EBook Gratuito

APPRENDIMENTO Xamarin.Android

Free unaffiliated eBook created from
Stack Overflow contributors.

#xamarin.android

Sommario

Di

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

It is an unofficial and free Xamarin.Android 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.Android.

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.Android

Osservazioni

Xamarin.Android consente di creare applicazioni Android native utilizzando gli stessi controlli dell'interfaccia utente di Java, tranne con la flessibilità e l'eleganza di un linguaggio moderno (C #), la potenza della libreria di base di .NET (BCL) e due IDE di prima classe - Xamarin Studio e Visual Studio - a portata di mano.

Per ulteriori informazioni sull'installazione di Xamarin.Android sul computer Mac o Windows, fare riferimento alle guide introduttive sul centro di sviluppo Xamarin

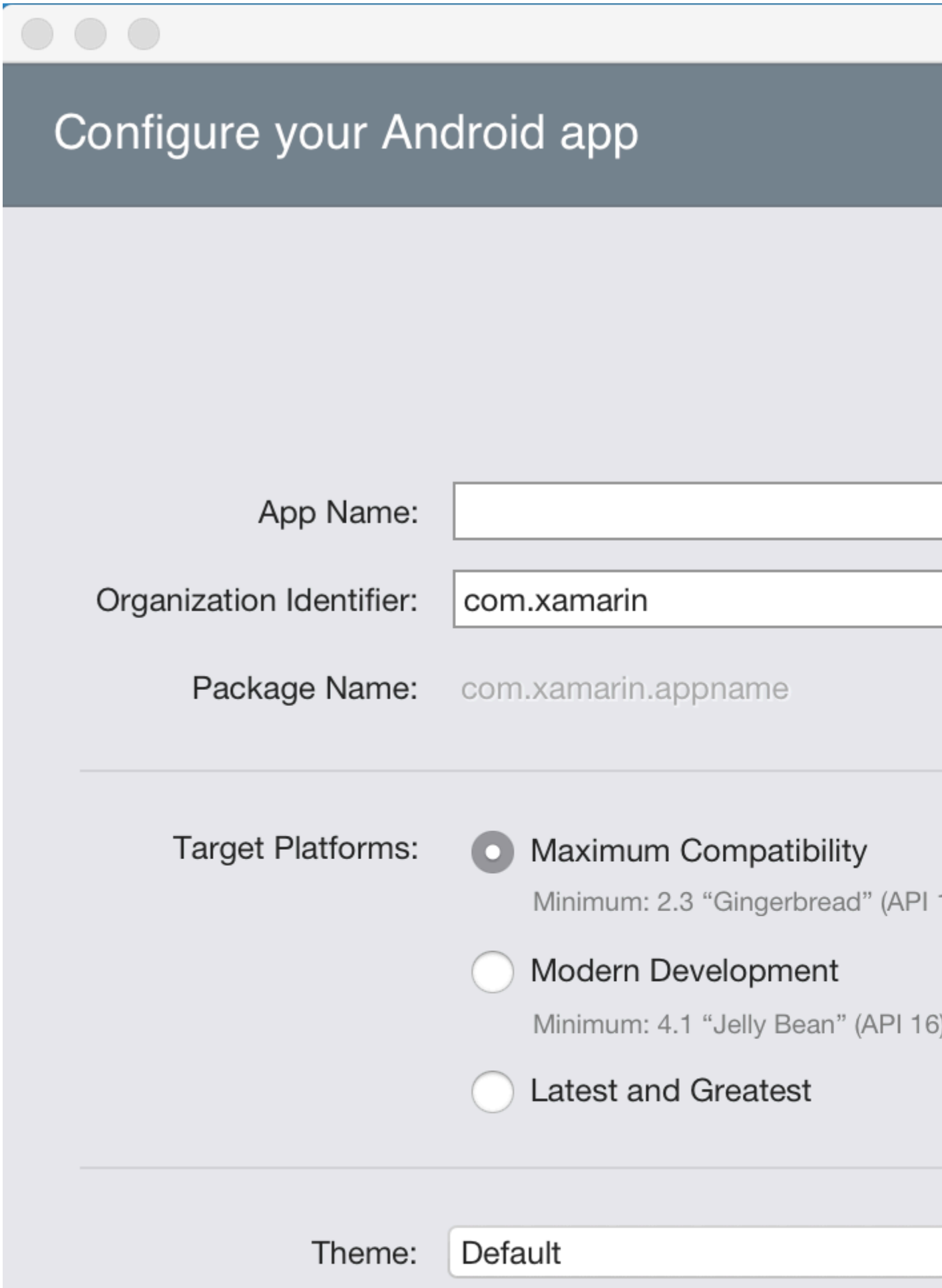
Versioni

Versione	Nome in codice	Livello API	Data di rilascio
1.0	Nessuna	1	2008-09-23
1.1	Nessuna	2	2009-02-09
1.5	Cupcake	3	2009-04-27
1.6	Ciambella	4	2009-09-15
2.0-2.1	pasticcino	5-7	2009-10-26
2.2-2.2.3	Froyo	8	2010-05-20
2.3-2.3.7	Pan di zenzero	9-10	2010-12-06
3.0-3.2.6	Favo	11-13	2011-02-22
4.0-4.0.4	Panino gelato	14-15	2011-10-18
4.1-4.3.1	Jelly Bean	16-18	2012-07-09
4.4-4.4.4, 4.4W-4.4W.2	KitKat	19-20	2013/10/31
5.0-5.1.1	Lecca-lecca	21-22	2014/11/12
6.0-6.0.1	Marshmallow	23	2015/10/05
7.0	Torrone	24	2016/08/22

Examples

Inizia in Xamarin Studio

1. Passare a **File> Nuovo> Soluzione** per visualizzare la nuova finestra di dialogo del progetto.
2. Seleziona **App Android** e premi **Avanti** .
3. Configura la tua app impostando il nome dell'app e l'ID dell'organizzazione. Seleziona la piattaforma di destinazione più adatta alle tue esigenze o lasciala come predefinita. Premi Successivo:

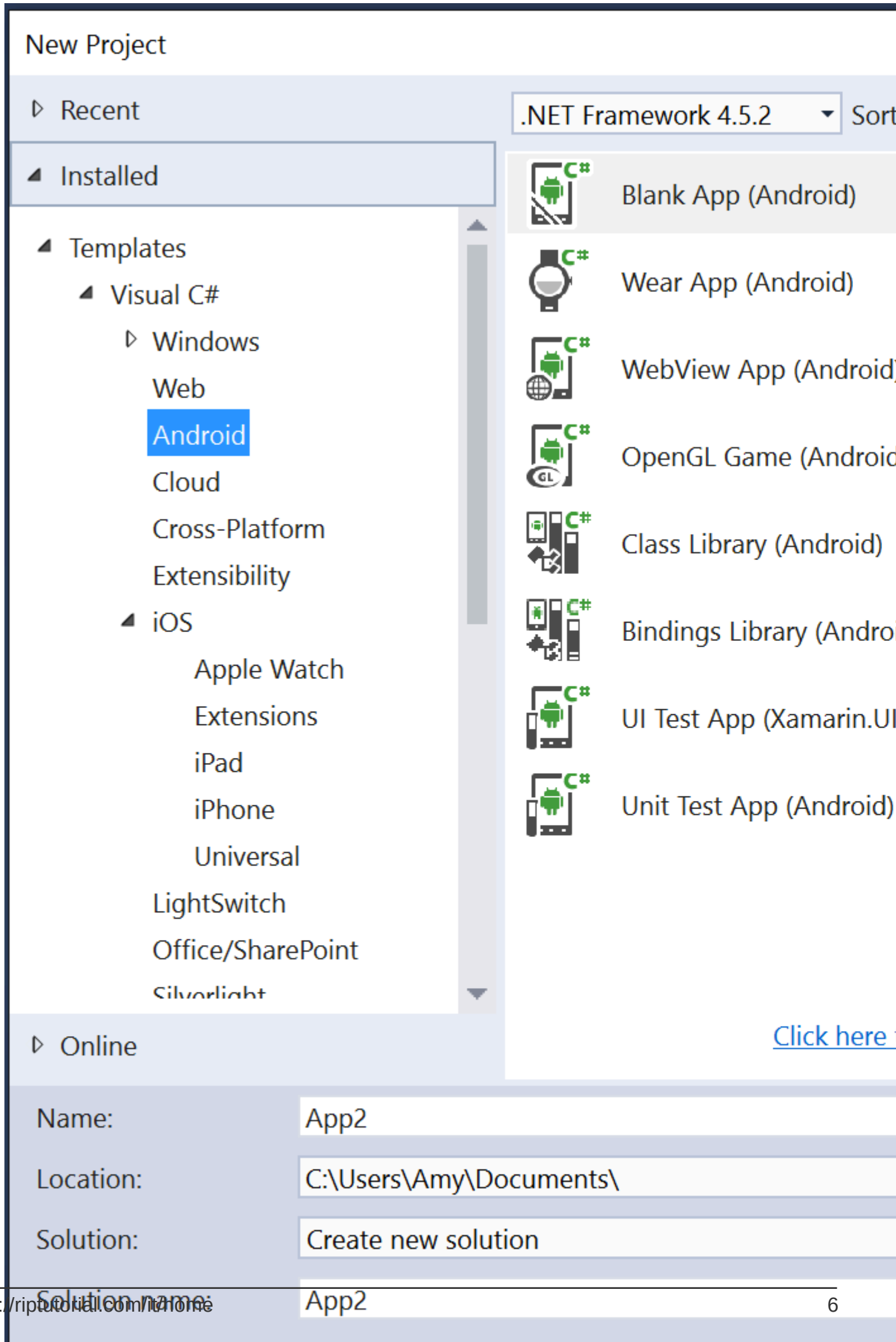

Configure your Android app
App Name:
Organization Identifier:
com.xamarin
Package Name:
com.xamarin.appname
Target Platforms:
Maximum Compatibility
Minimum: 2.3 “Gingerbread” (API
Modern Development
Minimum: 4.1 “Jelly Bean” (API 16)
Latest and Greatest
Theme:
Default

4. Imposta il nome del progetto e il nome della soluzione, o lascia il nome predefinito. Fai clic su Crea per creare il tuo progetto.
5. Configurare il dispositivo per la distribuzione o configurare un emulatore
6. Per eseguire l'applicazione, seleziona la configurazione **Debug** e premi il pulsante

   Riproduci:
   e premi il pulsante Riproduci:

Inizia in Visual Studio

1. Passare a **File> Nuovo> Progetto** per visualizzare la finestra di dialogo Nuovo progetto.
2. Vai a **Visual C #> Android** e seleziona App vuota:


New Project
Recent
.NET Framework 4.5.2
Sort
Installed
Templates
Visual C#
Windows
Web
Android
Cloud
Cross-Platform
Extensibility
iOS
Apple Watch
Extensions
iPad
iPhone
Universal
LightSwitch
Office/SharePoint
Silverlight
Online
Blank App (Android)
Wear App (Android)
WebView App (Android)
OpenGL Game (Android
Class Library (Android)
Bindings Library (Andro
UI Test App (Xamarin.UI
Unit Test App (Android)
Click here
Name:
App2
Location:
C:\Users\Amy\Documents\
Solution:
Create new solution
Solution name:
App2

3. Dai un **nome alla** tua app e premi **OK** per creare il tuo progetto.
4. Configurare il dispositivo per la distribuzione o configurare un emulatore
5. Per eseguire l'applicazione, selezionare la configurazione di **debug** e premere il pulsante

Start :

Start : 

Leggi Iniziare con Xamarin.Android online: https://riptutorial.com/it/xamarin-android/topic/403/iniziare-con-xamarin-android

Capitolo 2: Attacchi

Examples

Rimozione dei tipi

È possibile istruire Xamarin.Android Bindings Generator per ignorare un tipo Java e non vincolarlo. Questo viene fatto aggiungendo un elemento XML `remove-node` al file metadata.xml:

```
<remove-node path="/api/package[@name='{package_name}']/class[@name='{name}']" />
```

Implementazione di interfacce Java

Se una libreria java contiene interfacce che devono essere implementate dall'utente (ad es. `View.IOnClickListener` clic come `View.IOnClickListener` o callbacks), la classe di implementazione deve ereditare, direttamente o indirettamente, da `Java.Lang.Object` o `Java.Lang.Throwable` . Questo è un errore comune, perché i passaggi della confezione stampano semplicemente un avviso che viene trascurato facilmente:

> Digitare "MyListener" implementa Android.Runtime.IJavaObject ma non eredita da Java.Lang.Object. Non è supportato

Sbagliato

L'utilizzo di questa implementazione risulterà in un comportamento imprevisto.

```
class MyListener : View.IOnClickListener
{
    public IntPtr Handle { get; }

    public void Dispose()
    {
    }


    public void OnClick(View v)
    {
        // ...
    }
}
```

Corretta

```
class MyListener :
    Java.Lang.Object, // this is the important part
    View.IOnClickListener
{
    public void OnClick(View v)
    {
        // ...
```

```
    }
}
```

Le librerie di bind possono rinominare i metodi e le interfacce

Non tutto in una libreria di binding avrà lo stesso nome in C # come in Java.

In C #, i nomi delle interfacce iniziano con "I", ma Java non ha tale convenzione. Quando `SomeInterface` una libreria Java, un'interfaccia denominata `SomeInterface` diventerà `ISomeInterface` .

Allo stesso modo, Java non ha proprietà come C #. Quando una libreria è vincolata, i metodi getter e setter Java potrebbero essere refactored come proprietà. Ad esempio, il seguente codice Java

```
public int getX() { return someInt; }

public int setX(int someInt) { this.someInt = someInt; }
```

può essere refactorato come

```
public int X { get; set; }
```

quando è legato.

Leggi Attacchi online: https://riptutorial.com/it/xamarin-android/topic/771/attacchi

Capitolo 3: brindisi

Examples

Messaggio di base Toast

Innanzitutto, `MakeText()` un'istanza di un oggetto Toast con uno dei metodi `MakeText()` . Questo metodo richiede tre parametri: il `Context` dell'applicazione, il messaggio di testo e la durata del brindisi. Restituisce un oggetto Toast correttamente inizializzato. È possibile visualizzare la notifica del brindisi con `Show()` , come mostrato nell'esempio seguente:

```
Context context = Application.Context;
string text = "Hello toast!";
ToastLength duration = ToastLength.Short;

var toast = Toast.MakeText(context, text, duration);
toast.Show();
```

Questo esempio mostra tutto il necessario per la maggior parte delle notifiche di brindisi. Dovresti raramente aver bisogno di qualcos'altro. Tuttavia, è possibile posizionare il toast in modo diverso o persino utilizzare il proprio layout anziché un semplice messaggio di testo. Le seguenti sezioni descrivono come puoi fare queste cose.

Puoi anche concatenare i tuoi metodi, chiamare come one-liner ed evitare di mantenere l'oggetto Toast, come questo:

```
Toast.MakeText(Application.Context, "Hello toast!", ToastLength.Short).Show();
```

Per ulteriori informazioni, fare riferimento alla documentazione Android completa sull'argomento.

