in Java e / o Swift / Objective C)

Leggi Perché Xamarin Forms e When to use Xamarin Forms online: https://riptutorial.com/it/xamarin-forms/topic/6869/perche-xamarin-forms-e-when-to-use-xamarin-forms

Capitolo 32: Regolazioni visive specifiche della piattaforma

Examples

Regolazioni idioma

Le regolazioni specifiche di idioma possono essere eseguite dal codice C #, ad esempio per modificare l'orientamento del layout sia che venga visualizzata la vista sia che sia un telefono o un tablet.

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

Queste funzionalità sono anche disponibili direttamente dal codice 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>
```

Regolazioni della piattaforma

Le regolazioni possono essere fatte per piattaforme specifiche dal codice C #, ad esempio per cambiare padding per tutte le piattaforme target.

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

È anche disponibile un metodo di supporto per dichiarazioni C # abbreviate:

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

Queste funzionalità sono anche disponibili direttamente dal codice XAML:

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

Usando gli stili

Quando si lavora con XAML, l'uso di uno `Style` centralizzato consente di aggiornare una serie di viste in stile da un'unica posizione. Tutte le regolazioni di idioma e piattaforma possono anche essere integrate ai tuoi stili.

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

Utilizzo di viste personalizzate

Puoi creare visualizzazioni personalizzate che possono essere integrate nella tua pagina grazie a quegli strumenti di regolazione.

Seleziona `File > New > File... > Forms > Forms ContentView (Xaml)` e crea una vista per ogni specifico layout: `TabletHome.xaml` e `PhoneHome.xaml` .

Quindi selezionare `File > New > File... > Forms > Forms ContentPage` e creare una `HomePage.cs` che contenga:

```
using Xamarin.Forms;

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

    }
}
```

Ora hai una pagina `HomePage` che crea una gerarchia di visualizzazione diversa per gli idiomi di `Phone` e `Tablet` .

Leggi Regolazioni visive specifiche della piattaforma online: https://riptutorial.com/it/xamarin-forms/topic/5012/regolazioni-visive-specifiche-della-piattaforma

Capitolo 33: Renderizzatori personalizzati

Examples

Renderer personalizzato per ListView

I Renderizzatori personalizzati consentono agli sviluppatori di personalizzare l'aspetto e il comportamento dei controlli Xamarin.Forms su ciascuna piattaforma. Gli sviluppatori potrebbero utilizzare le funzionalità dei controlli nativi.

Ad esempio, abbiamo bisogno di disabilitare scorrere in `ListView` . Su iOS `ListView` è scorrevole anche se tutti gli elementi sono posizionati sullo schermo e l'utente non dovrebbe essere in grado di scorrere l'elenco. `Xamarin.Forms.ListView` non gestisce tale impostazione. In questo caso, un renderer sta venendo in aiuto.

In primo luogo, dovremmo creare un controllo personalizzato nel progetto PCL, che dichiarerà alcune proprietà bindable richieste:

```
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); }
    }
}
```

Il prossimo passo sarà la creazione di un renderer per ogni piattaforma.

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;
```

```
        }
    }
}
```

E Android (l'elenco di Android non ha scroll se tutti gli elementi sono posizionati sullo schermo, quindi non disabiliteremo lo scrolling, ma siamo comunque in grado di utilizzare le proprietà native):

```
[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` proprietà `Element` del renderer è il controllo `SuperListView` dal progetto PCL.

`Control` proprietà di `Control` del renderer è il controllo nativo. `Android.Widget.ListView` per Android e `UIKit.UITableView` per iOS.

E come lo useremo in `XAML` :

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


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

file `.cs` della pagina:

```
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);
    }
}
```

Renderizzatore personalizzato per BoxView

Aiuto Renderer personalizzato per consentire di aggiungere nuove proprietà e renderle in modo diverso nella piattaforma nativa che non può essere altrimenti attraverso il codice condiviso. In questo esempio aggiungeremo raggio e ombra a un boxview.

In primo luogo, dovremmo creare un controllo personalizzato nel progetto PCL, che dichiarerà alcune proprietà bindable richieste:

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

Il prossimo passo sarà la creazione di un renderer per ogni piattaforma.

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);
        }
    }

 }
}
```

Anche in questo caso è possibile personalizzare comunque come si desidera all'interno del metodo di disegno.

E lo stesso per 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);
        }
}
```

}

Lo XAML:

Per prima cosa facciamo riferimento al nostro controllo con lo spazio dei nomi definito in precedenza.

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

Quindi usiamo il controllo come segue e usiamo le proprietà definite all'inizio:

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

Accesso al renderer da un progetto nativo

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

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

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

Etichetta arrotondata con un renderizzatore personalizzato per Frame (parti PCL e iOS)

Primo passo: parte PCL

```
using Xamarin.Forms;

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

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

Secondo passo: parte 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;
            }
        }
    }
}
```

Terzo passo: codice XAML per chiamare un ExtendedFrame

Se vuoi usarlo in una parte XAML, non dimenticare di scrivere questo:

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

dopo

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

Ora puoi usare ExtendedFrame in questo modo:

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

BoxView arrotondato con colore di sfondo selezionabile

Primo passo: parte PCL

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

    public double CornerRadius
    {
        get
        {
            return (double)GetValue(CornerRadiusProperty);
        }
        set
        {
            SetValue(CornerRadiusProperty, value);
        }
    }

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

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

Secondo passo: parte 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);
        }
    }
}
```

Terzo passo: parte 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();
            }
        }
    }
}
```

Leggi Renderizzatori personalizzati online: https://riptutorial.com/it/xamarin-forms/topic/2949/renderizzatori-personalizzati

Capitolo 34: Selettore contatti - Form Xamarin (Android e iOS)

Osservazioni

Contatta Picker XF (Android e 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);
        }
    }
}
```

Leggi Selettore contatti - Form Xamarin (Android e iOS) online: https://riptutorial.com/it/xamarin-forms/topic/6659/selettore-contatti---form-xamarin--android-e-ios-

Capitolo 35: Servizi di dipendenza

Osservazioni

Accedi a API specifiche della piattaforma da PCL o Progetto condiviso.

Examples

Accedi a Fotocamera e Galleria.

(Codice di accesso) [https://github.com/vDoers/vDoersCameraAccess]

Leggi Servizi di dipendenza online: https://riptutorial.com/it/xamarin-forms/topic/6127/servizi-di-dipendenza

Capitolo 36: Test dell'unità BDD in Xamarin.Forms

Osservazioni

- Il contenitore / resolver DI che utilizziamo internamente in questa libreria è Autofac.
- Il framework di test è NUnit 3x.
- Dovresti essere in grado di utilizzare questa libreria con qualsiasi framework Xamarin.Forms
- Fonte e progetto di esempio disponibili qui

Examples

Simple Specflow per testare i comandi e la navigazione con NUnit Test Runner

perché ne abbiamo bisogno?

Il modo attuale di eseguire test unitari in Xamarin.Forms è tramite un runner di piattaforma, quindi il test dovrà essere eseguito all'interno di un ambiente ios, android, windows o mac UI: esecuzione dei test nell'IDE

Xamarin fornisce anche fantastici test dell'interfaccia utente con l'offerta Xamarin.TestCloud , ma quando si desidera implementare le pratiche di dev di BDD e avere la possibilità di testare ViewModels e Comandi, mentre si esegue a basso costo su un dispositivo di test di unità localmente o su un server di build, non c'è costruito in modo

Ho sviluppato una libreria che consente di utilizzare Specflow con Xamarin.Forms per implementare facilmente le funzionalità dalle definizioni Scenari fino a ViewModel, indipendentemente da qualsiasi framework MVVM utilizzato per l'app (come XLabs , MVVMCross , Prism)

Se sei nuovo a BDD, controlla l' uscita Specflow .

Uso:

- Se non lo hai ancora, installa l'estensione dello studio visivo specflow da qui (o dal tuo IDE di Visual Studio): https://visualstudiogallery.msdn.microsoft.com/c74211e7-cb6e-4dfa-855d-df0ad4a37dd6
- Aggiungi una libreria di classi al tuo progetto Xamarin.Forms. Questo è il tuo progetto di test.
- Aggiungi il pacchetto SpecFlow.Xamarin.Forms da nuget ai tuoi progetti di test.