Messaggi colorati Toast

A volte vogliamo dare ulteriori informazioni al nostro utente con i colori (ad esempio rosso significa che è successo qualcosa di sbagliato) Possiamo cambiare il colore di sfondo del messaggio di toast usando l'impostazione di un filtro colorato per la vista che il nostro brindisi ci dà (qui uso ColorMatrixColorFilter):

```
Toast t = Toast.MakeText(context, message, duration);
Color c = */your color/*;
ColorMatrixColorFilter CM = new ColorMatrixColorFilter(new float[]
    {
        0,0,0,0,c.R,
        0,0,0,0,c.G,
        0,0,0,0,c.B,
        0,0,0,1,0
    });
t.View.Background.SetColorFilter(CM);
t.Show();
```

E inoltre possiamo cambiare il colore del testo se lo sfondo è chiaro o scuro:

```
if ((((float)(c.R) + (float)(c.G) + (float)(c.B)) / 3) >= 128)
    t.View.FindViewById<TextView>(Android.Resource.Id.Message).SetTextColor(Color.Black);
else
//text color is white by default
```

Cambia posizione Toast

Possiamo cambiare il nostro brindisi usando il metodo SetGravity. Questo metodo richiede tre parametri: il primo è la gravità del pane tostato sullo schermo e altri due impostano l'offset del toast dalla posizione di partenza (impostata dal primo parametro):

```
//Toast at bottom left corner of screen
Toast t = Toast.MakeText(context, message, duration);
t.SetGravity(GravityFlags.Bottom | GravityFlags.Left, 0, 0);
t.Show();

//Toast at a custom position on screen
Toast t = Toast.MakeText(context, message, duration);
t.SetGravity(GravityFlags.Top | GravityFlags.Left, x, y);
t.Show();
```

Leggi brindisi online: https://riptutorial.com/it/xamarin-android/topic/3550/brindisi

Capitolo 4: Ciclo di vita dell'app - Xamarin.Andorid

introduzione

Il ciclo di vita delle applicazioni di Xamarin.Android è uguale alla normale app Android. Quando parliamo del ciclo di vita dobbiamo parlare di: ciclo di vita delle applicazioni, ciclo di vita delle attività e ciclo di vita dei frammenti.

Nel seguito proverò a fornire una buona descrizione e il modo di usarli. Ho ottenuto questa documentazione dalla documentazione ufficiale di Android e Xamarin e molte utili risorse web fornite nella sezione commenti sotto.

Osservazioni

Link interessanti per ampliare le tue conoscenze sul ciclo di vita delle applicazioni Android:

https://developer.android.com/reference/android/app/Activity.html

http://www.vogella.com/tutorials/AndroidLifeCycle/article.html

https://github.com/xxv/android-lifecycle

https://developer.android.com/guide/components/fragments.html

https://developer.xamarin.com/guides/android/platform_features/fragments/part_1_-_creating_a_fragment/

https://developer.android.com/guide/components/activities/activity-lifecycle.html

Examples

Ciclo di vita delle applicazioni

Prima di tutto devi sapere che puoi estendere la classe Android.Application in modo da poter accedere a due importanti metodi relativi al ciclo di vita delle app:

- OnCreate: chiamato quando l'applicazione viene avviata, prima che siano stati creati altri oggetti dell'applicazione (come MainActivity).
- OnTerminate: questo metodo è destinato all'uso in ambienti di processo emulati. Non verrà mai chiamato su un dispositivo Android di produzione, in cui i processi vengono rimossi semplicemente eliminandoli; Nessun codice utente (incluso questo callback) viene eseguito quando lo si fa. Dalla documentazione: https://developer.android.com/reference/android/app/Application.html#onTerminate ()

Nell'applicazione Xamarin.Android è possibile estendere la classe Application nel modo illustrato di seguito. Aggiungi una nuova classe chiamata "MyApplication.cs" al tuo progetto:

```
[Application]
public class MyApplication : Application
{
    public MyApplication(IntPtr handle, JniHandleOwnership ownerShip) : base(handle,
ownerShip)
    {
    }

    public override void OnCreate()
    {
        base.OnCreate();
    }

    public override void OnTerminate()
    {
        base.OnTerminate();
    }
}
```

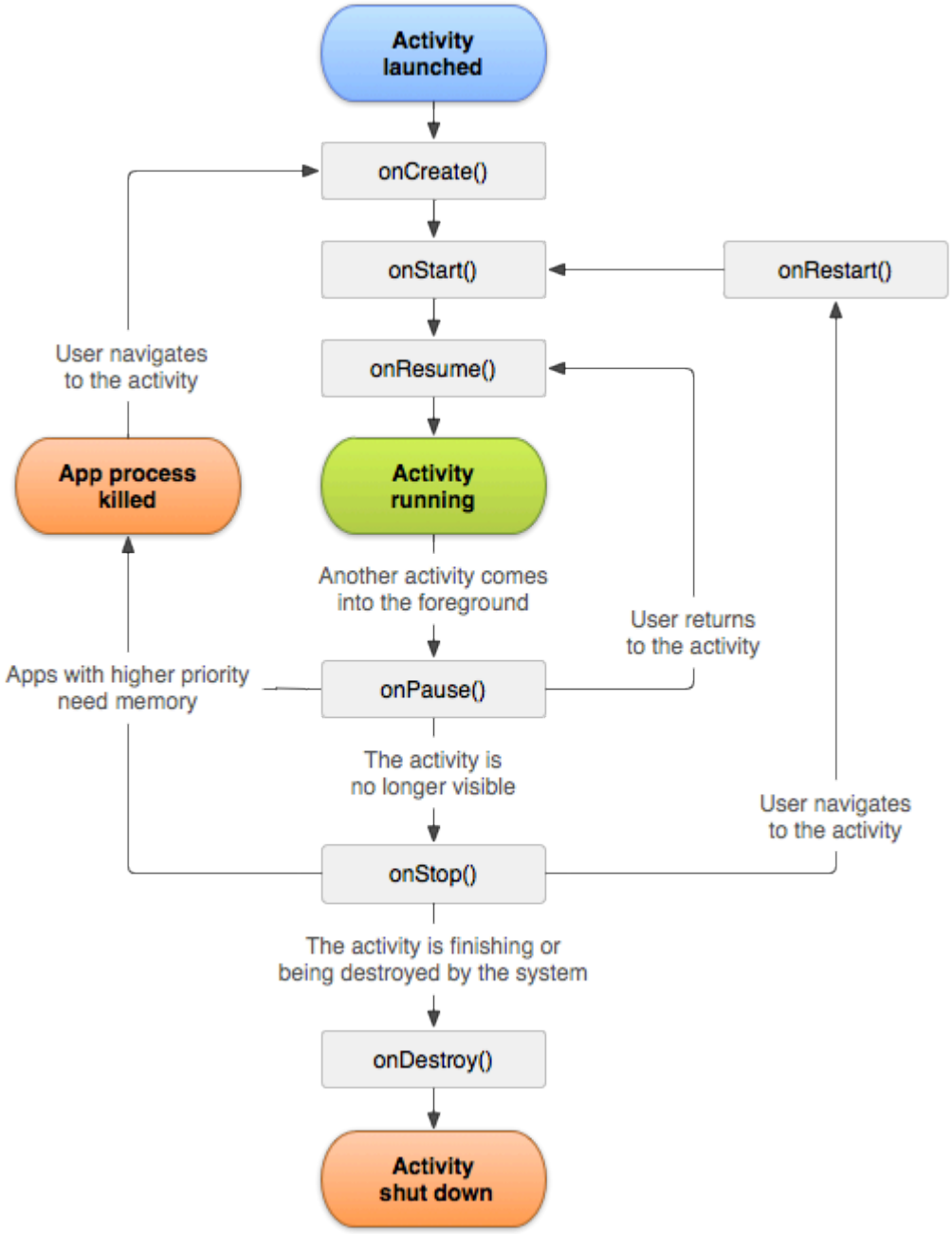
Come hai scritto sopra puoi usare il metodo OnCreate. Ad esempio, è possibile inizializzare il database locale o configurare una configurazione aggiuntiva.

Esistono anche altri metodi che possono essere sovrascritti come: OnConfigurationChanged o OnLowMemory.

Ciclo di vita delle attività

Il ciclo di vita delle attività è molto più complesso. Come sapete, Activity è una singola pagina nell'app Android in cui l'utente può interagire con esso.

Nello schema seguente puoi vedere come appare il ciclo di vita dell'Android Activity:



Come puoi vedere, esiste un flusso specifico del ciclo di vita delle attività. Nell'applicazione mobile sono naturalmente presenti metodi in ogni classe di attività che gestiscono un frammento specifico del ciclo di vita:

```
[Activity(Label = "LifecycleApp", MainLauncher = true, Icon = "@mipmap/icon")]
public class MainActivity : Activity
{
    protected override void OnCreate(Bundle savedInstanceState)
    {
        base.OnCreate(savedInstanceState);
        Log.Debug("OnCreate", "OnCreate called, Activity components are being created");

        // Set our view from the "main" layout resource
        SetContentView(Resource.Layout.MainActivity);
    }

    protected override void OnStart()
    {
```

```
        Log.Debug("OnStart", "OnStart called, App is Active");
        base.OnStart();
    }

    protected override void OnResume()
    {
        Log.Debug("OnResume", "OnResume called, app is ready to interact with the user");
        base.OnResume();
    }

    protected override void OnPause()
    {
        Log.Debug("OnPause", "OnPause called, App is moving to background");
        base.OnPause();
    }

    protected override void OnStop()
    {
        Log.Debug("OnStop", "OnStop called, App is in the background");
        base.OnStop();
    }

    protected override void OnDestroy()
    {
        base.OnDestroy();
        Log.Debug("OnDestroy", "OnDestroy called, App is Terminating");
    }
}
```

C'è una buona descrizione nella documentazione ufficiale di Android:

- L'intera durata di un'attività avviene tra la prima chiamata a onCreate (Bundle) fino a una singola chiamata finale a onDestroy (). Un'attività eseguirà tutte le impostazioni dello stato "globale" in onCreate () e rilascerà tutte le risorse rimanenti in onDestroy (). Ad esempio, se ha un thread in esecuzione in background per scaricare i dati dalla rete, può creare quel thread in onCreate () e quindi interrompere il thread in onDestroy ().

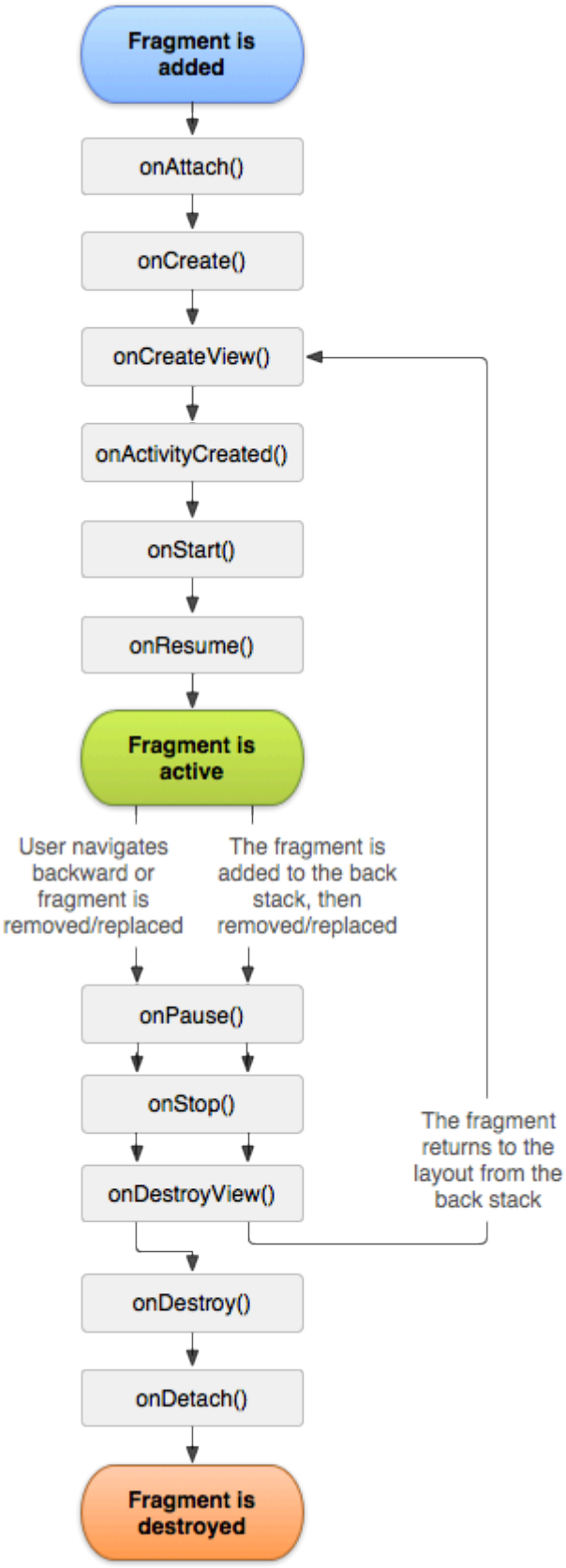
- La durata visibile di un'attività si verifica tra una chiamata a onStart () fino alla corrispondente chiamata a onStop (). Durante questo periodo, l'utente può vedere l'attività sullo schermo, anche se potrebbe non essere in primo piano e interagire con l'utente. Tra questi due metodi è possibile conservare le risorse necessarie per mostrare l'attività all'utente. Ad esempio, è possibile registrare un BroadcastReceiver in onStart () per monitorare le modifiche che hanno un impatto sull'interfaccia utente e annullarne la registrazione in onStop () quando l'utente non vede più ciò che si sta visualizzando. I metodi onStart () e onStop () possono essere chiamati più volte, in quanto l'attività diventa visibile e nascosta all'utente.

- La durata in primo piano di un'attività si verifica tra una chiamata a onResume () fino alla corrispondente chiamata a onPause (). Durante questo periodo l'attività è di fronte a tutte le altre attività e interagisce con l'utente. Un'attività può spesso passare tra gli stati di ripresa e di pausa - ad esempio quando il dispositivo va in stop, quando viene consegnato un risultato di attività, quando viene consegnato un nuovo intento - quindi il codice in questi metodi dovrebbe essere abbastanza leggero.

Ciclo di vita del frammento

Come sai, puoi avere un'attività, ma in essa sono incorporati diversi frammenti. Ecco perché il ciclo di vita dei frammenti è importante anche per gli sviluppatori.

Nello schema seguente puoi vedere come appare il ciclo di vita del frammento di Android:



Come descritto nella documentazione ufficiale di Android, è necessario implementare almeno tre metodi:

- OnCreate: il sistema chiama questo quando crea il frammento. Nell'implementazione, è necessario inizializzare i componenti essenziali del frammento che si desidera conservare quando il frammento è in pausa o arrestato, quindi riprendere.
- OnCreateView: il sistema chiama questo quando è il momento per il frammento di disegnare la sua interfaccia utente per la prima volta. Per disegnare un'interfaccia utente per il frammento, è necessario restituire una vista da questo metodo che rappresenta la radice del layout del frammento. È possibile restituire null se il frammento non fornisce un'interfaccia utente.
- OnPause: il sistema chiama questo metodo come prima indicazione che l'utente lascia il frammento (sebbene non sempre significhi che il frammento è stato distrutto). Questo di solito è dove si dovrebbero commettere eventuali modifiche che dovrebbero essere mantenute oltre la sessione utente corrente (perché l'utente potrebbe non tornare).