- Aggiungi una classe al tuo progetto di test che eredita "TestApp" e registra le coppie view / modelmodels e aggiungi qualsiasi registrazione DI, come di seguito:

```
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();
    }
}
```

- Aggiungi una classe SetupHook al tuo progetto di test, al fine di aggiungere ganci Specflow. Dovrai eseguire il bootstrap dell'applicazione di test come di seguito, fornendo la classe che hai creato sopra e il tuo viewmodel iniziale dell'app:

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

- Sarà necessario aggiungere un blocco catch al codice code di xamarin.forms per ignorare il framework xamarin.forms che ti obbliga a eseguire l'interfaccia utente ui (qualcosa che non vogliamo fare):

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

- Aggiungi una funzione specflow al tuo progetto (usando i modelli vs specflow forniti con l'estensione vs specflow)
- Crea / Genera una classe di passaggi che eredita TestStepBase, passando il parametro scenarioContext alla base.
- Utilizza i servizi di navigazione e gli helper per navigare, eseguire comandi e testare i tuoi modelli di visualizzazione:

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

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


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

        }

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

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

Uso avanzato per MVVM

Per aggiungere al primo esempio, per testare le istruzioni di navigazione che si verificano all'interno dell'applicazione, è necessario fornire ViewModel con un gancio per la navigazione. Per realizzare questo:

- Aggiungi il pacchetto SpecFlow.Xamarin.Forms.IViewModel da nuget al tuo progetto PCL Xamarin.Forms
- Implementare l'interfaccia IViewModel nel ViewModel. Questo esporrà semplicemente la proprietà INavigation di Xamarin.Form:
- `public class MainViewModel : INotifyPropertyChanged, IViewModel.IViewModel { public INavigation Navigation { get; set; }`
- Il framework di test lo individuerà e gestirà la navigazione interna
- È possibile utilizzare qualsiasi framework MVVM per l'applicazione (come XLabs ,

MVVMCross , Prism per citarne alcuni) finché l'interfaccia IViewModel viene implementata nel ViewModel, il framework lo raccoglierà.

Leggi Test dell'unità BDD in Xamarin.Forms online: https://riptutorial.com/it/xamarin-forms/topic/6172/test-dell-unita-bdd-in-xamarin-forms

Capitolo 37: Test unitario

Examples

Test dei modelli di visualizzazione

Prima di iniziare ...

In termini di livelli applicativi, ViewModel è una classe contenente tutte le regole e le regole aziendali che fanno sì che l'app esegua ciò che dovrebbe in base ai requisiti. È anche importante renderlo il più indipendente possibile riducendo i riferimenti all'interfaccia utente, al livello dati, alle funzioni native e alle chiamate API ecc. Tutto ciò rende la tua VM testabile.
In breve, il tuo ViewModel:

- Non dovrebbe dipendere da classi di UI (viste, pagine, stili, eventi);
- Non usare dati statici di altre classi (il più possibile);
- Dovrebbe implementare la logica di business e preparare i dati da inserire nell'interfaccia utente;
- Dovrebbe utilizzare altri componenti (database, HTTP, specifici dell'interfaccia utente) tramite le interfacce che vengono risolte utilizzando Iniezione delle dipendenze.

  Il tuo ViewModel potrebbe avere anche le proprietà di altri tipi di VM. Ad esempio `ContactsPageViewModel` avrà propery del tipo di raccolta come `ObservableCollection<ContactListItemViewModel>`

Requisiti aziendali

Diciamo che abbiamo la seguente funzionalità da implementare:

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

Dopo aver chiarito la user story, abbiamo definito i seguenti scenari:

```
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
```

Resteremo solo con questi due scenari. Ovviamente, ci dovrebbero essere molti più casi e dovresti definirli tutti prima della vera codifica, ma è abbastanza per noi ora avere familiarità con i test unitari dei modelli di vista.

Seguiamo il classico approccio TDD e iniziamo con la scrittura di una classe vuota sottoposta a test. Quindi scriveremo dei test e li renderemo verdi implementando la funzionalità aziendale.

Classi comuni

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

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

Servizi

Ti ricordi che il nostro modello di visualizzazione non deve utilizzare direttamente le classi UI e HTTP? Dovresti invece definirli come astrazioni e non dipendere dai dettagli di implementazione .

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

Costruire lo stub ViewModel

Ok, avremo la classe di pagina per la schermata di accesso, ma iniziamo con ViewModel prima:

```
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);
    }
}
```

Abbiamo definito due proprietà `string` e un comando da associare all'interfaccia utente. Non descriveremo come costruire una classe di pagina, markup XAML e associare ViewModel a questo argomento in quanto non hanno nulla di specifico.

Come creare un'istanza LoginPageViewModel?

Penso che probabilmente stavi creando le macchine virtuali solo con il costruttore. Ora come puoi vedere la nostra VM dipende da 2 servizi che vengono iniettati come parametri del costruttore quindi non puoi fare solo `var viewModel = new LoginPageViewModel()` . Se non hai familiarità con Dependency Injection è il momento migliore per conoscerlo. Un corretto test unitario è impossibile senza conoscere e seguire questo principio.

test

Ora scriviamo alcuni test in base ai casi d'uso sopra elencati. Prima di tutto è necessario creare un nuovo assembly (solo una libreria di classi o selezionare un progetto di test speciale se si desidera utilizzare gli strumenti di test dell'unità Microsoft). `ProjectName.Tests` nome a `ProjectName.Tests` e aggiungi un riferimento al tuo progetto PCL originale.

In questo esempio userò NUnit e Moq ma puoi continuare con qualsiasi lib di prova a tua scelta. Non ci sarà nulla di speciale con loro.

Ok, questa è la classe di test:

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

Test di scrittura

Ecco i metodi di prova per i primi due scenari. Prova a mantenere 1 metodo di prova per 1 risultato previsto e non a controllare tutto in un test. Ciò ti aiuterà a ricevere rapporti più chiari su ciò che è fallito nel codice.

```
[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;
    }
}
```

E qui andiamo:

Ora l'obiettivo è scrivere un'implementazione corretta per il metodo di `Login` di ViewModel e il gioco è fatto.

Implementazione della logica aziendale

```
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);
    }
}
```

E dopo aver eseguito nuovamente i test:

Ora puoi continuare a coprire il tuo codice con nuovi test rendendolo più stabile e sicuro per la regressione.

Leggi Test unitario online: https://riptutorial.com/it/xamarin-forms/topic/3529/test-unitario

Capitolo 38: Trigger & Behaviors

Examples

Esempio di trigger di Xamarin Form

`Trigger` sono un modo semplice per aggiungere un po 'di reattività UX alla tua applicazione. Un modo semplice per farlo è aggiungere un `Trigger` che modifichi il `TextColor` `Label` basandosi sul fatto che la relativa `Entry` relativa abbia inserito o meno testo.

L'utilizzo di un `Trigger` per questo consente a `Label.TextColor` di passare dal grigio (quando non viene inserito alcun testo) al nero (non appena gli utenti immettono il testo):

Convertitore (ad ogni convertitore viene assegnata una variabile di `Instance` utilizzata nel bind in modo che non venga creata una nuova istanza della classe ogni volta che viene utilizzata):

```
/// <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 (il codice XAML usa `x:Name` della `Entry` per capire nella proprietà `Entry.Text` è di oltre 3 caratteri.):

```
<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 trigger

MultiTrigger non è necessario frequentemente, ma ci sono alcune situazioni in cui è molto utile. MultiTrigger si comporta in modo simile a Trigger o DataTrigger ma ha più condizioni. Tutte le condizioni devono essere vere per un Setter di sparare. Qui c'è un semplice esempio:

```
<!-- 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'esempio ha due voci diverse, telefono ed e-mail, e una di queste è richiesta per essere riempita. Il MultiTrigger disabilita il pulsante di invio quando entrambi i campi sono vuoti.

Leggi Trigger & Behaviors online: https://riptutorial.com/it/xamarin-forms/topic/6012/trigger--amp--behaviors

Capitolo 39: Utilizzando ListViews

introduzione

Questa documentazione spiega come utilizzare i diversi componenti di Xamarin Forms ListView

Examples

Pull to Refresh in XAML e Code behind

Per abilitare Pull to Refresh in un `ListView` in Xamarin, è necessario prima specificare che sia attivato `PullToRefresh` e quindi specificare il nome del comando che si desidera richiamare sul `ListView` viene estratto:

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

Lo stesso può essere ottenuto nel codice sottostante:

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

Quindi, devi specificare cosa fa il comando di `Refresh` nel tuo codice:

```
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;
    }
}
```

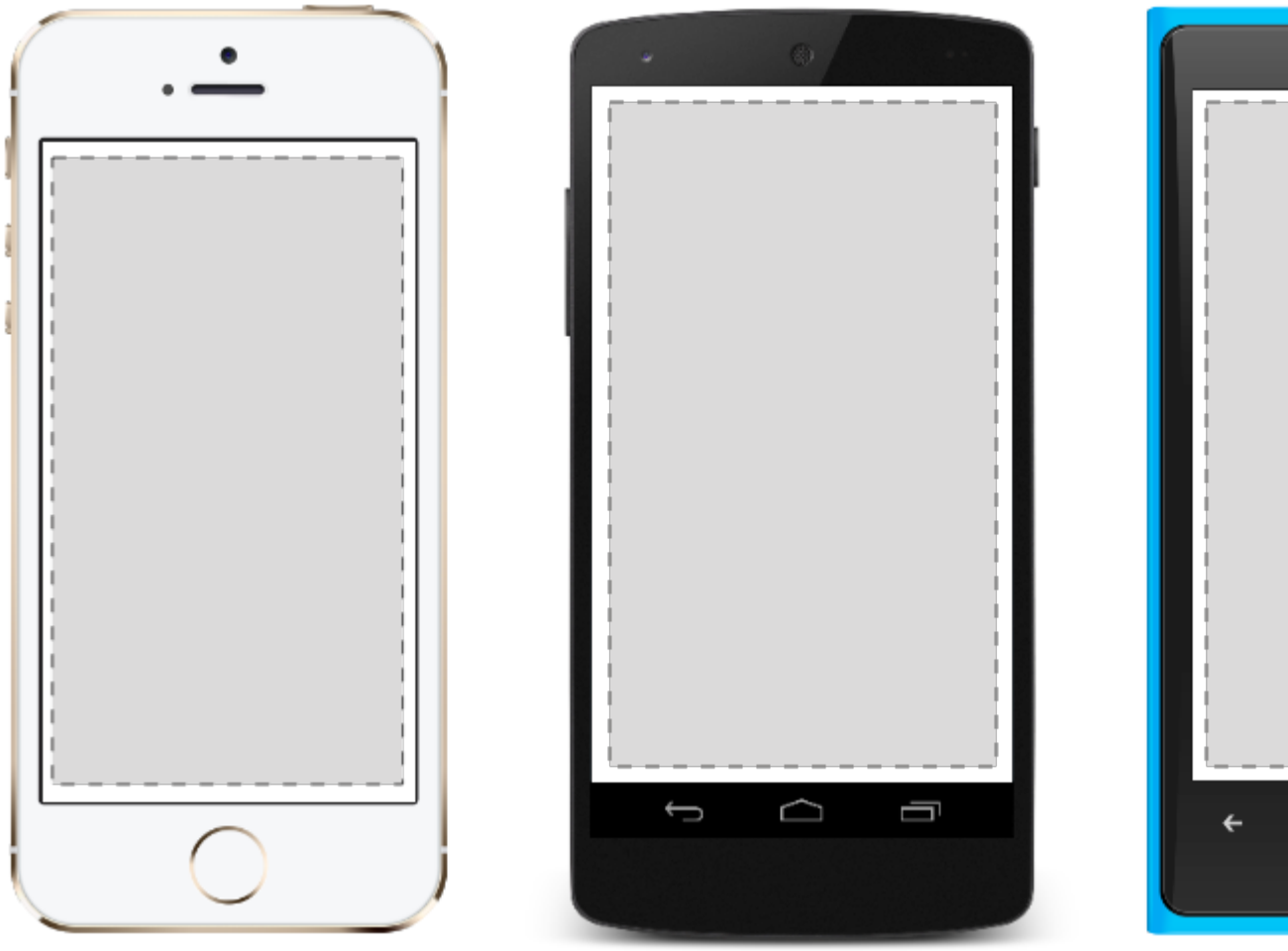
Leggi Utilizzando ListViews online: https://riptutorial.com/it/xamarin-forms/topic/9487/utilizzando-listviews

Capitolo 40: Xamarin Forms Layouts

Examples

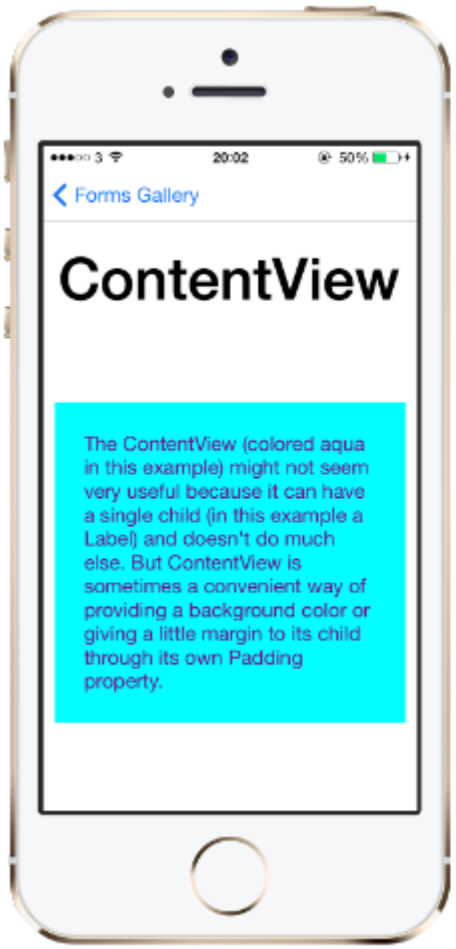
ContentPresenter

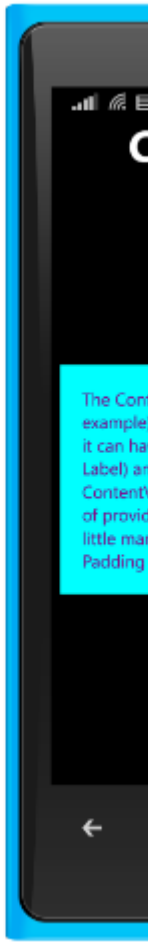
Un gestore di layout per le viste basate su modelli. Utilizzato all'interno di un oggetto ControlTemplate per indicare dove apparirà il contenuto da presentare.



contentView

Un elemento con un singolo contenuto. ContentView ha un uso molto limitato. Il suo scopo è quello di fungere da classe base per le viste composte definite dall'utente.





XAML

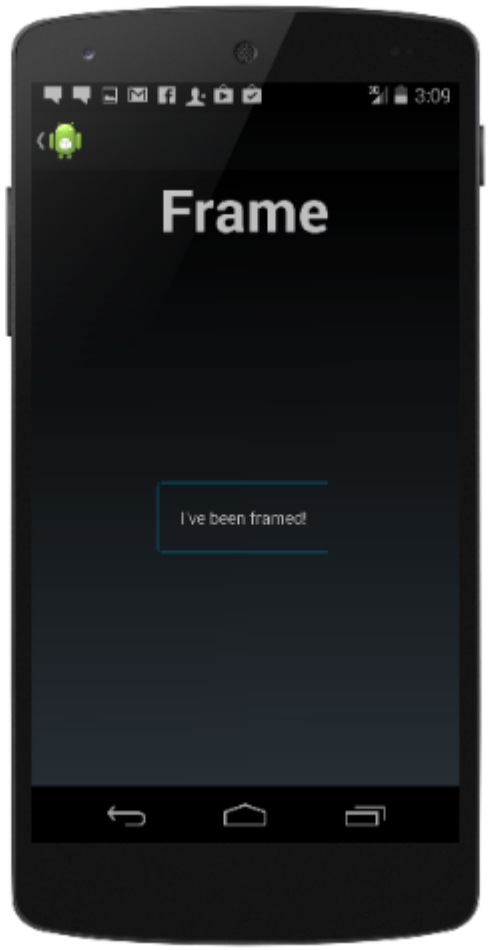
```
<ContentView>
<Label Text="Hi, I'm a simple Label inside of a simple ContentView"
HorizontalOptions="Center"
VerticalOptions="Center"/>
</ContentView>
```

Codice

```
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
}
};
```

Telaio

Un elemento contenente un singolo figlio, con alcune opzioni di framing. Frame ha un predefinito Xamarin.Forms.Layout.Padding di 20.



XAML

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

Codice

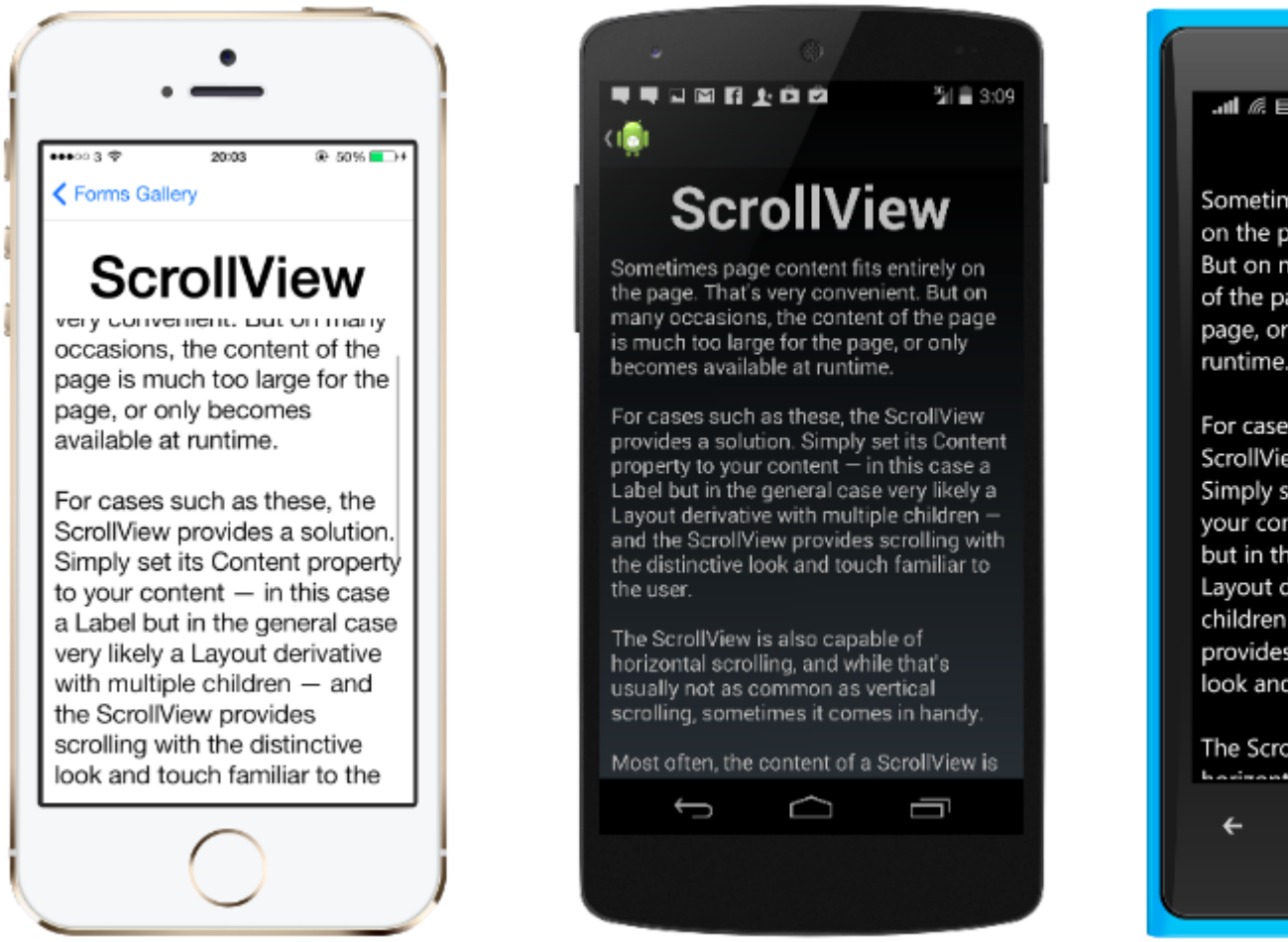
```
var frameView = new Frame {
Content = new Label {
 Text = "I've been framed!",
HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center
},
OutlineColor = Color.Red
};
```

ScrollView

Un elemento in grado di scorrere se richiesto da `Content` .

`ScrollView` contiene layout e consente loro di scorrere fuori dallo schermo. `ScrollView` viene anche utilizzato per consentire alle viste di spostarsi automaticamente sulla porzione visibile dello

schermo quando viene visualizzata la tastiera.



Nota: `ScrollViews` non deve essere annidato. Inoltre, `ScrollViews` non deve essere annidato con altri controlli che forniscono lo scorrimento, come `ListView` e `WebView` .

Un `ScrollView` è facile da definire. In XAML:

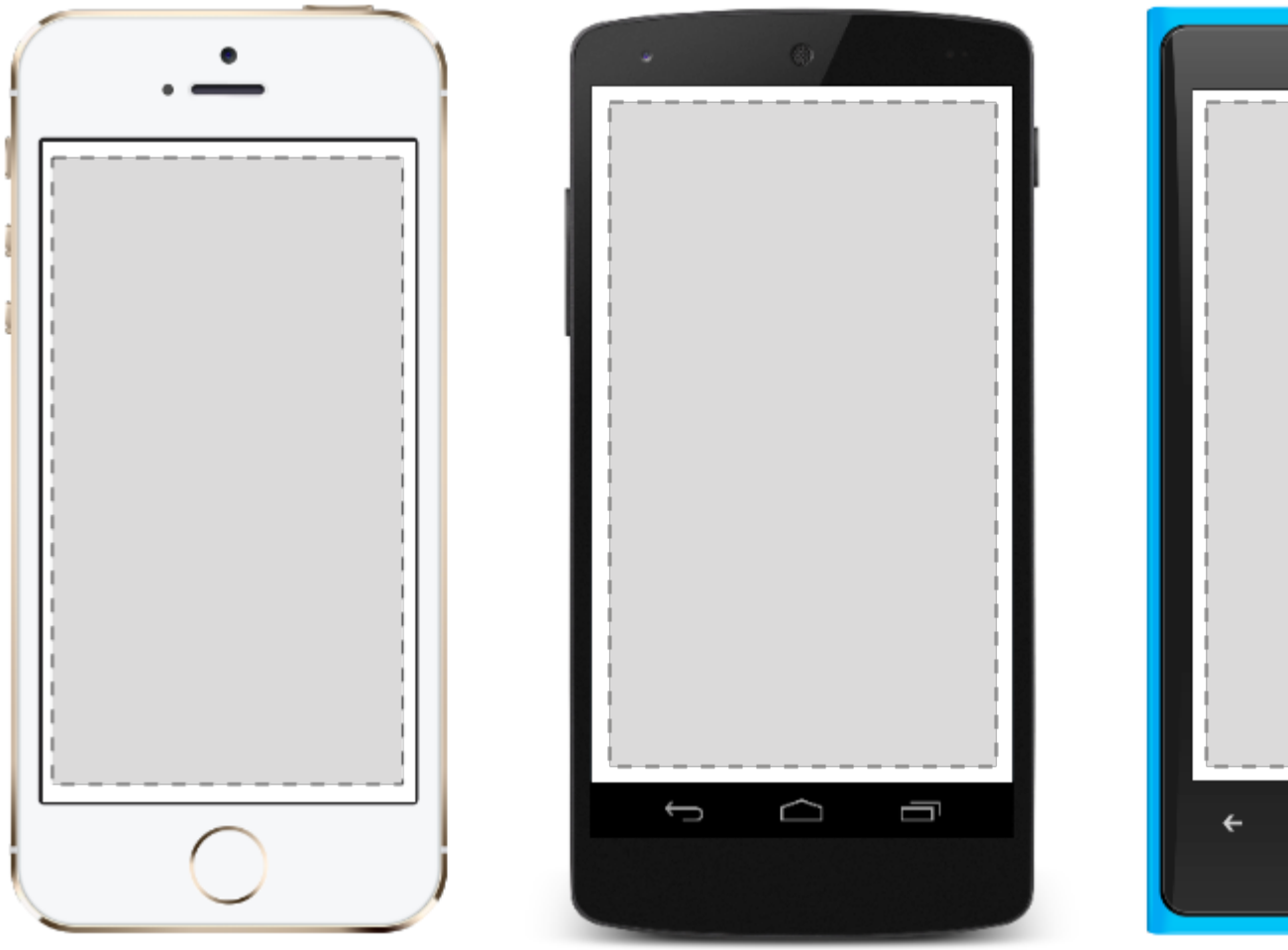
```
<ContentPage.Content>
    <ScrollView>
        <StackLayout>
            <BoxView BackgroundColor="Red" HeightRequest="600" WidthRequest="150" />
            <Entry />
        </StackLayout>
    </ScrollView>
</ContentPage.Content>
```

La stessa definizione nel codice:

```
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());
```

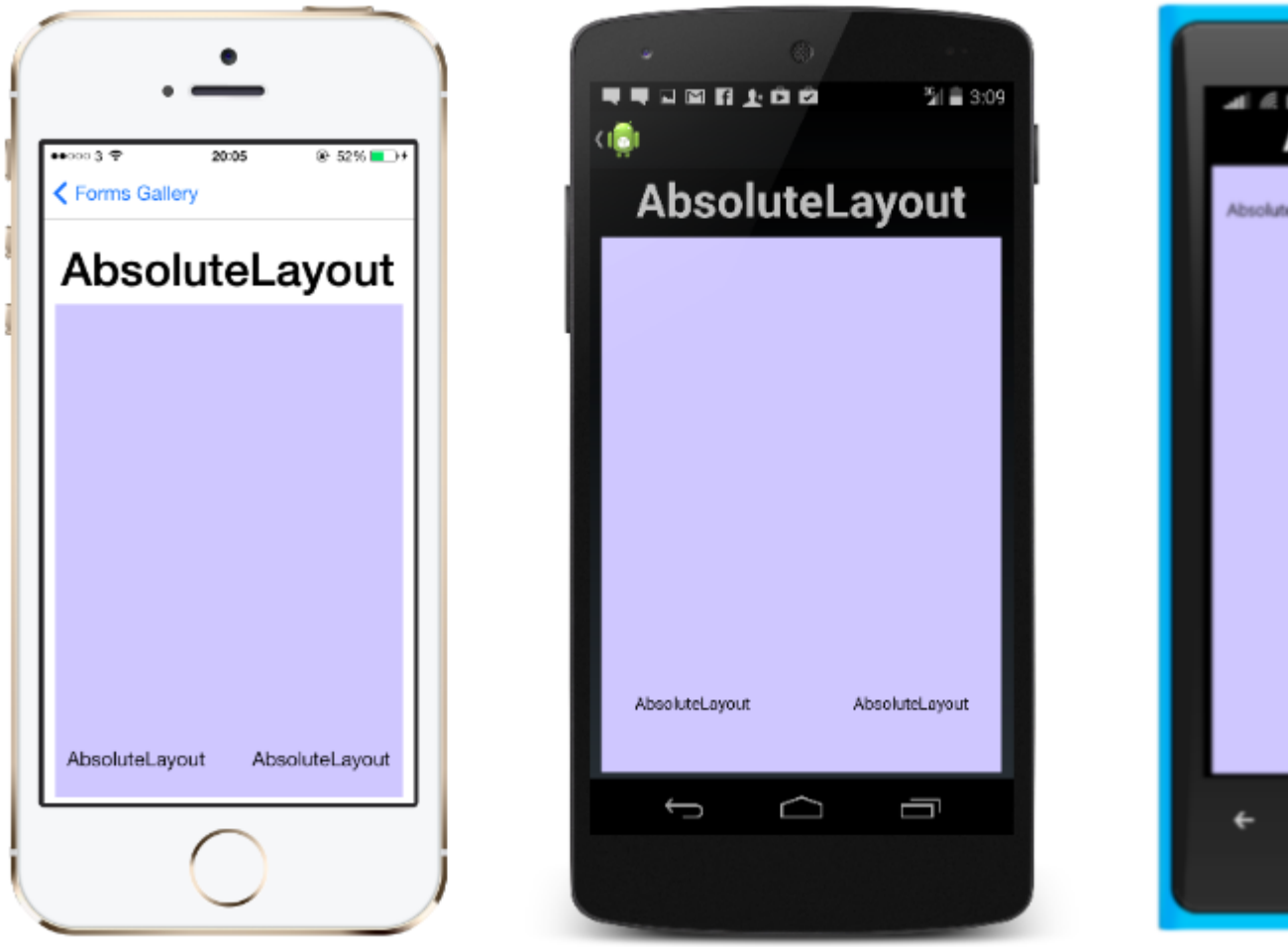
TemplatedView

Un elemento che visualizza il contenuto con un modello di controllo e la classe base per `ContentView` .



AbsoluteLayout

`AbsoluteLayout` posiziona e dimensiona elementi figlio proporzionali alla propria dimensione e posizione o valori assoluti. Le viste figlio possono essere posizionate e ridimensionate utilizzando valori proporzionali o valori statici e possono essere mischiati valori proporzionali e statici.



Una definizione di `AbsoluteLayout` in XAML si presenta così:

```
<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>
```

Lo stesso layout sarebbe simile a questo nel codice:

```
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);
```

Il controllo AbsoluteLayout in Xamarin.Forms ti consente di specificare dove esattamente sullo schermo vuoi che vengano visualizzati gli elementi figli, nonché le loro dimensioni e forma (limiti).

Esistono diversi modi per impostare i limiti degli elementi figlio in base all'enumerazione AbsoluteLayoutFlags utilizzata durante questo processo. L'enumerazione **AbsoluteLayoutFlags** contiene i seguenti valori:

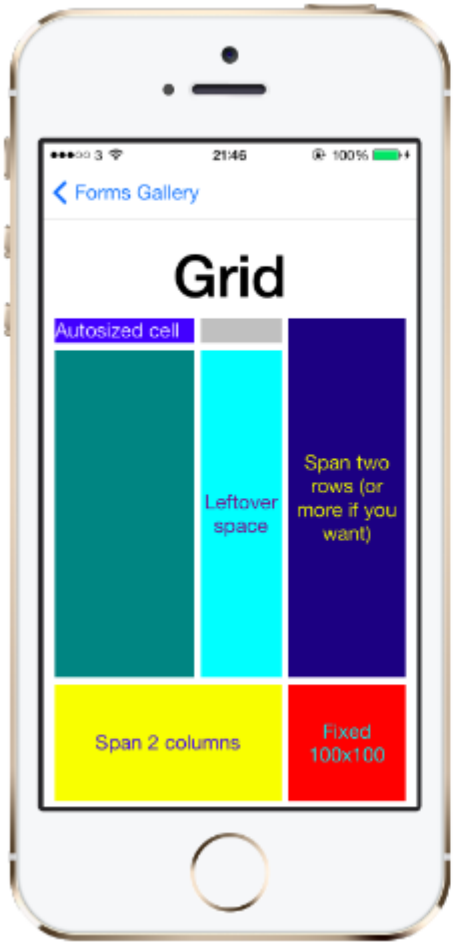
- **Tutti** : tutte le dimensioni sono proporzionali.
- **HeightProportional** : l'altezza è proporzionale al layout.
- **Nessuno** : nessuna interpretazione è stata eseguita.
- **PositionProportional** : combina XProportional e YProportional.
- **SizeProportional** : Combina WidthProportional e HeightProportional.
- **WidthProportional:** larghezza è proporzionale al layout.
- **XProporzionale** : la proprietà X è proporzionale al layout.
- **YProporzionale** : la proprietà Y è proporzionale al layout.

Il processo di lavorare con il layout del contenitore AbsoluteLayout può sembrare inizialmente poco intuitivo, ma con un po 'di uso diventerà familiare. Una volta creati gli elementi figlio, per impostarli in una posizione assoluta all'interno del contenitore, dovrai seguire tre passaggi. Dovrai impostare i flag assegnati agli elementi usando il metodo **AbsoluteLayout.SetLayoutFlags ()** . Dovrai anche usare il metodo **AbsoluteLayout.SetLayoutBounds ()** per dare agli elementi i loro limiti. Infine, dovrai aggiungere gli elementi figlio alla collezione Bambini. Poiché Xamarin.Forms è un livello di astrazione tra Xamarin e le implementazioni specifiche del dispositivo, i valori

posizionali possono essere indipendenti dai pixel del dispositivo. È qui che entrano in gioco le bandiere di layout menzionate in precedenza. È possibile scegliere in che modo il processo di layout dei controlli Xamarin.Forms deve interpretare i valori definiti.

Griglia

Un layout contenente viste disposte in righe e colonne.



Questa è una tipica definizione di `Grid` in XAML.

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="2*" />
    <RowDefinition Height="*" />
    <RowDefinition Height="200" />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="*" />
  </Grid.ColumnDefinitions>

  <ContentView Grid.Row="0" Grid.Column="0"/>
  <ContentView Grid.Row="1" Grid.Column="0"/>
  <ContentView Grid.Row="2" Grid.Column="0"/>

  <ContentView Grid.Row="0" Grid.Column="1"/>
```

```
  <ContentView Grid.Row="1" Grid.Column="1"/>
  <ContentView Grid.Row="2" Grid.Column="1"/>

</Grid>
```

La stessa `Grid` definita nel codice assomiglia a questo:

```
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) });
```

Per aggiungere elementi alla griglia: In `XAML` :

```
<Grid>

   <--DEFINITIONS...--!>

   <ContentView Grid.Row="0" Grid.Column="0"/>
   <ContentView Grid.Row="1" Grid.Column="0"/>
   <ContentView Grid.Row="2" Grid.Column="0"/>

   <ContentView Grid.Row="0" Grid.Column="1"/>
   <ContentView Grid.Row="1" Grid.Column="1"/>
   <ContentView Grid.Row="2" Grid.Column="1"/>

</Grid>
```

Nel codice 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);
```

Per `Height` e `Width` sono disponibili un numero di unità.

- **Auto** : ridimensiona automaticamente il contenuto nella riga o nella colonna. Specificato come GridUnitType.Auto in C # o Auto in XAML.
- **Proporzionale** : misura le colonne e le righe come proporzione dello spazio rimanente. Specificato come valore e GridUnitType.Star in C # e come # * in XAML, dove # rappresenta il valore desiderato. Specificare una riga / colonna con * lo farà riempire lo spazio disponibile.
- **Assoluto** : **consente di** dimensionare colonne e righe con valori specifici di altezza e larghezza fissi. Specificato come valore e GridUnitType.Absolute in C # e come # in XAML, con # come valore desiderato.

Nota: i valori di larghezza per le colonne sono impostati automaticamente su Xamarin.Forms, il che significa che la larghezza è determinata dalla dimensione dei bambini. Si noti che questo differisce dall'implementazione di XAML su piattaforme Microsoft, dove la larghezza predefinita è *, che riempirà lo spazio disponibile.

RelativeLayout

Un `Layout` che utilizza i `Constraints` per disporre i propri figli.

`RelativeLayout` viene utilizzato per posizionare e dimensionare le viste relative alle proprietà delle viste di layout o di pari livello. A differenza di `AbsoluteLayout` , `RelativeLayout` non ha il concetto dell'ancora mobile e non ha le strutture per posizionare gli elementi relativi ai bordi inferiore o destro del layout. `RelativeLayout` supporta elementi di posizionamento al di fuori dei propri limiti.

Un RelativeLayout in XAML, è come questo:

```
<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"
```

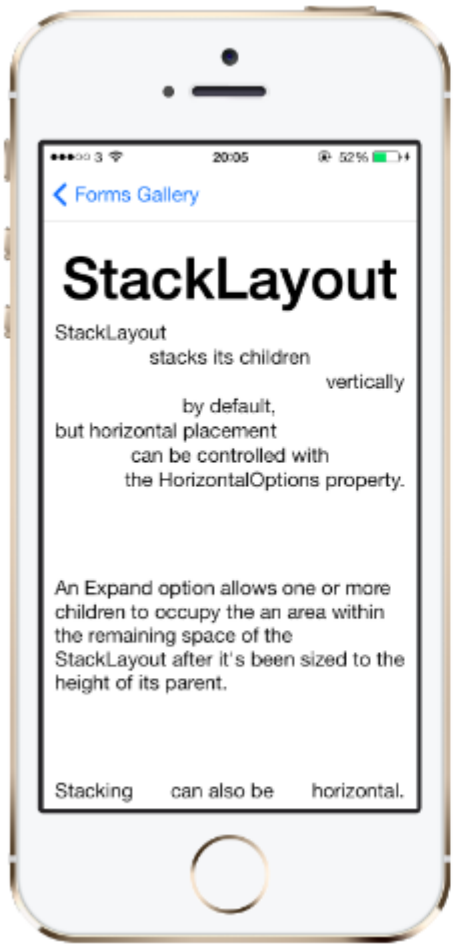
```
        RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToView,
            ElementName=redBox,Property=Y,Factor=1,Constant=20}"
        RelativeLayout.XConstraint="{ConstraintExpression Type=RelativeToView,
            ElementName=redBox,Property=X,Factor=1,Constant=20}"
        RelativeLayout.WidthConstraint="{ConstraintExpression
            Type=RelativeToParent,Property=Width,Factor=.5,Constant=0}"
        RelativeLayout.HeightConstraint="{ConstraintExpression
            Type=RelativeToParent,Property=Height,Factor=.5,Constant=0}" />
</RelativeLayout>
```

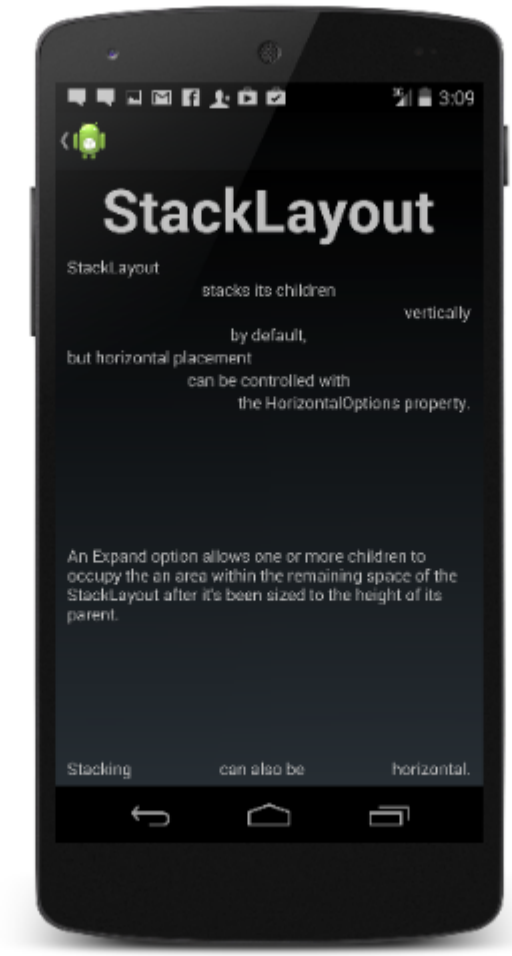
Lo stesso layout può essere realizzato con questo codice:

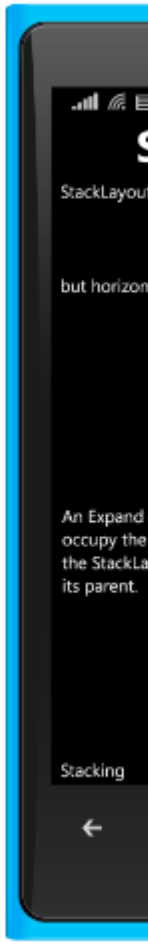
```
layout.Children.Add (redBox, Constraint.RelativeToParent ((parent) => {
        return parent.X;
    }), Constraint.RelativeToParent ((parent) => {
        return parent.Y * .15;
    }), Constraint.RelativeToParent((parent) => {
        return parent.Width;
    }), Constraint.RelativeToParent((parent) => {
        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` organizza le viste in una linea unidimensionale ("stack"), sia in orizzontale che in verticale. `Views` in uno `StackLayout` possono essere ridimensionate in base allo spazio nel layout utilizzando le opzioni di layout. Il posizionamento è determinato dalle viste dell'ordine aggiunte al layout e dalle opzioni di layout delle viste.







Utilizzo in XAML

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

Utilizzo nel codice

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

Leggi Xamarin Forms Layouts online: https://riptutorial.com/it/xamarin-forms/topic/6273/xamarin-

forms-layouts

Capitolo 41: Xamarin Plugin

Examples

Condividi Plugin

Un modo semplice per condividere un messaggio o un collegamento, copiare il testo negli Appunti o aprire un browser in qualsiasi app Xamarin o Windows.

Disponibile su NuGet: https://www.nuget.org/packages/Plugin.Share/

XAML

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

 </StackLayout>
```

C #

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

ExternalMaps

Plug-in per mappe esterne Aprire le mappe esterne per navigare verso una geolocalizzazione o un indirizzo specifici. Opzione da lanciare con l'opzione di navigazione anche su iOS.

Disponibile su 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>
```

Codice

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

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

Geolocator Plugin

Accedi facilmente alla geolocalizzazione su Xamarin.iOS, Xamarin.Android e Windows.

Disponibile Nuget: [https://www.nuget.org/packages/Xam.Plugin.Geolocator/][1]

XAML

```
 <StackLayout Spacing="10" Padding="10">
      <Button x:Name="buttonGetGPS" Text="Get GPS"/>
      <Label x:Name="labelGPS"/>
      <Button x:Name="buttonTrack" Text="Track Movements"/>
      <Label x:Name="labelGPSTrack"/>
      <Label Text=""/>

    </StackLayout>
```

Codice

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

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

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

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

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

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

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

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

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

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

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


        }

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

Media Plugin

Scatta o scegli foto e video da un'API multipiattaforma.

Disponibile Nuget: [https://www.nuget.org/packages/Xam.Plugin.Media/][1]

XAML

```
<StackLayout Spacing="10" Padding="10">
      <Button x:Name="takePhoto" Text="Take Photo"/>
```

```
      <Button x:Name="pickPhoto" Text="Pick Photo"/>
      <Button x:Name="takeVideo" Text="Take Video"/>
      <Button x:Name="pickVideo" Text="Pick Video"/>
      <Label Text="Save to Gallery"/>
      <Switch x:Name="saveToGallery" IsToggled="false" HorizontalOptions="Center"/>
      <Label Text="Image will show here"/>
      <Image x:Name="image"/>
      <Label Text=""/>

    </StackLayout>
```

Codice

```
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 di messaggistica

Plugin di messaggistica per Xamarin e Windows per effettuare una chiamata telefonica, inviare un sms o inviare un'e-mail utilizzando le applicazioni di messaggistica predefinite sulle diverse piattaforme mobili.

Nuget disponibile: [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>
```

Codice

```
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 di autorizzazioni

Verifica se i tuoi utenti hanno concesso o negato le autorizzazioni per i gruppi di permessi comuni su iOS e Android.

Inoltre, puoi richiedere le autorizzazioni con una semplice API asincrona / attendibile multipiattaforma.

Disponibile Nuget: https://www.nuget.org/packages/Plugin.Permissions inserisci la descrizione del link qui **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>
```

Codice

```
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;
        }
```

Leggi Xamarin Plugin online: https://riptutorial.com/it/xamarin-forms/topic/7017/xamarin-plugin

Capitolo 42: Xamarin Relative Layout

Osservazioni

L'uso di *ForceLayout* in questo caso

Le dimensioni dell'etichetta e del pulsante cambiano in base al testo all'interno di esse. Pertanto, quando i bambini vengono aggiunti al layout, la loro dimensione rimane 0 sia in larghezza che in altezza. Per esempio:

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

L'espressione sopra restituirà 0 perché la larghezza è 0 al momento. Per ovviare a questo, dobbiamo ascoltare l'evento *SizeChanged* e quando la dimensione cambia dovremmo forzare il layout per ridisegnarlo.

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

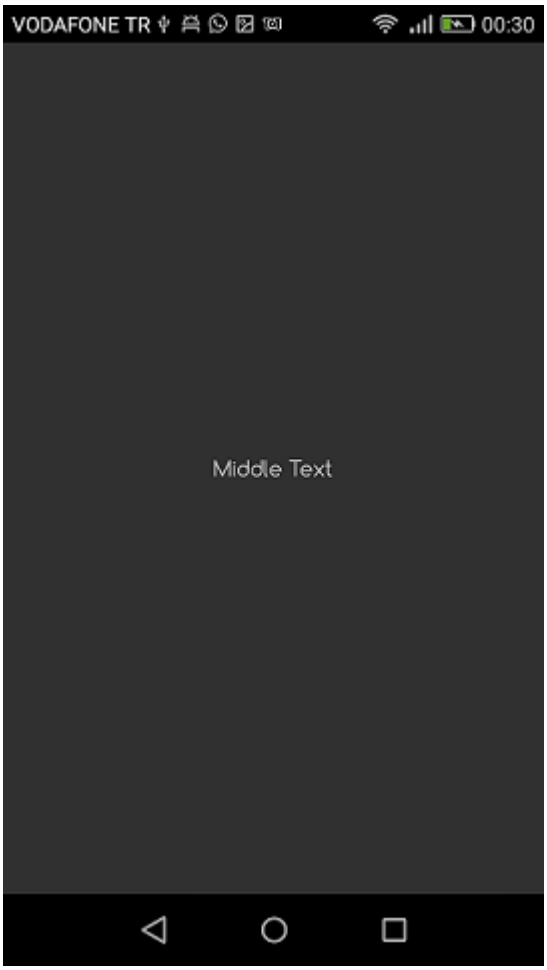
Per una vista come *BoxView* questo non è necessario. Perché possiamo definire le loro dimensioni in istanza. L'altra cosa è che in entrambi i casi possiamo definire la loro larghezza e altezza come un vincolo quando li stiamo aggiungendo al layout. Per esempio:

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

Ciò aggiungerà l'etichetta al punto 0, 0. La larghezza e l'altezza dell'etichetta saranno 30 e 40. Tuttavia, se il testo è troppo lungo, alcuni potrebbero non essere visualizzati. Se l'etichetta ha o potrebbe avere un'altezza elevata, puoi utilizzare la proprietà *LineBreakMode* dell'etichetta. Quale può avvolgere il testo. Ci sono molte opzioni in *enum LineBreakMode* .