Ecco un'implementazione di esempio in Xamarin.Android:

```
public class MainFragment : Fragment
{
    public override void OnCreate(Bundle savedInstanceState)
    {
        base.OnCreate(savedInstanceState);

        // Create your fragment here
        // You should initialize essential components of the fragment
        // that you want to retain when the fragment is paused or stopped, then resumed.
    }

    public override View OnCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState)
    {
        // Use this to return your custom view for this Fragment
        // The system calls this when it's time for the fragment to draw its user interface
for the first time.

        var mainView = inflater.Inflate(Resource.Layout.MainFragment, container, false);
        return mainView;
    }

    public override void OnPause()
    {
        // The system calls this method as the first indication that the user is leaving the
fragment

        base.OnPause();
    }
}
```

Ovviamente è possibile aggiungere ulteriori metodi qui se si desidera gestire stati diversi.

Esempio completo su GitHub

Se desideri ottenere un progetto di base con i metodi descritti di seguito, puoi scaricare il modello di applicazione Xamarin.Android dal mio GitHub. Puoi trovare esempi per:

- Metodi del ciclo di vita delle applicazioni
- Metodi del ciclo di vita delle attività
- Metodi del ciclo di vita dei frammenti

https://github.com/Daniel-Krzyczkowski/XamarinAndroid/tree/master/AndroidLifecycle/LifecycleApp

Leggi Ciclo di vita dell'app - Xamarin.Andorid online: https://riptutorial.com/it/xamarin-android/topic/8842/ciclo-di-vita-dell-app---xamarin-andorid

Capitolo 5: Come correggere l'orientamento di un'immagine catturata dal dispositivo Android

Osservazioni

1. Questo esempio di app è disponibile sul mio GitHub di seguito:

https://github.com/Daniel-Krzyczkowski/XamarinAndroid/tree/master/AndroidPictureOrientation/PictureOrientationApp

2. La documentazione dei componenti di Xamarin Mobile è disponibile di seguito:

https://components.xamarin.com/view/xamarin.mobile

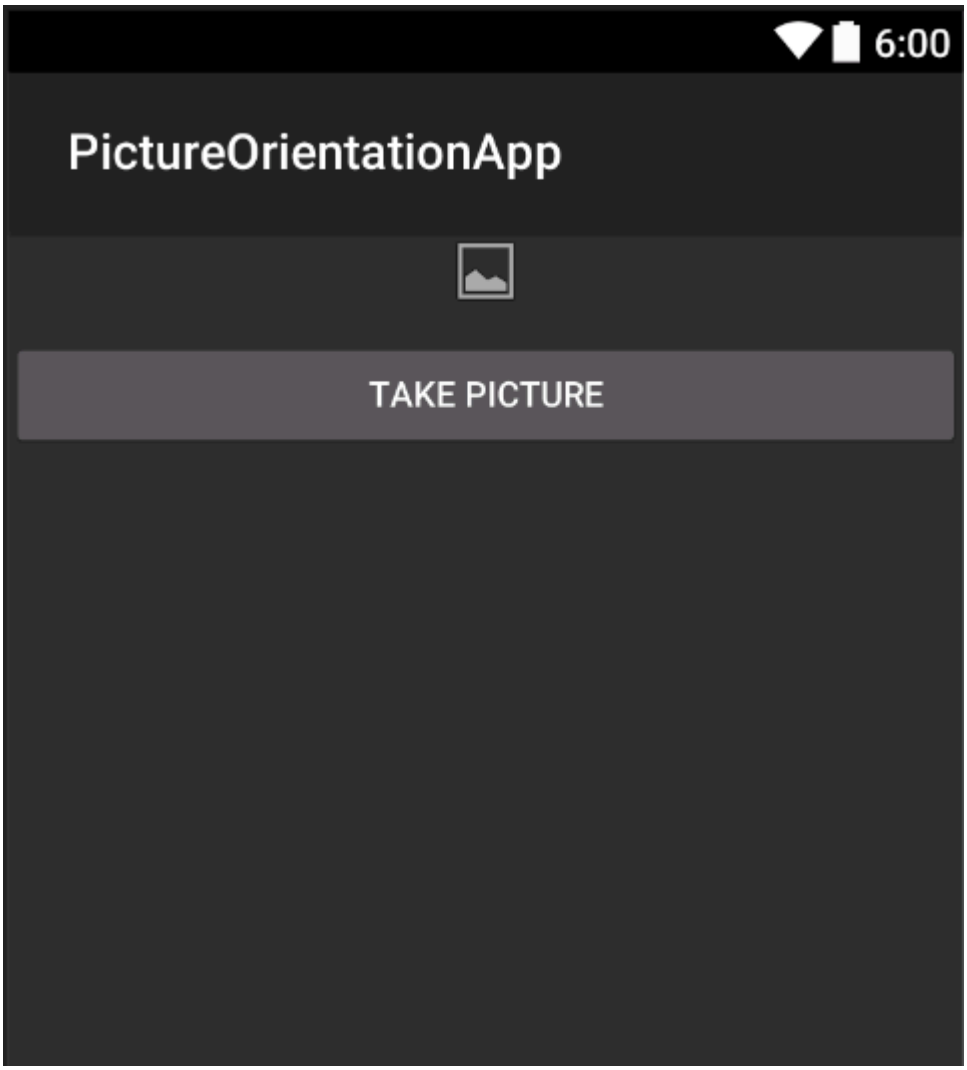
Examples

Come correggere l'orientamento di un'immagine catturata dal dispositivo Android

Questo esempio mostra come acquisire l'immagine e visualizzarla correttamente sul dispositivo Android.

Per prima cosa dobbiamo creare un'applicazione di esempio con un pulsante e una vista di immagini. Una volta che l'utente fa clic sul pulsante, la fotocamera viene avviata e, dopo che l'utente ha selezionato l'immagine, verrà visualizzata con l'orientamento corretto sullo schermo.

1. Aggiungi pulsante denominato "TakePictureButton" e vista immagine denominata "TakenPictureImageView":

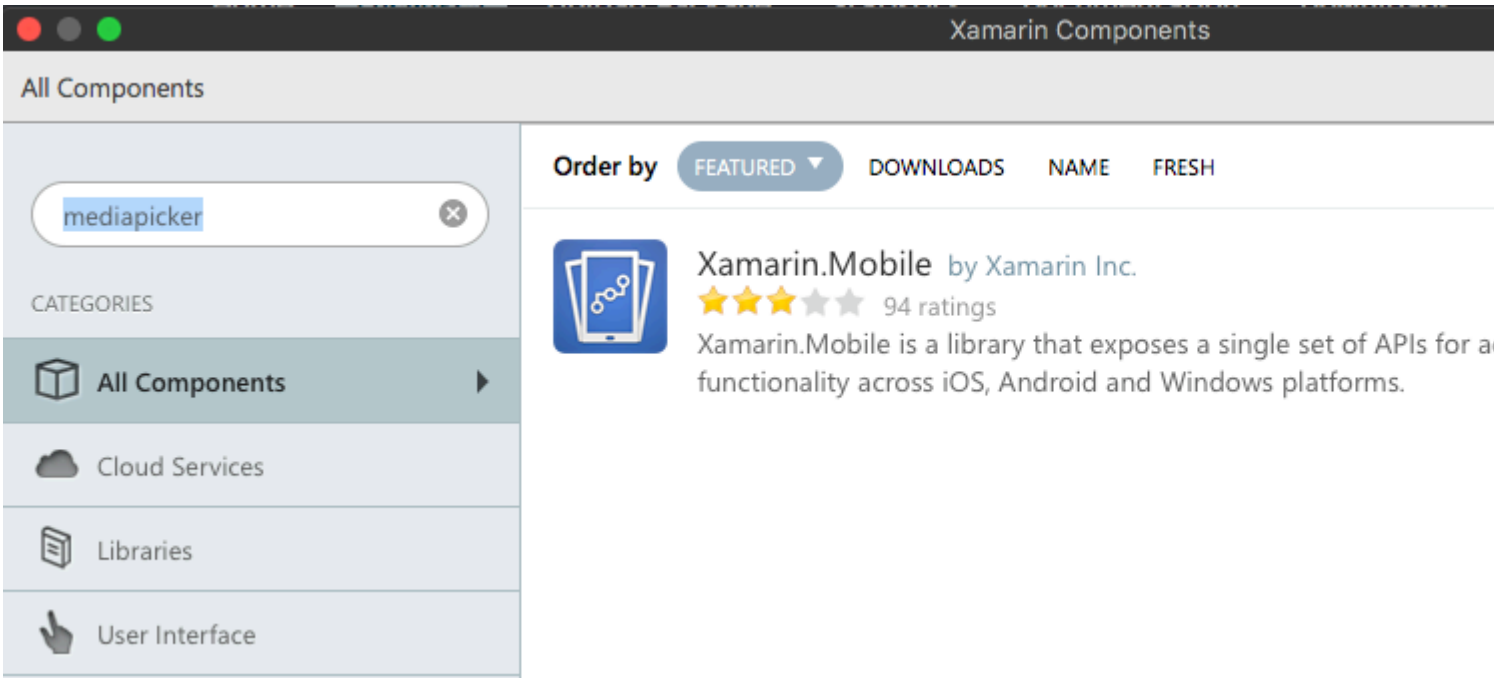


2. Ora apri il codice dell'attività dietro:

Qui in primo luogo ottieni un riferimento ai tuoi controlli:

```
ImageView _takenPictureImageView;
Button _takePictureButton;

    protected override void OnCreate(Bundle savedInstanceState)
    {
        base.OnCreate(savedInstanceState);
        SetContentView(Resource.Layout.Main);

        _takenPictureImageView = FindViewById<ImageView>(Resource.Id.TakenPictureImageView);
        _takePictureButton = FindViewById<Button>(Resource.Id.TakePictureButton);

        _takePictureButton.Click += delegate
        {
            takePicture();
        };
    }
```

3. Nella nostra applicazione useremo il componente Xamarin Mobile disponibile nel Components Store:



4. Una volta che lo aggiungi al progetto, possiamo andare avanti. Aggiungi sotto il codice che è responsabile del lancio della telecamera. Questo metodo dovrebbe essere invocato nel clic del pulsante come puoi vedere nel codice precedente:

```
void takePicture()
 {
     var picker = new MediaPicker(this);
     DateTime now = DateTime.Now;
     var intent = picker.GetTakePhotoUI(new StoreCameraMediaOptions
     {
         Name = "picture_" + now.Day + "_" + now.Month + "_" + now.Year + ".jpg",
         Directory = null
     });
     StartActivityForResult(intent, 1);
 }
```

5. Una volta che l'utente prende l'immagine, dovremmo visualizzarla nell'orientamento corretto. Per farlo usa il metodo seguente. È responsabile del recupero delle informazioni exif dall'immagine scattata (incluso l'orientamento durante il momento in cui scatta la foto) e della creazione di bitmap con l'orientamento corretto:

```
Bitmap loadAndResizeBitmap(string filePath)
 {
     BitmapFactory.Options options = new BitmapFactory.Options { InJustDecodeBounds =
true };
     BitmapFactory.DecodeFile(filePath, options);

     int REQUIRED_SIZE = 100;
     int width_tmp = options.OutWidth, height_tmp = options.OutHeight;
     int scale = 4;
     while (true)
     {
         if (width_tmp / 2 < REQUIRED_SIZE || height_tmp / 2 < REQUIRED_SIZE)
             break;
         width_tmp /= 2;
```

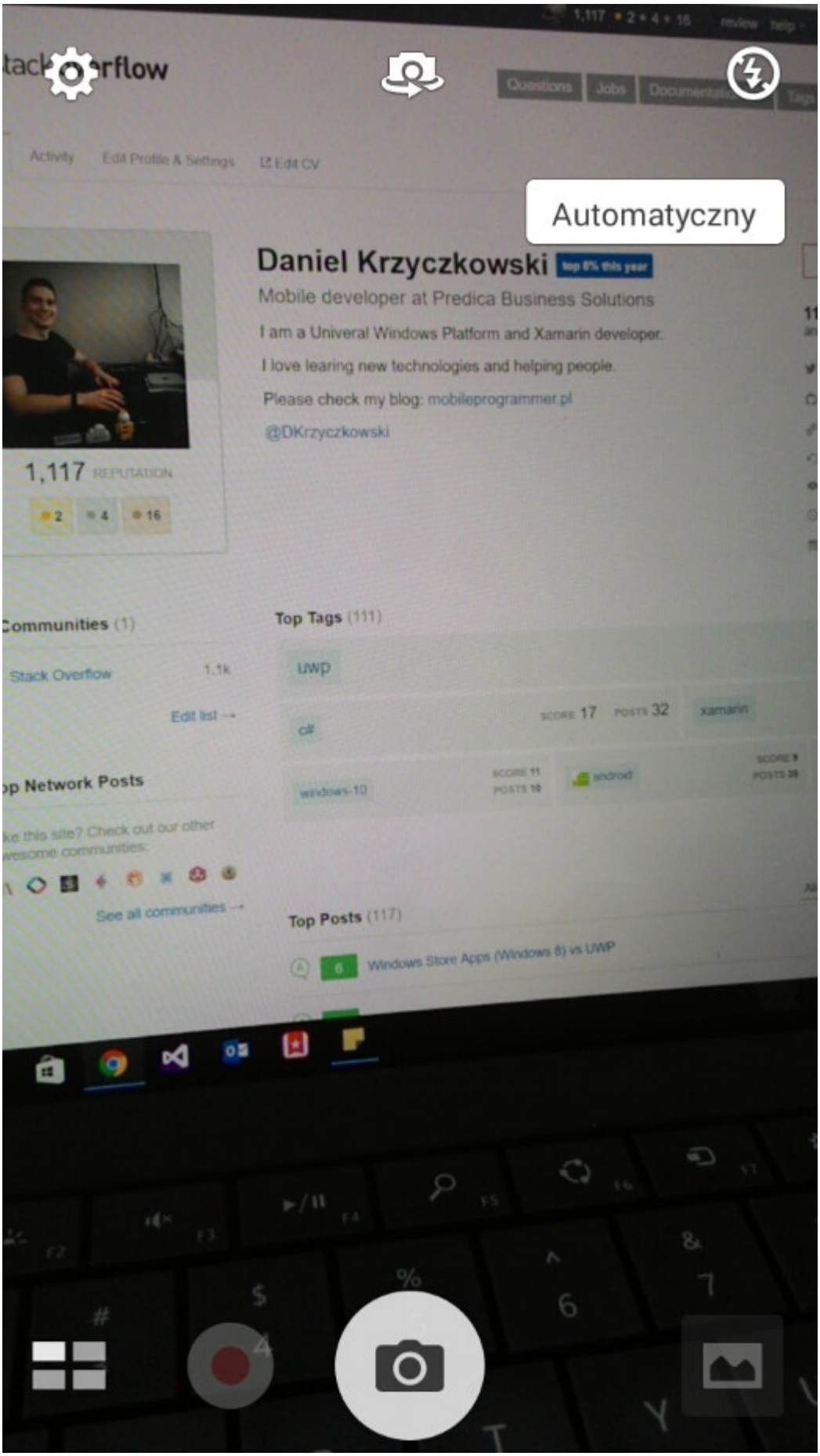
```
        height_tmp /= 2;
        scale++;
    }

    options.InSampleSize = scale;
    options.InJustDecodeBounds = false;
    Bitmap resizedBitmap = BitmapFactory.DecodeFile(filePath, options);

    ExifInterface exif = null;
    try
    {
        exif = new ExifInterface(filePath);
        string orientation = exif.GetAttribute(ExifInterface.TagOrientation);

        Matrix matrix = new Matrix();
        switch (orientation)
        {
            case "1": // landscape
                break;
            case "3":
                matrix.PreRotate(180);
                resizedBitmap = Bitmap.CreateBitmap(resizedBitmap, 0, 0,
resizedBitmap.Width, resizedBitmap.Height, matrix, false);
                matrix.Dispose();
                matrix = null;
                break;
            case "4":
                matrix.PreRotate(180);
                resizedBitmap = Bitmap.CreateBitmap(resizedBitmap, 0, 0,
resizedBitmap.Width, resizedBitmap.Height, matrix, false);
                matrix.Dispose();
                matrix = null;
                break;
            case "5":
                matrix.PreRotate(90);
                resizedBitmap = Bitmap.CreateBitmap(resizedBitmap, 0, 0,
resizedBitmap.Width, resizedBitmap.Height, matrix, false);
                matrix.Dispose();
                matrix = null;
                break;
            case "6": // portrait
                matrix.PreRotate(90);
                resizedBitmap = Bitmap.CreateBitmap(resizedBitmap, 0, 0,
resizedBitmap.Width, resizedBitmap.Height, matrix, false);
                matrix.Dispose();
                matrix = null;
                break;
            case "7":
                matrix.PreRotate(-90);
                resizedBitmap = Bitmap.CreateBitmap(resizedBitmap, 0, 0,
resizedBitmap.Width, resizedBitmap.Height, matrix, false);
                matrix.Dispose();
                matrix = null;
                break;
            case "8":
                matrix.PreRotate(-90);
                resizedBitmap = Bitmap.CreateBitmap(resizedBitmap, 0, 0,
resizedBitmap.Width, resizedBitmap.Height, matrix, false);
                matrix.Dispose();
                matrix = null;
                break;
```

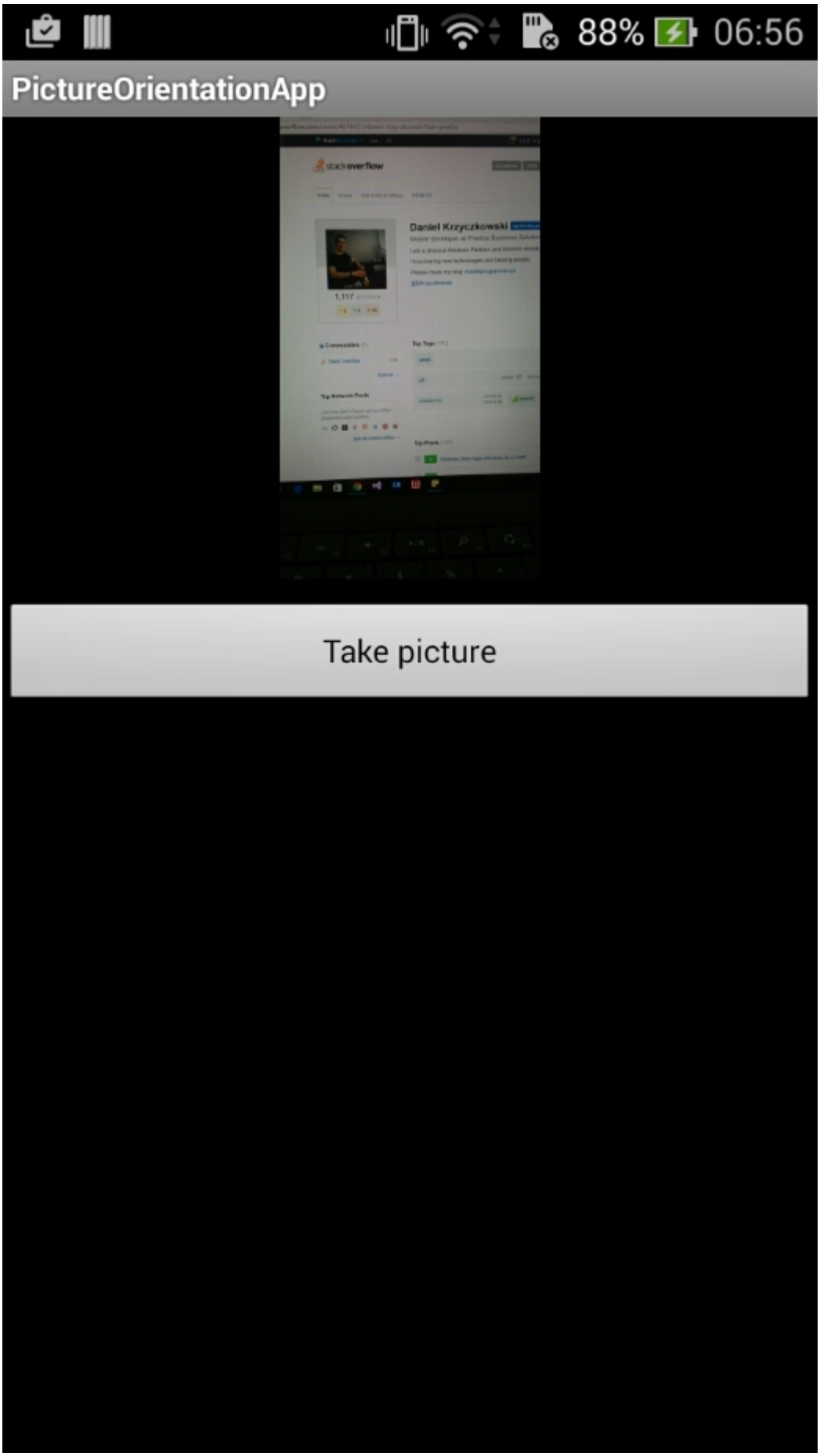
```
        }

        return resizedBitmap;
    }

    catch (IOException ex)
    {
        Console.WriteLine("An exception was thrown when reading exif from media
file...:" + ex.Message);
        return null;
    }
 }
```

6. Il metodo sopra deve essere invocato nel metodo OnActivityResult richiamato dopo che l'utente ha scattato l'immagine:

```
protected override void OnActivityResult(int requestCode, Result resultCode, Intent data)
 {
     base.OnActivityResult(requestCode, resultCode, data);

     if (requestCode == 1)
     {
         if (resultCode == Result.Ok)
         {
             data.GetMediaFileExtraAsync(this).ContinueWith(t =>
             {
                 using (Bitmap bmp = loadAndResizeBitmap(t.Result.Path))
                 {
                     if (bmp != null)
                     _takenPictureImageView.SetImageBitmap(bmp);
                 }

             }, TaskScheduler.FromCurrentSynchronizationContext());
         }
     }
 }
```

7. Avvia l'applicazione. Scatta una foto e vedi il risultato:


Automatyczny
Daniel Krzyczkowski
Mobile developer at Predica Business Solutions
I am a Univeral Windows Platform and Xamarin developer.
I love learing new technologies and helping people.
Please check my blog: mobileprogrammer.pl
@DKrzyczkowski
1,117 REPUTATION
Communities (1)
Stack Overflow
Edit list
Top Tags (111)
uwp
c#
xamarin
windows-10
android
Top Posts (117)
Windows Store Apps (Windows 8) vs UWP



Questo è tutto. Ora avrai tutte le foto scattate visualizzate con l'orientamento corretto.

Leggi Come correggere l'orientamento di un'immagine catturata dal dispositivo Android online: https://riptutorial.com/it/xamarin-android/topic/6683/come-correggere-l-orientamento-di-un-immagine-catturata-dal-dispositivo-android

Capitolo 6: Finestre di dialogo

Osservazioni

Impostazione del `Context` della finestra di dialogo

Quando si crea una `Dialog` da un `Activiy` possiamo usare `this` come il contesto.

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
```

Con `Fragments` usiamo la proprietà `Context` .

```
AlertDialog.Builder builder = new AlertDialog.Builder(Context);
```

Tipi di pulsanti

`SetNeutralButton()` può essere utilizzato per una semplice notifica e conferma che la notifica viene letta. `SetPositiveButton()` può essere utilizzato per una conferma: "Sei sicuro di voler eliminare questo elemento?" `SetNegativeButton()` serve per `SetNegativeButton()` la finestra di dialogo e annullare la sua azione.

Disabilita l'annullamento da backbutton

Se vogliamo essere sicuri che l'utente non possa ignorare la finestra di dialogo con il pulsante Indietro, possiamo chiamare `SetCanceable(false)` . Funziona solo per il pulsante Indietro.

Rotazione

Se lo schermo viene ruotato mentre una finestra di dialogo è visibile, verrà chiusa e le azioni di ok e annulla non verranno richiamate. Dovrai gestirlo all'interno della tua attività e mostrare nuovamente la finestra di dialogo dopo che l'attività è stata ricaricata.

Per `DialogFragment` ciò utilizzare invece un `DialogFragment` .

Examples

Finestra di dialogo di avviso

Creazione di una finestra di avviso

```
AlertDialog.Builder builder = new AlertDialog.Builder(Context);
builder.SetIcon(Resource.Drawable.Icon);
builder.SetTitle(title);
builder.SetMessage(message);
```

```
builder.SetNeutralButton("Neutral", (evt, args) => {
    // code here for handling the Neutral tap
});

builder.SetPositiveButton("Ok", (evt, args) => {
    // code here for handling the OK tap
});

builder.SetNegativeButton("Cancel", (evt, args) => {
    // code here for handling the Cancel tap
});

builder.SetCancelable(false);
builder.Show();
```

Leggi Finestre di dialogo online: https://riptutorial.com/it/xamarin-android/topic/2510/finestre-di-dialogo

Capitolo 7: Finestre di dialogo

Parametri

Metodo pubblico comunemente usato	Uso
SetTitle (String)	Imposta il titolo per la finestra di dialogo
SetIcon (Drawable)	Imposta icona per la finestra di avviso
SetMessage (stringa)	Imposta il messaggio da visualizzare.
SetNegativeButton (String, EventHandler)	Imposta un ascoltatore da richiamare quando viene premuto il pulsante negativo della finestra di dialogo.
SetPositiveButton (String, EventHandler)	Imposta un ascoltatore da richiamare quando viene premuto il pulsante positivo della finestra di dialogo.
SetNeutralButton (String, EventHandler)	Imposta un ascoltatore da richiamare quando viene premuto il pulsante neutro della finestra di dialogo.
SetOnCancelListener (IDialogInterfaceOnCancelListener)	Imposta il callback che verrà chiamato se la finestra di dialogo viene annullata.
SetOnDismissListener (IDialogInterfaceOnDismissListener)	Imposta il callback che verrà chiamato quando la finestra di dialogo viene chiusa per qualsiasi motivo.
Mostrare()	Crea un AlertDialog con gli argomenti forniti a questo builder e Dialog.Show è la finestra di dialogo.

Osservazioni

Requisiti

Spazio dei nomi: Android.App

Assembly: Mono.Android (in Mono.Android.dll)

Versioni di montaggio: 0.0.0.0

Costruttori pubblici

AlertDialog.Builder (Context): -

Costruttore che utilizza un contesto per questo builder e il AlertDialog che crea.

AlertDialog.Builder (Context, Int32): -

Costruttore che utilizza un contesto e un tema per questo builder e il AlertDialog che crea.

Uso di Material Design AlertDialog

Per utilizzare il moderno AlertDialog:

1. Installa supporto Libreria v7 AppCompat dai pacchetti NuGet
2. Sostituisci AlertDialog con Android.Support.V7.App.AlertDialog o aggiungi la seguente istruzione nella parte superiore per far risplendere il tuo dialogo.

```
        using AlertDialog = Android.Support.V7.App.AlertDialog;
```

Examples

AlertDialog

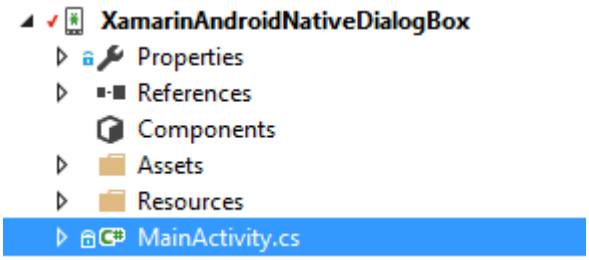
```
// 1. Instantiate an AlertDialog.Builder with its constructor
// the parameter this is the context (usually your activity)
AlertDialog.Builder builder = new AlertDialog.Builder(this);

// 2. Chain together various setter methods to set the dialog characteristics
builder.SetMessage(Resource.String.dialog_message)
       .SetTitle(Resource.String.dialog_title);