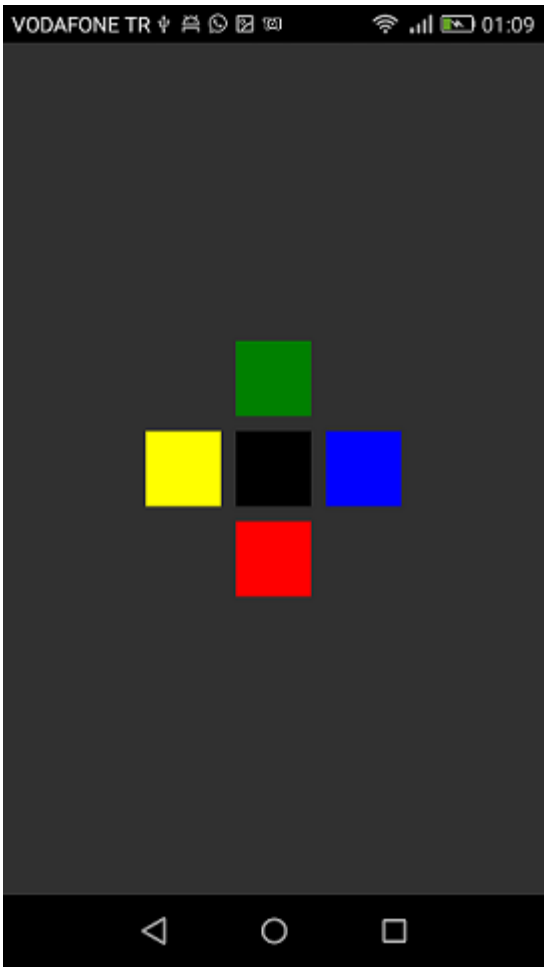
Examples

Pagina con un'etichetta semplice al centro




```
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;
    }
}
```

Scatola dopo scatola




```
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;
    }
}
```

Leggi Xamarin Relative Layout online: https://riptutorial.com/it/xamarin-forms/topic/6583/xamarin-relative-layout

Capitolo 43: Xamarin.Forms Views

Examples

Pulsante

Il **pulsante** è probabilmente il controllo più comune non solo nelle applicazioni mobili, ma in tutte le applicazioni che dispongono di un'interfaccia utente. Il concetto di un pulsante ha troppi scopi da elencare qui. In generale, tuttavia, si utilizzerà un pulsante per consentire agli utenti di avviare una sorta di azione o operazione all'interno dell'applicazione. Questa operazione potrebbe includere qualsiasi cosa, dalla navigazione di base all'interno della tua app, all'invio di dati a un servizio web da qualche parte su 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!";
}
```

Codice

```
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!";
};
```

Date picker

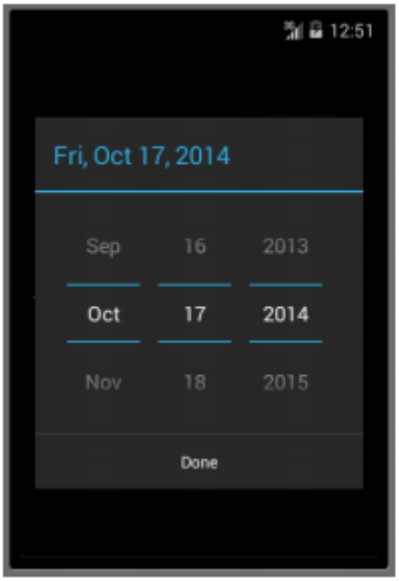
Molto spesso all'interno delle applicazioni mobili, ci sarà un motivo per gestire le date. Quando lavori con le date, probabilmente avrai bisogno di una sorta di input da parte dell'utente per selezionare una data. Ciò potrebbe verificarsi quando si lavora con un'app di pianificazione o di calendario. In questo caso, è meglio fornire agli utenti un controllo specializzato che consenta loro di selezionare in modo interattivo una data, anziché richiedere agli utenti di digitare manualmente una data. È qui che il controllo DatePicker è davvero utile.

XAML

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

Codice

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



Iscrizione

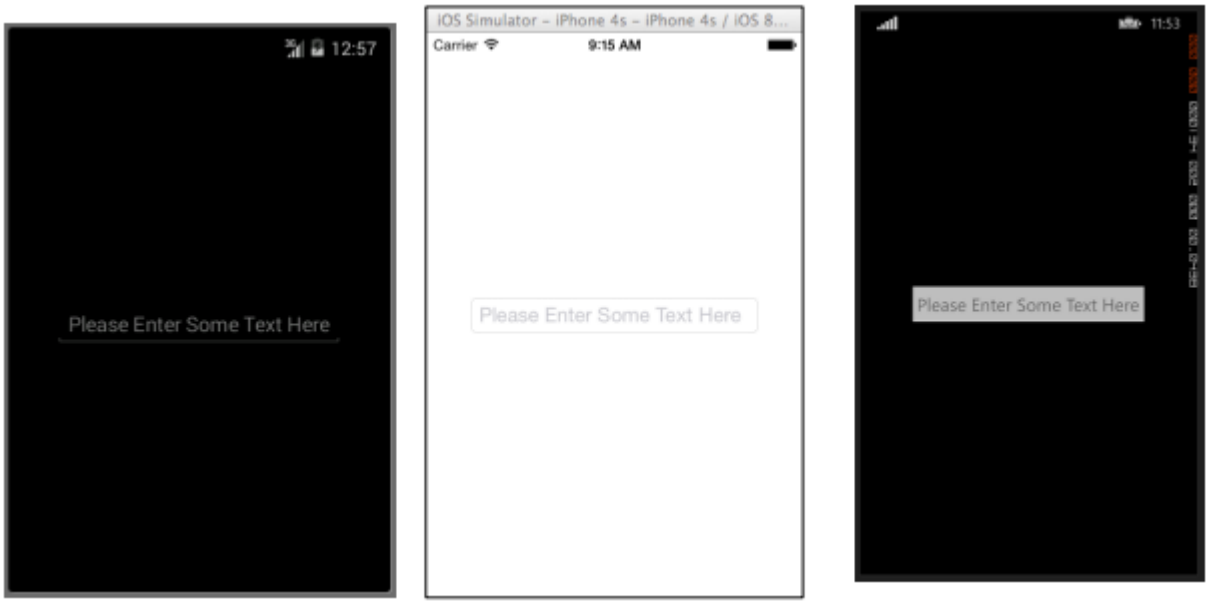
La Visualizzazione voce viene utilizzata per consentire agli utenti di digitare una singola riga di testo. Questa singola riga di testo può essere utilizzata per più scopi, incluso l'inserimento di note di base, credenziali, URL e altro. Questa vista è una vista multiuso, il che significa che se hai bisogno di digitare un testo normale o di voler oscurare una password, tutto avviene attraverso questo singolo controllo.

XAML

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

Codice

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



editore

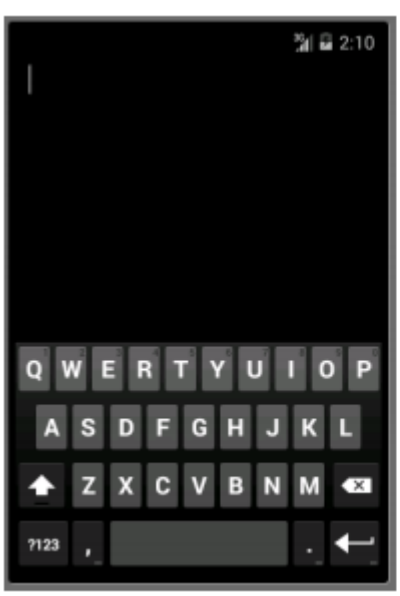
L'editor è molto simile alla voce in quanto consente agli utenti di inserire del testo in forma libera. La differenza è che l'editor consente l'immissione su più righe, mentre l'ingresso è utilizzato solo per l'ingresso a linea singola. La voce fornisce anche alcune proprietà in più rispetto all'Editor per consentire un'ulteriore personalizzazione della vista.

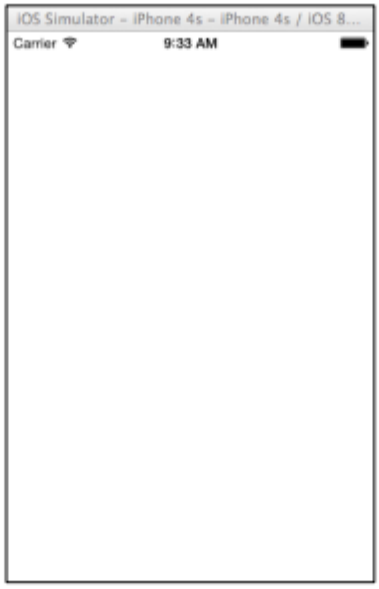
XAML

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

Codice

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





Immagine

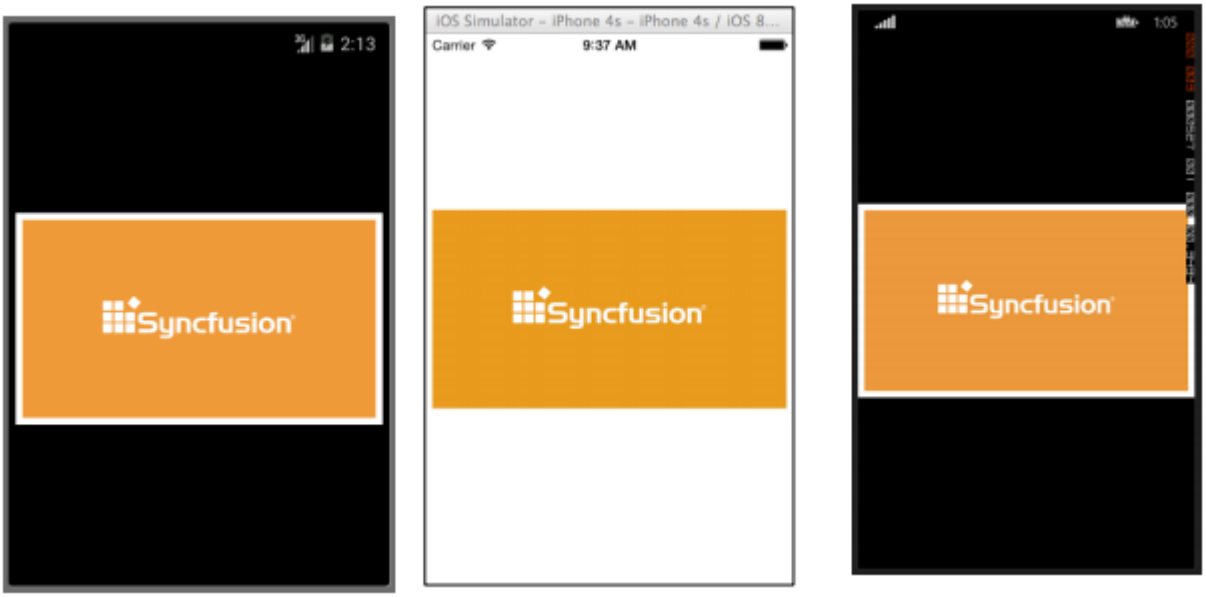
Le immagini sono parti molto importanti di qualsiasi applicazione. Offrono l'opportunità di inserire ulteriori elementi visivi e il branding nella tua applicazione. Per non parlare del fatto che le immagini sono in genere più interessanti da guardare rispetto a testo o pulsanti. Puoi utilizzare un'immagine come elemento autonomo all'interno dell'applicazione, ma un elemento Immagine può anche essere aggiunto ad altri elementi di visualizzazione come un pulsante.

XAML

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

Codice