// 3. Get the AlertDialog from create()
AlertDialog dialog = builder.Create();

dialog.Show();
```

Esempio di dialogo di avviso semplice

Creeremo una semplice finestra di dialogo degli avvisi in Xamarin.Android

Considerando ora che hai seguito la guida introduttiva dalla documentazione.

Devi avere la struttura del progetto in questo modo:



La tua attività principale deve essere simile a questa:

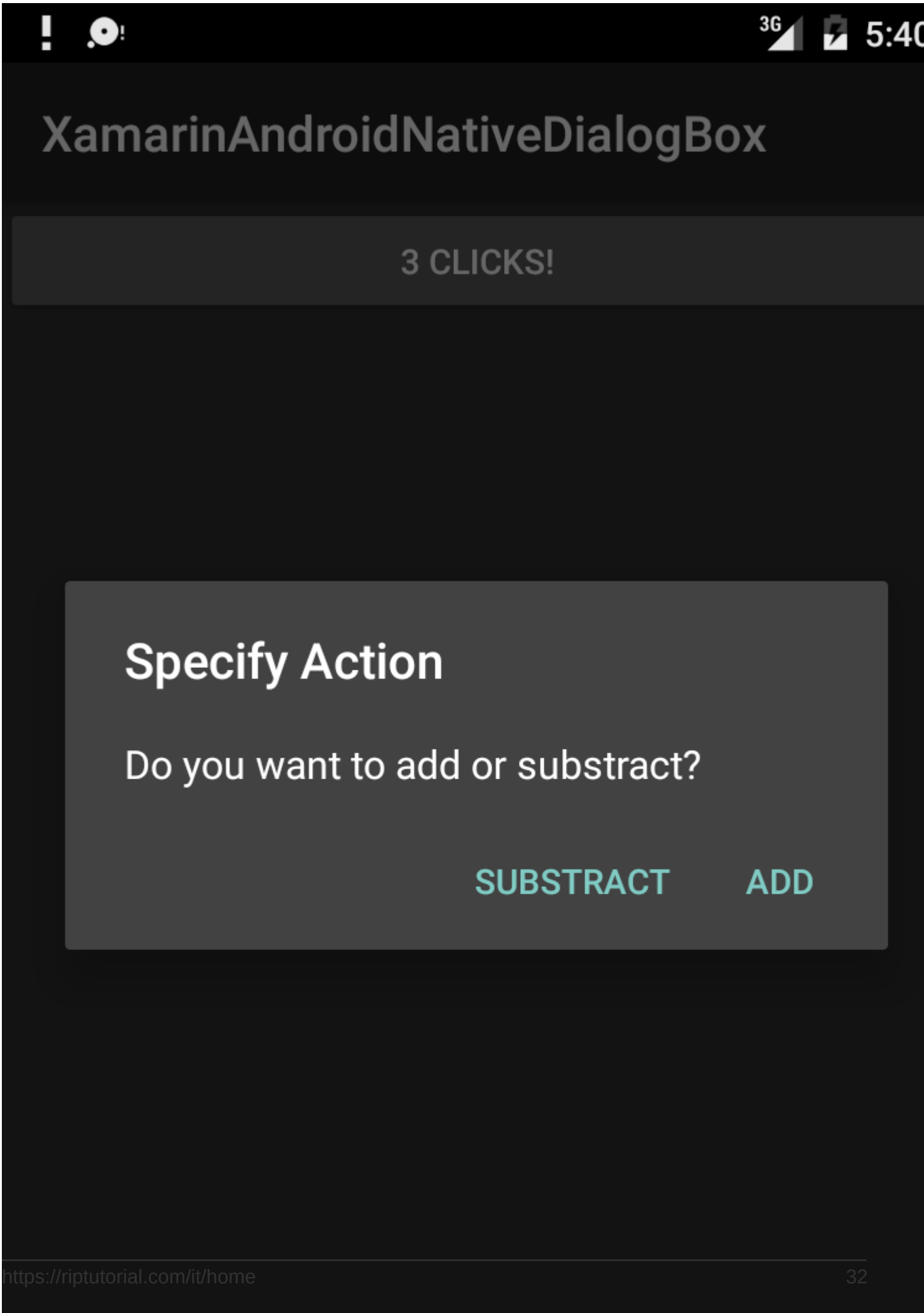
```
public class MainActivity : Activity
{
int count = 1;

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

// Set our view from the "main" layout resource
SetContentView(Resource.Layout.Main);

// Get our button from the layout resource,
// and attach an event to it
Button button = FindViewById<Button>(Resource.Id.MyButton);

button.Click += delegate { button.Text = string.Format("{0} clicks!", count++); };
}
}
```

Ora, cosa dovremmo fare, invece di aggiungerne uno al contatore sul clic del pulsante, chiederemo all'utente se desidera aggiungerne o sottrarre uno in una semplice finestra di dialogo degli avvisi

E al clic del pulsante Positivo o negativo effettueremo l'azione.

```
button.Click += delegate {
AlertDialog.Builder alert = new AlertDialog.Builder(this);
alert.SetTitle("Specify Action");
alert.SetMessage("Do you want to add or substract?");

alert.SetPositiveButton("Add", (senderAlert, args) =>
{
count++;
button.Text = string.Format("{0} clicks!", count);
 });

 alert.SetNegativeButton("Substract", (senderAlert, args) =>
 {
 count--;
 button.Text = string.Format("{0} clicks!", count);
 });

 Dialog dialog = alert.Create();
     dialog.Show();
};
```

immagine dello schermo:


XamarinAndroidNativeDialogBox
3 CLICKS!
Specify Action
Do you want to add or substract?
SUBSTRACT
ADD

https://riptutorial.com/it/xamarin-android/topic/4367/finestre-di-dialogo

Capitolo 8: ListView personalizzato

Examples

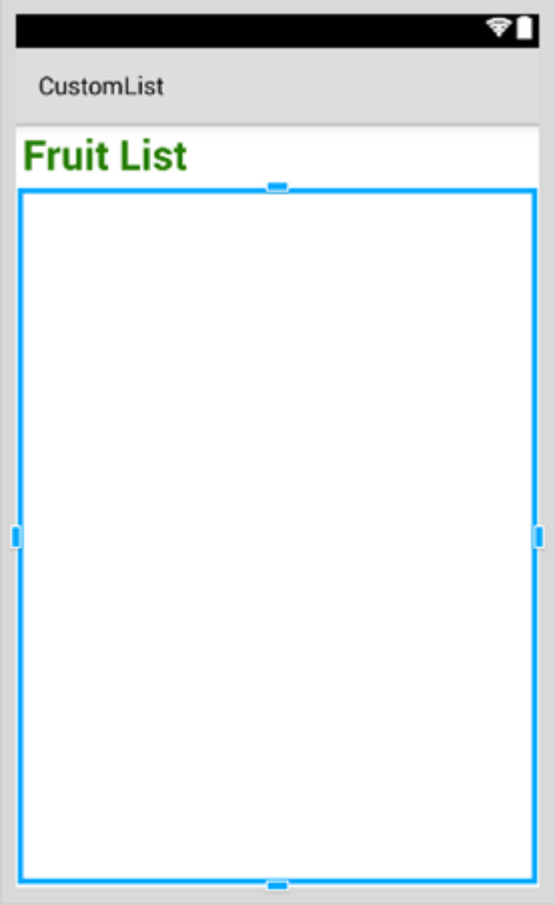
ListView personalizzato comprende le righe progettate secondo le esigenze degli utenti.

Per il layout sopra il tuo file customrow.axml è come mostrato di seguito

```
    <?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:padding="8dp">
    <ImageView
        android:id="@+id/Image"
        android:layout_width="80dp"
        android:layout_height="80dp"
        android:layout_alignParentLeft="true"
        android:layout_marginRight="8dp"
        android:src="@drawable/icon" />
    <TextView
        android:id="@+id/Text1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignTop="@id/Image"
        android:layout_toRightOf="@id/Image"
        android:layout_marginTop="5dip"
        android:text="This is Line1"
        android:textSize="20dip"
        android:textStyle="bold" />
    <TextView
        android:id="@+id/Text2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/Text1"
        android:layout_marginTop="1dip"
```

```
        android:text="This is line2"
        android:textSize="15dip"
        android:layout_toRightOf="@id/Image" />
</RelativeLayout>
```

Quindi puoi progettare il tuo main.axml, che contiene una textview per l'intestazione e un listview.



Spero che sia facile ...

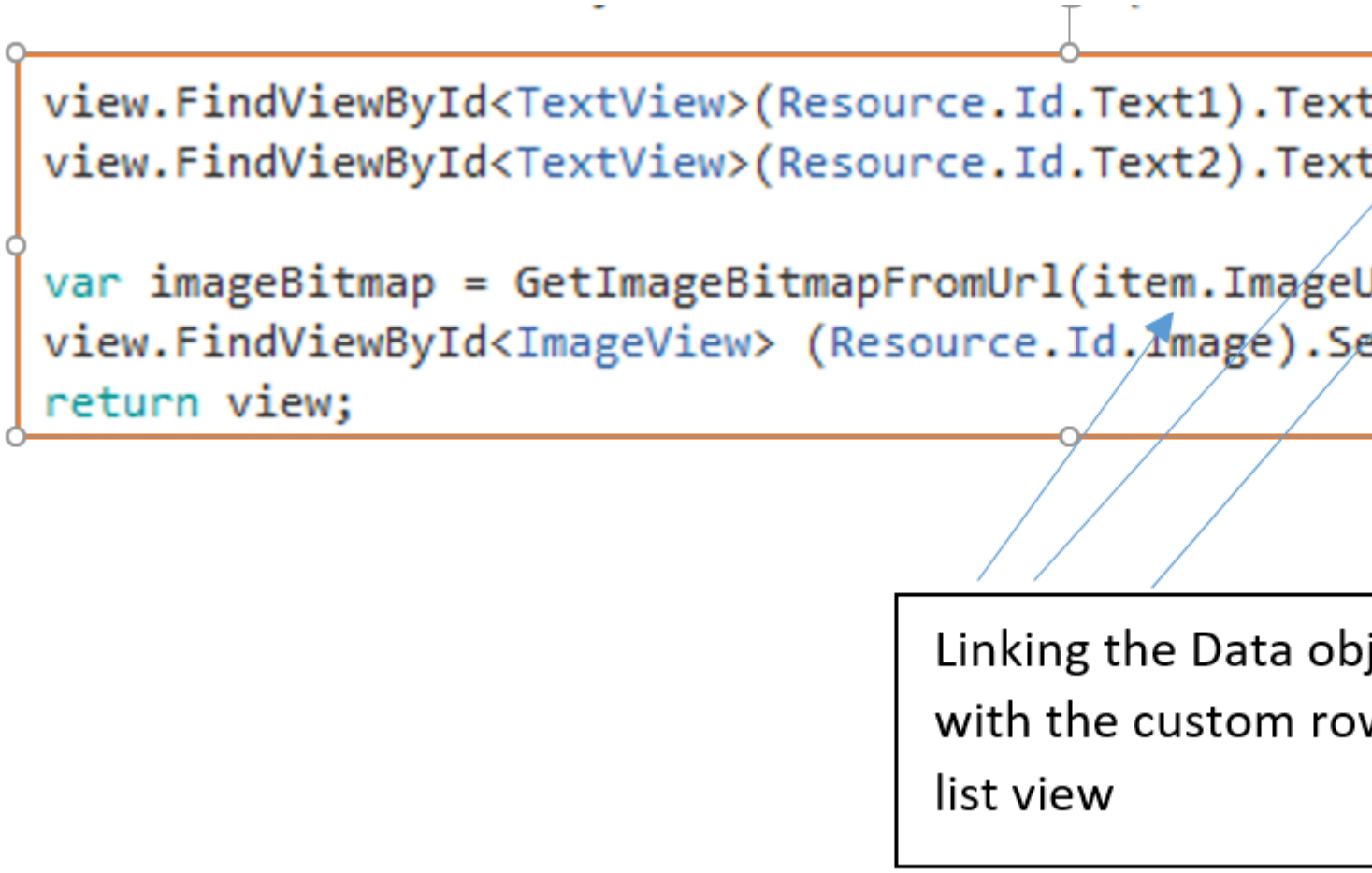
Quindi crea la classe Data.cs che rappresenterà i tuoi oggetti riga

```
public class Data
{
    public string Heading;
    public string SubHeading;
    public string ImageURI;

    public Data ()
    {
        Heading = "";
        SubHeading = "";
        ImageURI = "";
    }
}
```

Successivamente hai bisogno della classe DataAdapter.cs, gli adattatori collegano i tuoi dati con la vista sottostante

```
public class DataAdapter : BaseAdapter<Data> {

    List<Data> items;

    Activity context;
    public DataAdapter(Activity context, List<Data> items)
        : base()
    {
        this.context = context;
        this.items = items;
    }
    public override long GetItemId(int position)
    {
        return position;
    }
    public override Data this[int position]
    {
        get { return items[position]; }
    }
    public override int Count
    {
        get { return items.Count; }
    }
    public override View GetView(int position, View convertView, ViewGroup parent)
    {
        var item = items[position];
        View view = convertView;
        if (view == null) // no view to re-use, create new
            view = context.LayoutInflater.Inflate(Resource.Layout.CustomRow, null);

        view.FindViewById<TextView>(Resource.Id.Text1).Text = item.Heading;
        view.FindViewById<TextView>(Resource.Id.Text2).Text = item.SubHeading;

        var imageBitmap = GetImageBitmapFromUrl(item.ImageURI);
        view.FindViewById<ImageView> (Resource.Id.Image).SetImageBitmap (imageBitmap);
        return view;
    }

    private Bitmap GetImageBitmapFromUrl(string url)
    {
        Bitmap imageBitmap = null;
        if(!(url=="null"))
            using (var webClient = new WebClient())
            {
                var imageBytes = webClient.DownloadData(url);
                if (imageBytes != null && imageBytes.Length > 0)
                {
                    imageBitmap = BitmapFactory.DecodeByteArray(imageBytes, 0,
imageBytes.Length);
                }
            }

        return imageBitmap;
    }

}
```

La parte più importante è all'interno della funzione GetView, qui è dove si collega l'oggetto alla riga personalizzata.



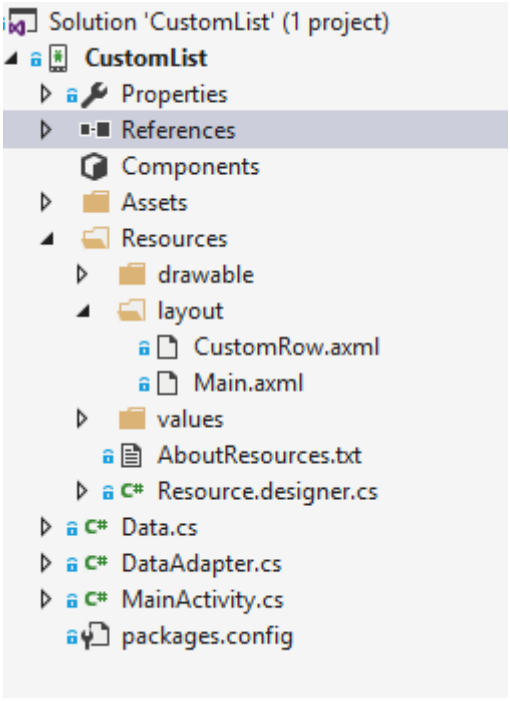
GetImageBitmapFromUrl non fa parte del dataadapter ma l'ho messo qui per semplicità.

Finalmente arriviamo al MainActivity.cs

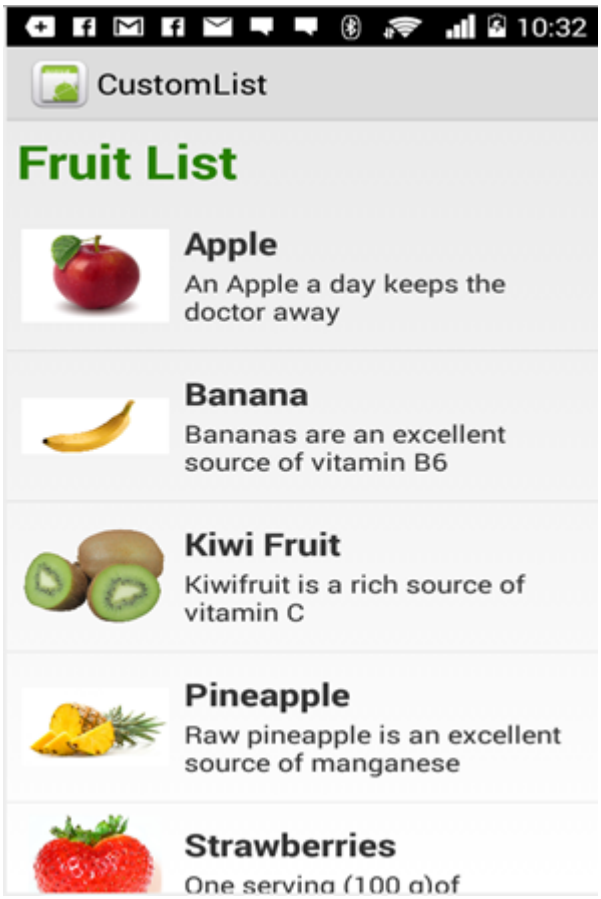
```
public class MainActivity : Activity
{

    ListView listView;

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

        // Set our view from the "main" layout resource
        SetContentView (Resource.Layout.Main);
        listView = FindViewById<ListView>(Resource.Id.List);

        List<Data> myList = new List<Data> ();

        Data obj = new Data ();
        obj.Heading = "Apple";
        obj.SubHeading = "An Apple a day keeps the doctor away";
        obj.ImageURI =
"http://www.thestar.com/content/dam/thestar/opinion/editorials/star_s_view_/2011/10/12/an_apple_a_day_r



        myList.Add (obj);

        Data obj1 = new Data();
        obj1.Heading = "Banana";
```

```
        obj1.SubHeading = "Bananas are an excellent source of vitamin B6 ";
        obj1.ImageURI =
"http://www.bbcgoodfood.com/sites/bbcgoodfood.com/files/glossary/banana-crop.jpg";

        myList.Add(obj1);

        Data obj2 = new Data();
        obj2.Heading = "Kiwi Fruit";
        obj2.SubHeading = "Kiwifruit is a rich source of vitamin C";
        obj2.ImageURI = "http://www.wiffens.com/wp-content/uploads/kiwi.png";

        myList.Add(obj2);

        Data obj3 = new Data();
        obj3.Heading = "Pineapple";
        obj3.SubHeading = "Raw pineapple is an excellent source of manganese";
        obj3.ImageURI =
"http://www.medicalnewstoday.com/images/articles/276/276903/pineapple.jpg";

        myList.Add(obj3);

        Data obj4 = new Data();
        obj4.Heading = "Strawberries";
        obj4.SubHeading = "One serving (100 g)of strawberries contains approximately 33
kilocalories";
        obj4.ImageURI = "https://ecs3.tokopedia.net/newimg/product-
1/2014/8/18/5088/5088_8dac78de-2694-11e4-8c99-6be54908a8c2.jpg";

        myList.Add (obj4);
        listView.Adapter = new DataAdapter(this,myList);

    }
```

La tua struttura del progetto finale è come mostrato di seguito.



Se tutto va bene dovresti vedere l'output come mostrato



Leggi ListView personalizzato online: https://riptutorial.com/it/xamarin-android/topic/6406/listview-personalizzato

Capitolo 9: Pubblicare il tuo APK Xamarin.Android

introduzione

Questo argomento mostra informazioni su come preparare la tua app Xamarin.Android per la modalità di rilascio e su come ottimizzarla.

Examples

Preparazione dell'APK in Visual Studio

Hai finito la tua app, testato in modalità di debug e funziona perfettamente. Ora, vuoi prepararlo per la pubblicazione nel Google Play Store.

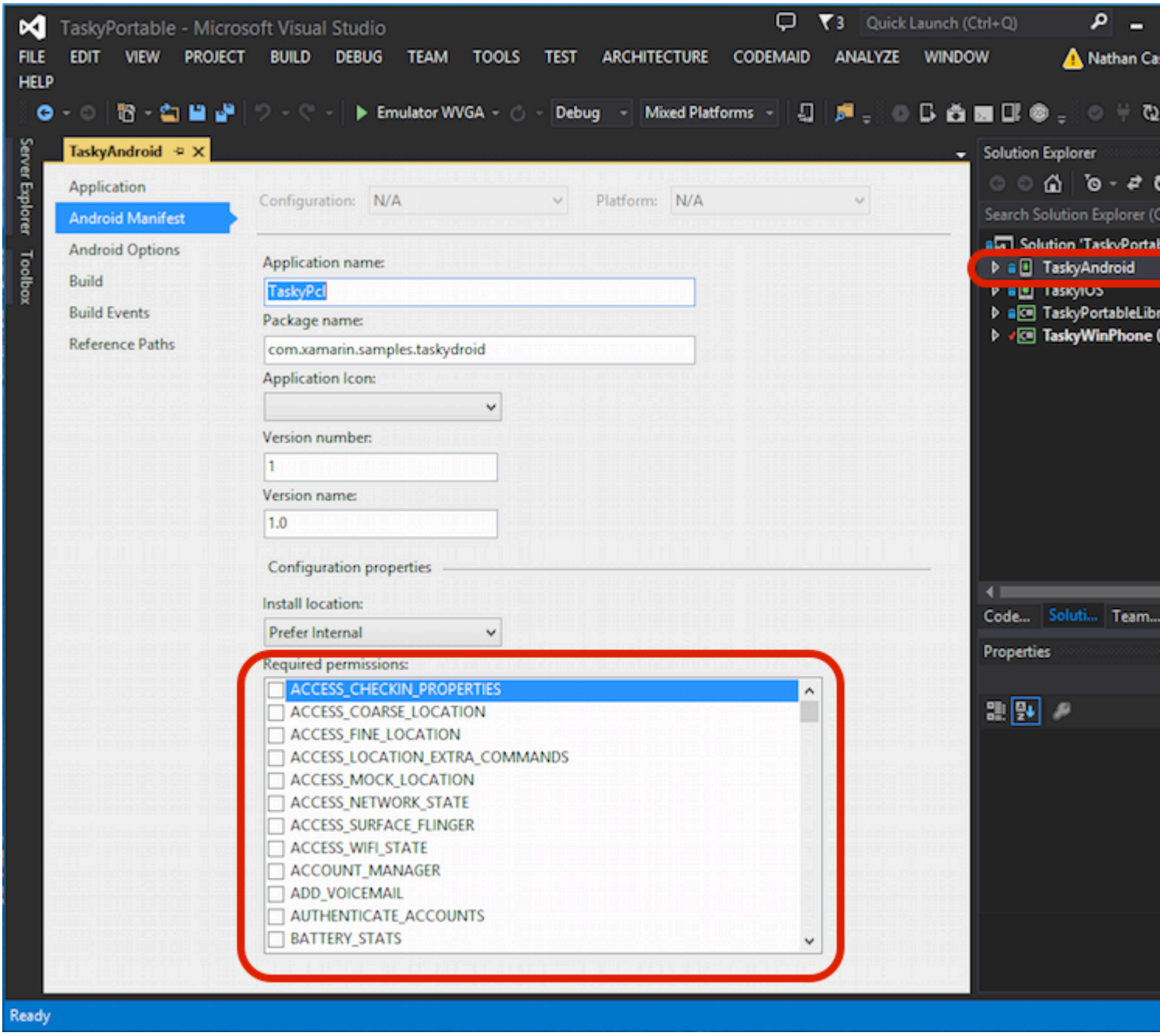
La documentazione di Xamarin fornisce buone informazioni qui:

https://developer.xamarin.com/guides/android/deployment,_testing,_and_metrics/publishing_an_applicati

---

Manifest Android

Innanzitutto, in Visual Studio, fai clic con il pulsante destro del mouse sul progetto Xamarin.Android in Esplora soluzioni e seleziona Proprietà. Quindi, vai alla scheda Manifest Android, per vedere questa schermata:



A differenza di Android Studio o Eclipse, non è necessario impostare il file AndroidManifest.xml scrivendo; Xamarin e Visual Studio lo fanno per te. Attività, BroadcastReceivers e Servizi sono inseriti in Android Manifest dichiarando attributi specifici nelle loro classi .

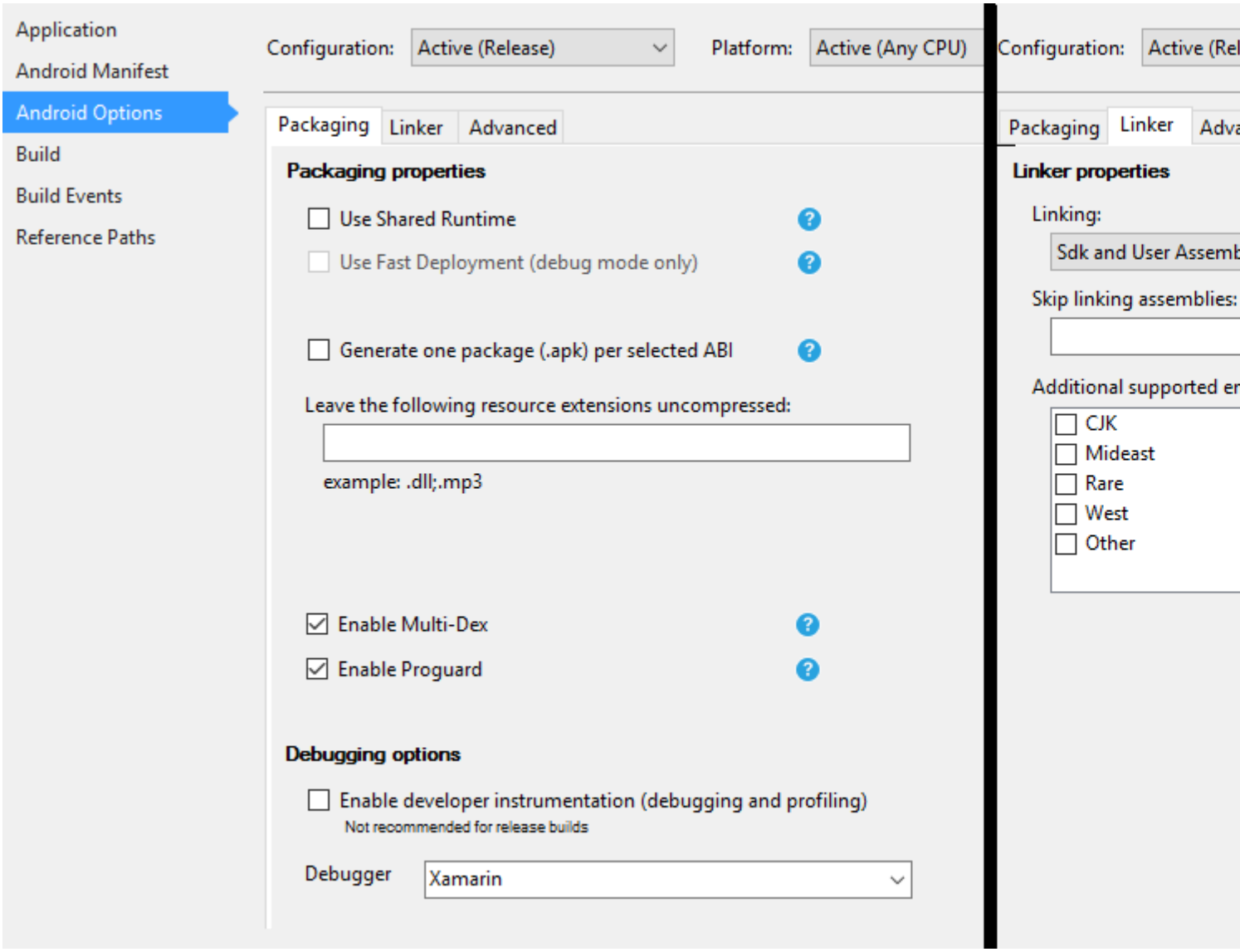
In questa schermata, le opzioni sono:

- **Nome dell'applicazione** : questo è il nome dell'app che sarà visibile all'utente.
- **Nome del pacchetto** : questo è il nome del pacchetto. Deve essere univoco, il che significa che non deve utilizzare lo stesso nome di pacchetto di altre app nel Google Play Store.
- **Icona dell'applicazione** : questa è l'icona che sarà visibile all'utente, equivalente al @ drawable / ic_launcher utilizzato nei progetti Android Studio o Eclipse.
- **Numero di versione** : il numero di versione è utilizzato da Google Play per il controllo della versione. Quando si desidera pubblicare un APK per una versione aggiornata della propria app, è necessario aggiungere 1 a questo numero per ogni nuovo aggiornamento.
- **Nome versione** : questo è il nome della versione che verrà visualizzato all'utente.

- **Posizione di installazione** : determina dove verrà installato l'APK, nella memoria del dispositivo o nella scheda SD.
- **Autorizzazioni richieste** : qui puoi determinare quali autorizzazioni sono necessarie per la tua app.

Opzioni Android

Nella schermata sottostante, puoi configurare le opzioni del compilatore. L'uso delle opzioni giuste qui può ridurre molto la dimensione del tuo APK e anche prevenire errori.



- **Configurazione** : **attiva (rilascio)** .
- **Piattaforma** : **attiva (qualsiasi CPU)** . Questi sono necessari per creare il tuo APK per Google Play Store. Se la configurazione è impostata su debug, non sarà accettata da Google Play.
- **Usa runtime condiviso** : **falso** . Se si imposta su true, l'APK utilizzerà Mono Runtime per l'esecuzione. Mono Runtime viene installato automaticamente durante il debug tramite USB, ma non nel Release APK. Se Mono Runtime non è installato nel dispositivo e questa opzione è impostata su true nell'APK di rilascio, l'app si arresta in modo anomalo.
- **Genera un pacchetto (.apk) per ogni ABI selezionato** : **falso** . Crea il tuo APK per quante

più piattaforme possibili, per ragioni di compatibilità.
- **Attiva Multi-Dex** : **true** , ma puoi impostarlo su false se la tua app non è molto complessa (cioè ha meno di 65536 metodi, vedi qui).
- **Abilita Proguard** : **vero** . Ciò abilita lo strumento Proguard che nasconde il codice Java nella tua app. Si noti che non si applica al codice .NET; se vuoi offuscare il codice .NET, devi usare Dotfuscator . Maggiori informazioni su Proguard per Xamarin.Android possono essere trovate qui .
- **Abilita la strumentazione per sviluppatori (debugging e profiling)** : **false** per Release APK.
- **Collegamento** : **SDK e gruppi utente** . In questo modo Xamarin Linker rimuoverà tutte le classi non utilizzate dall'SDK e dal codice, riducendo la dimensione dell'APK.

Importante

Xamarin.Linker può talvolta rimuovere classi che non sembrano essere utilizzate dal codice, specialmente se si trovano nel Core del progetto (libreria PCL). Per evitare ciò, è possibile impostare il collegamento su "Solo assembly Sdk" o utilizzare l'attributo Preserve nelle classi, ad esempio:

PreserveAttribute.cs

```
namespace My_App_Core.Models
{
    public sealed class PreserveAttribute : System.Attribute
    {
        public bool AllMembers;
        public bool Conditional;
    }
}
```

In una classe:

```
using System;

namespace My_App_Core.Models
{
    [Preserve(AllMembers = true)]
    public class ServiceException : Exception
    {
        public int errorCode;