```
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"))
};
```



Etichetta

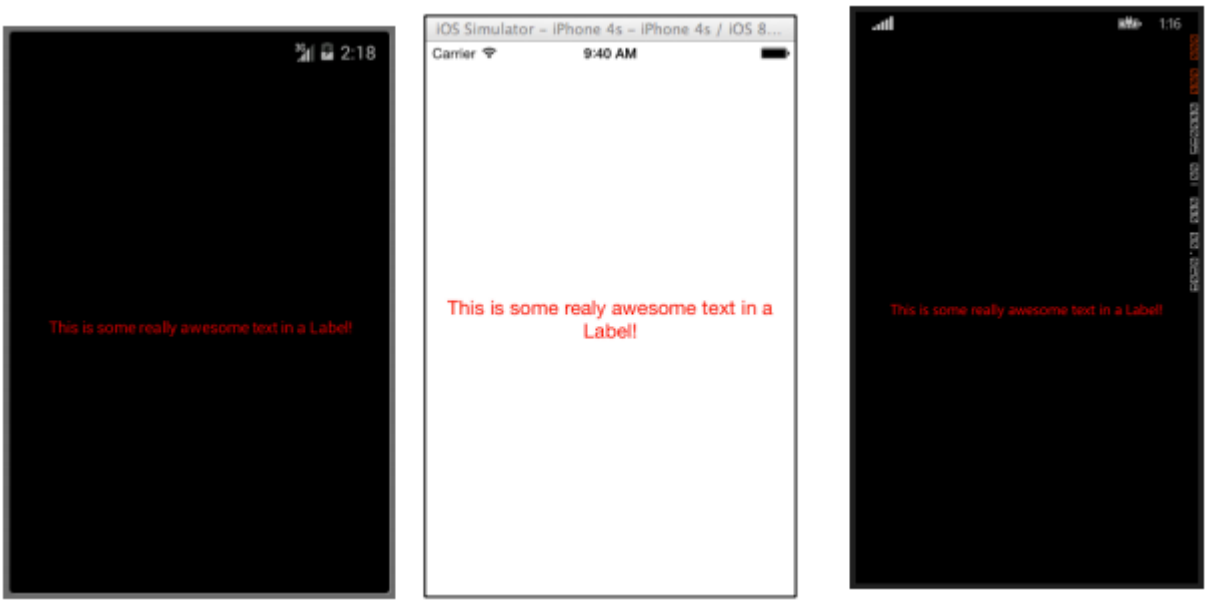
Che ci crediate o no, l'etichetta è una delle classi di vista più importanti ma non sottovalutate, non solo in Xamarin.Forms, ma nello sviluppo dell'interfaccia utente in generale. È visto come una linea di testo piuttosto noiosa, ma senza quella linea di testo sarebbe molto difficile trasmettere certe idee all'utente. I controlli delle etichette possono essere utilizzati per descrivere ciò che l'utente dovrebbe inserire in un editor o controllo di inserimento. Possono descrivere una sezione dell'interfaccia utente e dargli un contesto. Possono essere utilizzati per mostrare il totale in un'app calcolatrice. Sì, l'etichetta è davvero il controllo più versatile nella tua borsa degli attrezzi che potrebbe non suscitare sempre molta attenzione, ma è il primo a notarlo se non è lì.

XAML

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

Codice

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



Leggi Xamarin.Forms Views online: https://riptutorial.com/it/xamarin-forms/topic/7369/xamarin-forms-views

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con Xamarin.Forms	Akshay Kulkarni, chrisntr, Community, Demitrian, hankide, jdstaerk, Manohar, patridge, Sergey Metlov, spaceplane
2	Accesso alle funzionalità native con DependencyService	Gerald Versluis, hankide, hvaughan3, Sergey Metlov
3	AppSettings Reader in Xamarin.Forms	Ben Ishiyama-Levy, GvSharma
4	Associazione dati	Andrew, Matthew, Yehor Hromadskyi
5	Avviso display	aboozz pallikara, GvSharma, Sreeraj, Yehor Hromadskyi
6	caching	Sergey Metlov
7	Caratteri personalizzati negli stili	Roma Rudyak
8	CarouselView - Versione pre-release	dpserge
9	Cellule Xamarin.Forms	Eng Soon Cheah
10	Ciclo di vita generico dell'app Xamarin.Forms? Platform-dipendente!	Zverev Eugene
11	Comportamento specifico della piattaforma	Ege Aydın
12	Creazione di controlli personalizzati	hvaughan3, spaceplane, Yehor Hromadskyi
13	Database SQL e API in Xamarin Forms.	RIYAZ
14	DependencyService	Steven Thewissen

15	effetti	Swaminathan Vetri
16	Gesti	doerig, Gerald Versluis, Michael Rumpler
17	Gesto Xamarin	Joehl
18	La gestione delle eccezioni	Yehor Hromadskyi
19	Lavorare con database locali	Luis Beltran, Manohar
20	Lavorare con Maps	Taras Shevchuk
21	Le notifiche push	Gerald Versluis, user1568891
22	MessagingCenter	Gerald Versluis
23	Navigazione in Xamarin.Forms	Fernando Arreguín, jimmgarr, Lucas Moura Veloso, Paul, Sergey Metlov, Taras Shevchuk, Willian D. Andrade
24	OAuth2	Eng Soon Cheah
25	Pagina Xamarin.Forms	Eng Soon Cheah
26	Perché Xamarin Forms e When to use Xamarin Forms	Daniel Krzyczkowski, mike
27	Regolazioni visive specifiche della piattaforma	Alois, GalaxiaGuy, Paul
28	Renderizzatori personalizzati	Bonelol, hankide, Nicolas Bodin-Ripert, Nicolas Bodin-Ripert, nishantvodoo, Yehor Hromadskyi, Zverev Eugene
29	Selettore contatti - Form Xamarin (Android e iOS)	Pucho Eric
30	Servizi di dipendenza	RIYAZ
31	Test dell'unità BDD in Xamarin.Forms	Ben Ishiyama-Levy
32	Test unitario	jerone, Sergey Metlov
33	Trigger & Behaviors	hamalaiv, hvaughan3

34	Utilizzando ListViews	cvanbeek
35	Xamarin Forms Layouts	Eng Soon Cheah, Gerald Versluis, Lucas Moura Veloso
36	Xamarin Plugin	Eng Soon Cheah
37	Xamarin Relative Layout	Ege Aydın
38	Xamarin.Forms Views	Aaron Thompson, Eng Soon Cheah