        [Preserve(AllMembers = true)]
        public ServiceException() { }

        [Preserve(AllMembers = true)]
        public ServiceException(int errorCode)
        {
            this.errorCode = errorCode;
        }
    }
}
```

- **Architetture supportate** : **seleziona tutto** , per motivi di compatibilità.

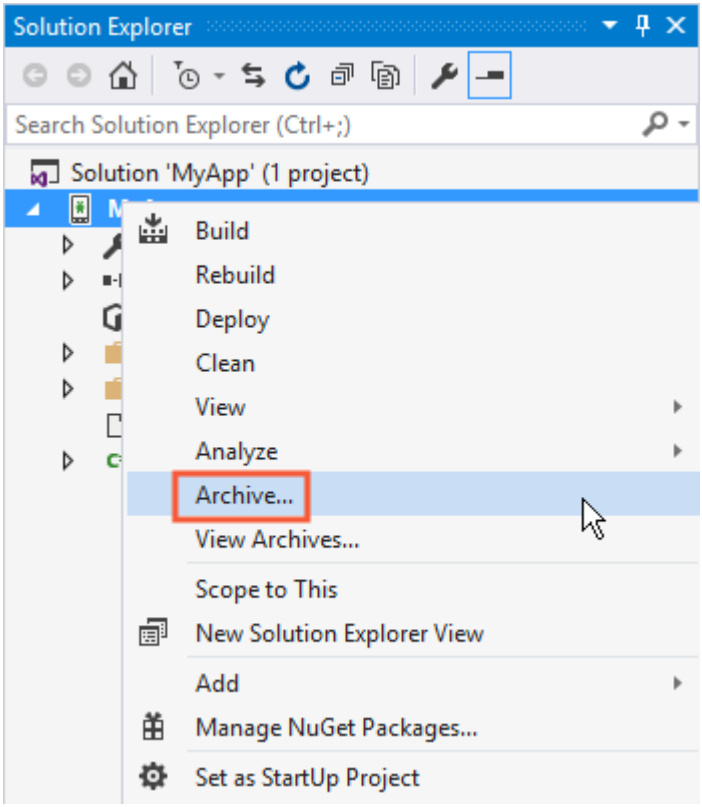
Dopo aver configurato tutto, ricompilare il progetto per assicurarsi che si sviluppi correttamente.

Creazione dell'APK per la modalità di rilascio

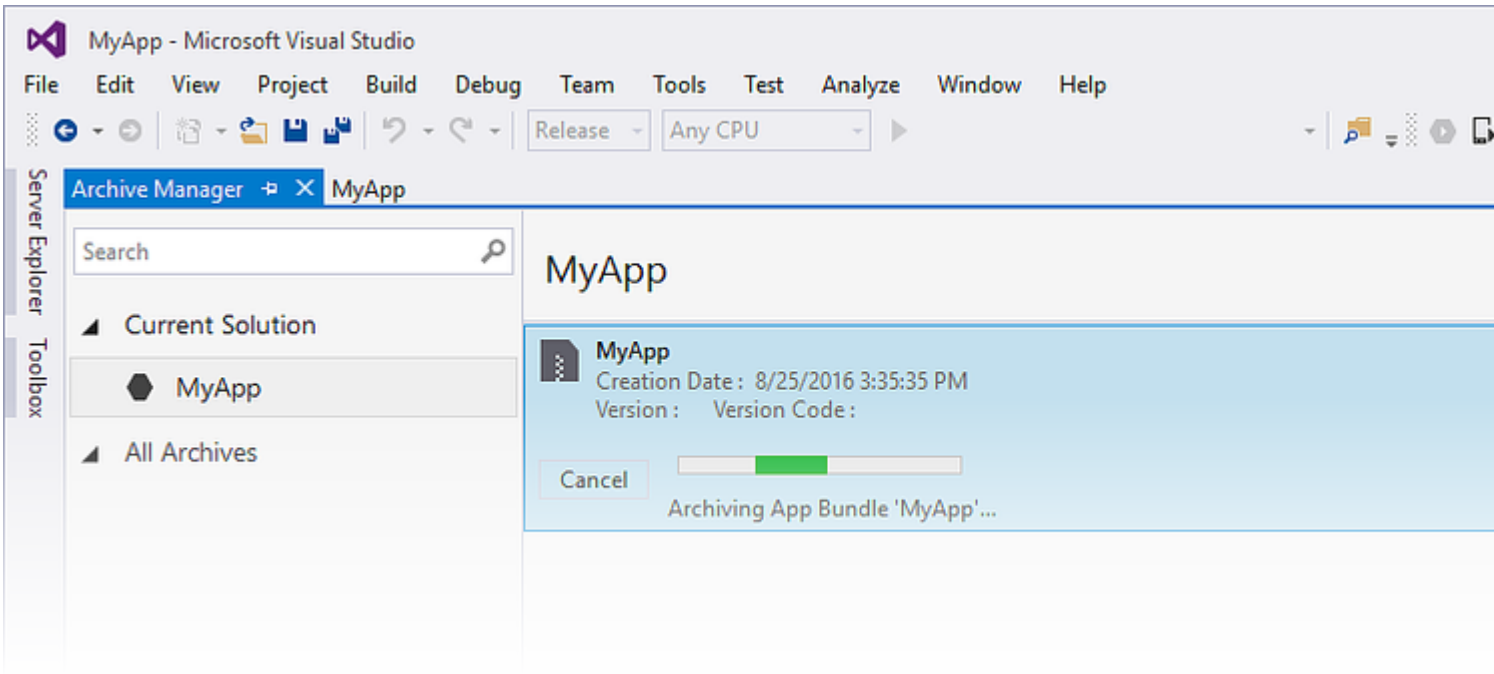
Hai finito di configurare il tuo progetto Android per la versione. Il tutorial qui sotto mostra come generare l'APK in Visual Studio. Un tutorial completo dalla documentazione di Xamarin può essere trovato qui:

https://developer.xamarin.com/guides/android/deployment,_testing,_and_metrics/publishing_an_applicati
_signing_the_android_application_package/

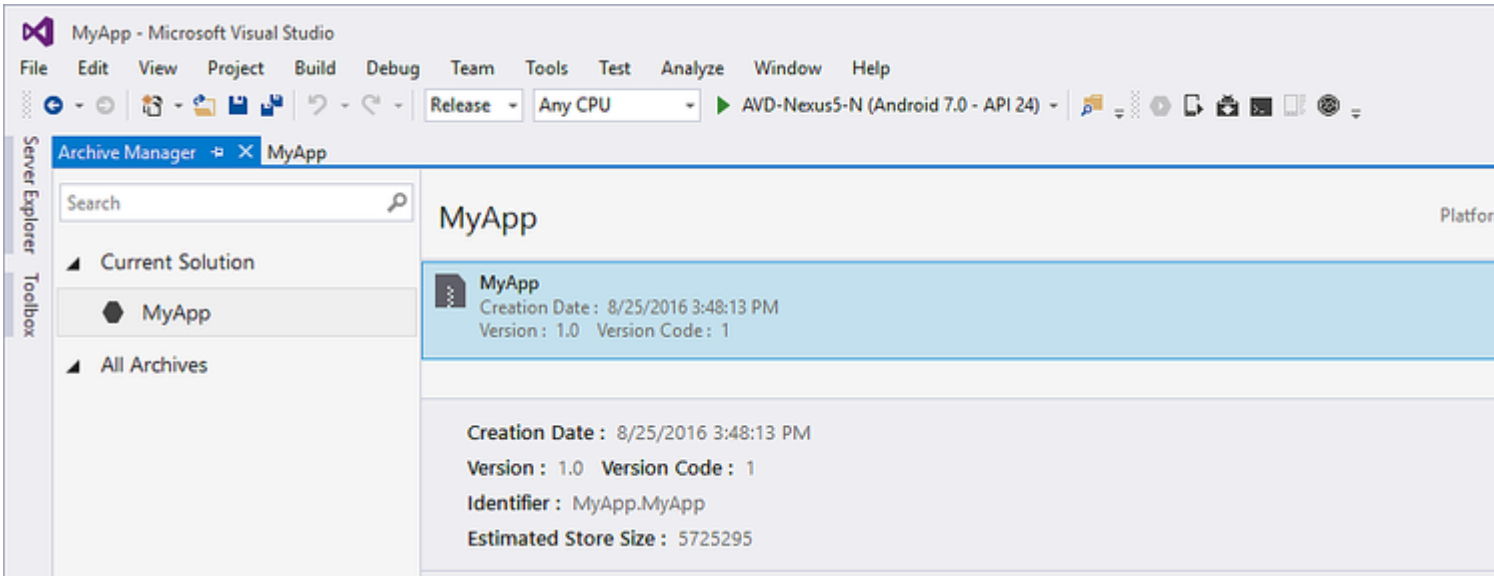
Per creare il file APK, fare clic con il tasto destro del mouse sul progetto Xamarin.Android in Solution Explorer e selezionare Archivia ...



Questo aprirà il gestore degli archivi e inizierà l'archiviazione del progetto, preparandosi a creare il file APK.

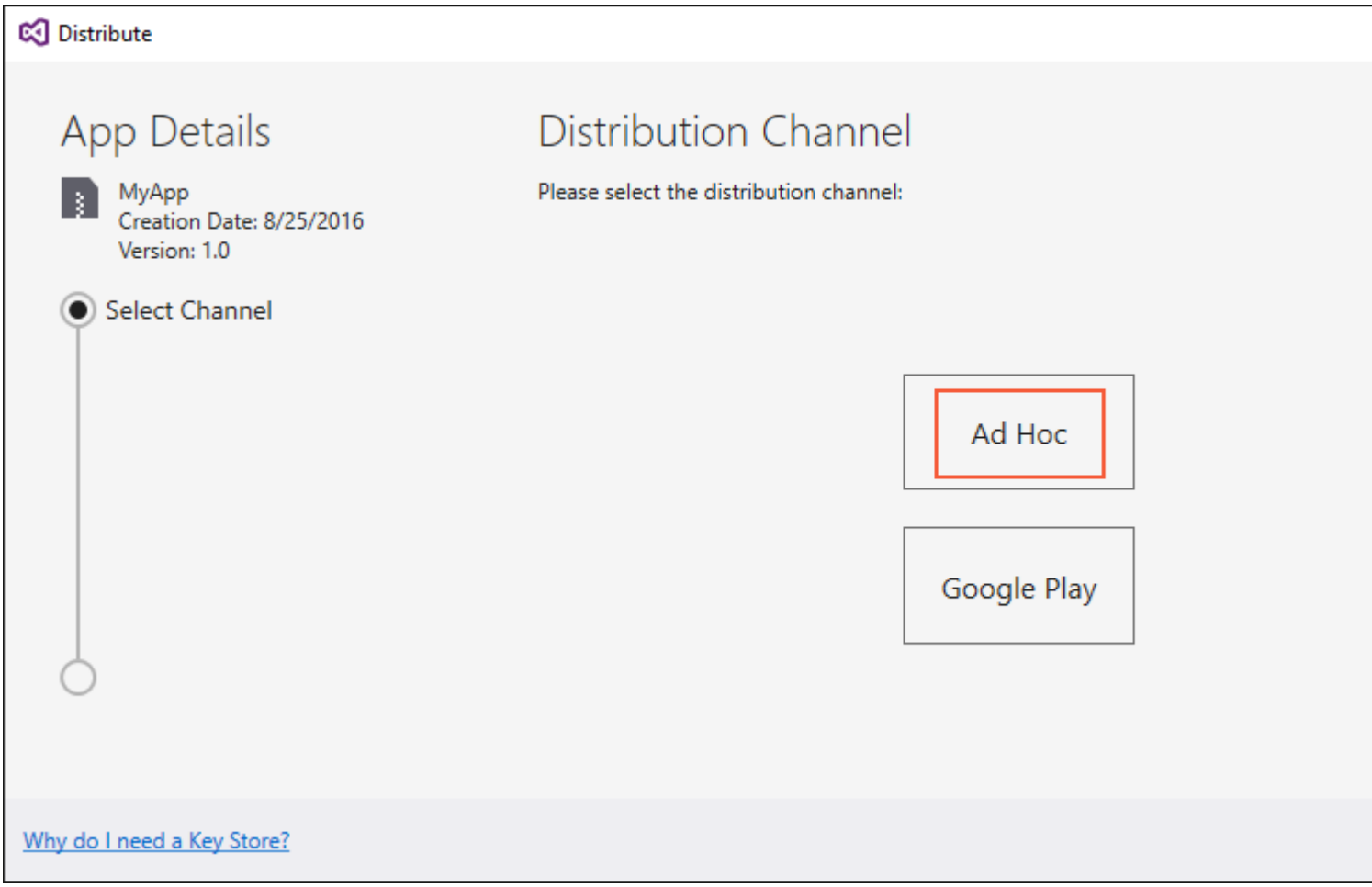


Al termine dell'archiviazione del progetto, fai clic su Distribuisci ... per procedere.

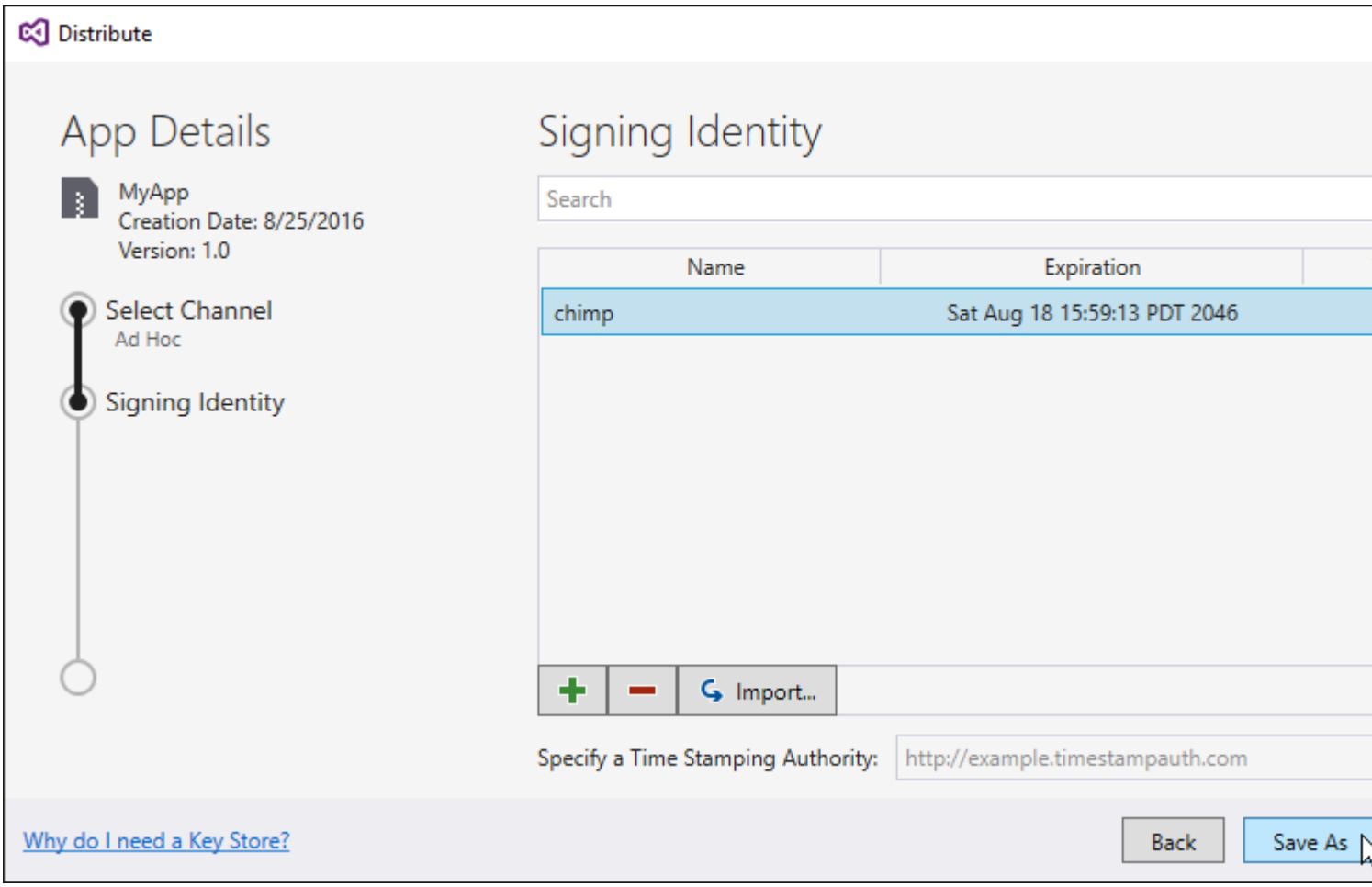


La schermata Distribuisci ti presenterà due opzioni: Ad-hoc e Google Play. Il primo creerà un APK e lo salverà sul tuo computer. Il secondo pubblicherà direttamente l'app su Google Play.

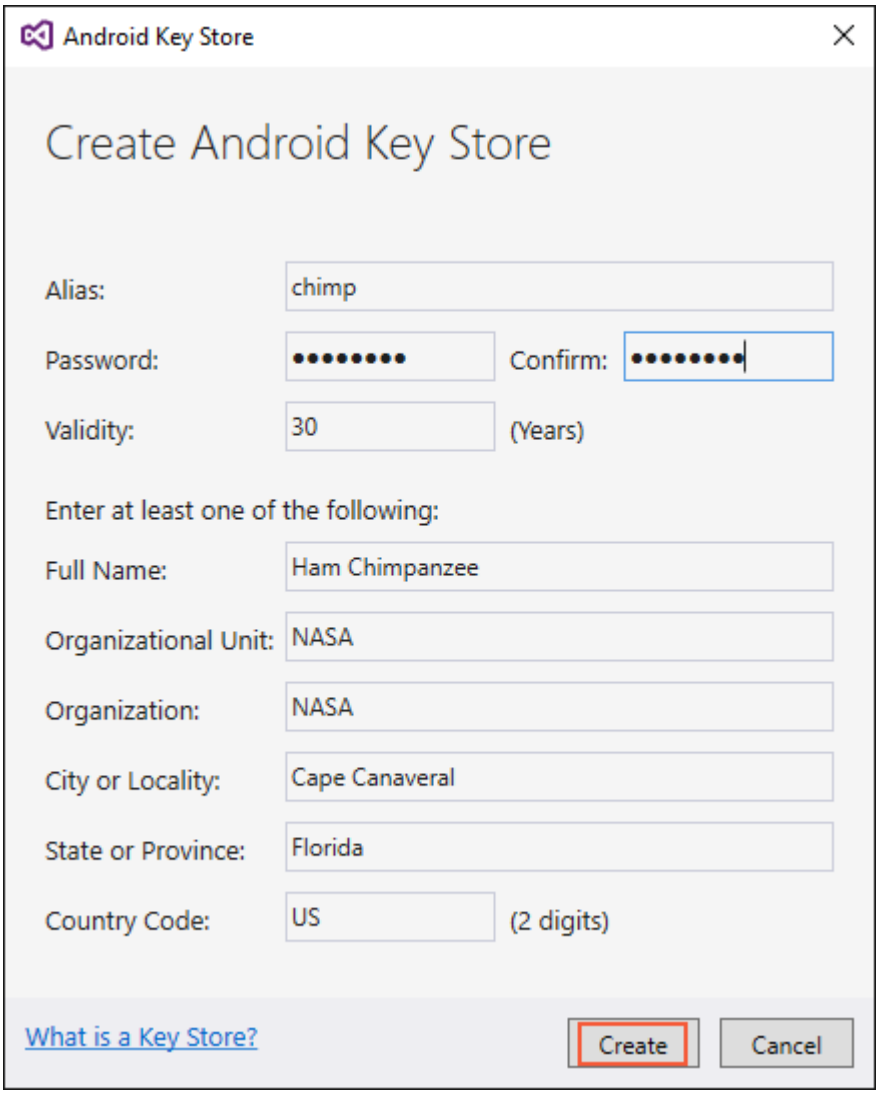
La scelta del primo è consigliata, quindi puoi testare l'APK in altri dispositivi, se lo desideri.



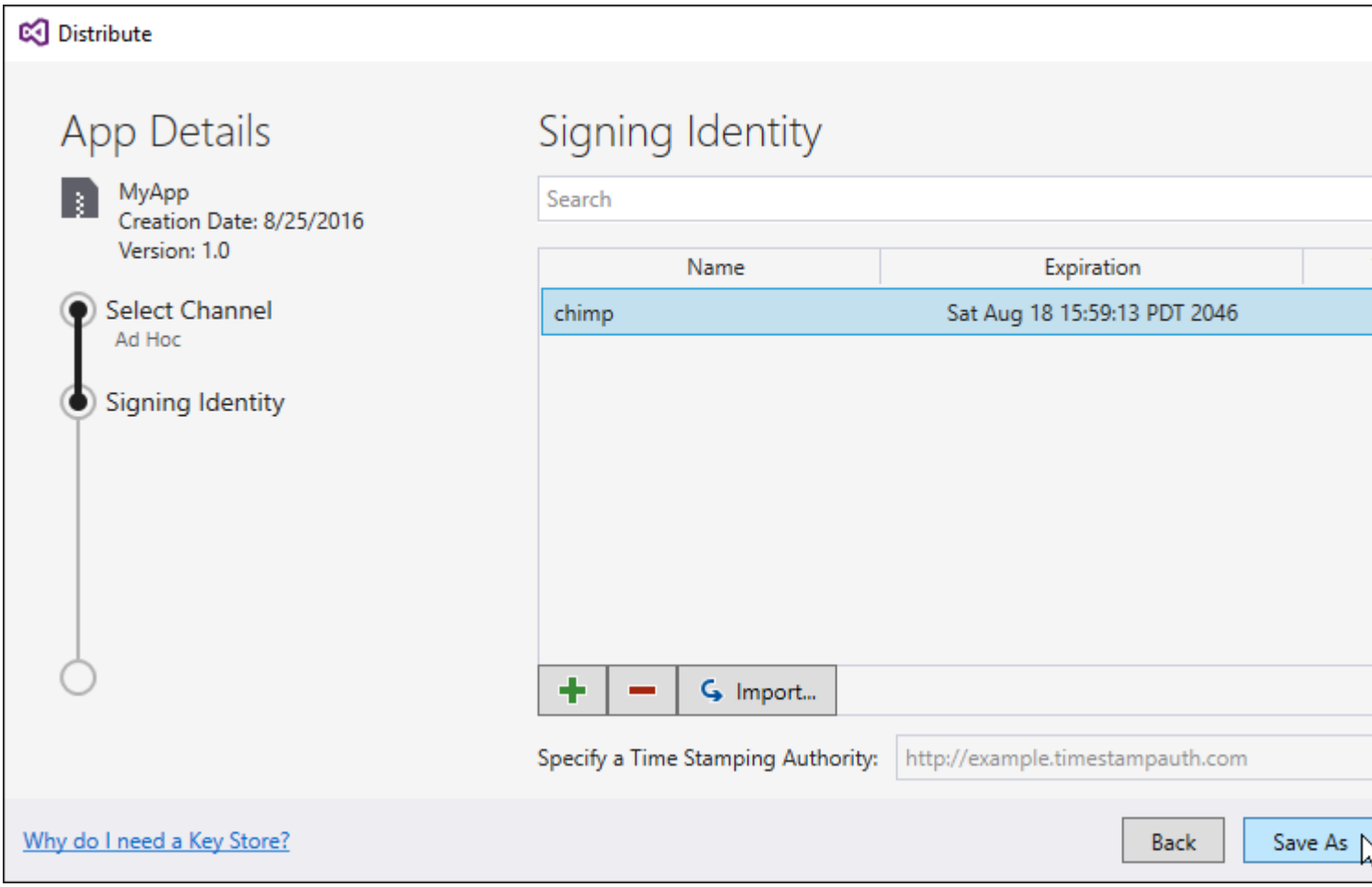
Nella schermata seguente è necessario un Key Store Android per firmare l'APK. Se ne hai già uno, puoi usarlo facendo clic su Importa ...; se non lo fai, puoi creare un nuovo Key Store Android facendo clic su +.

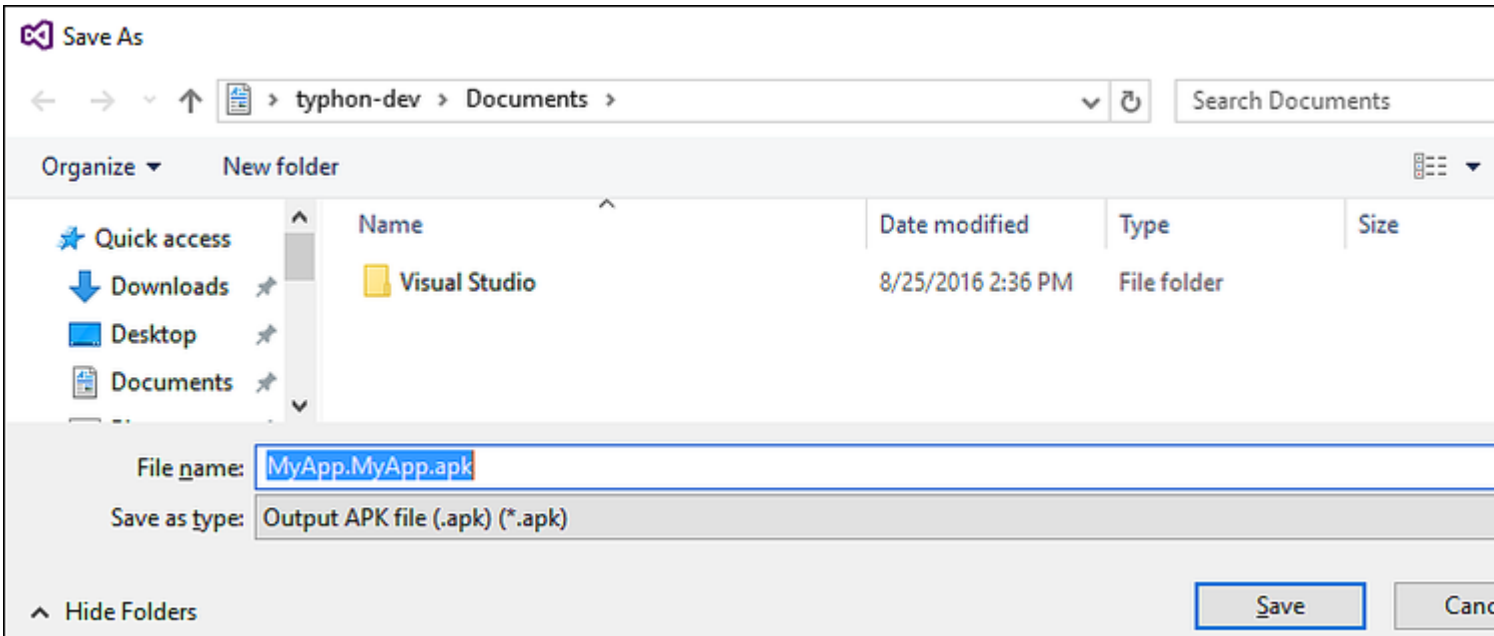


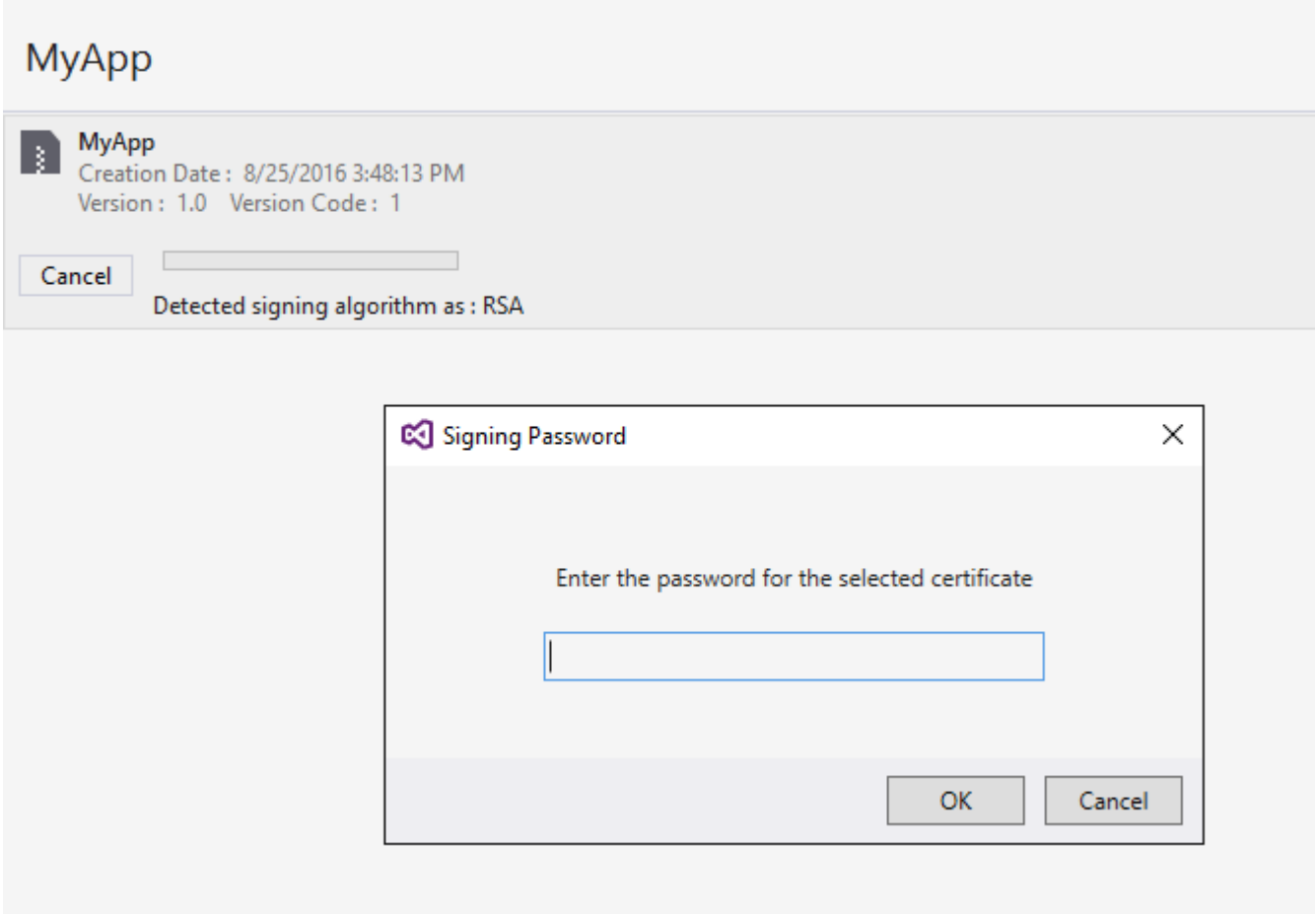
Creazione di una nuova schermata di Key Store Android:



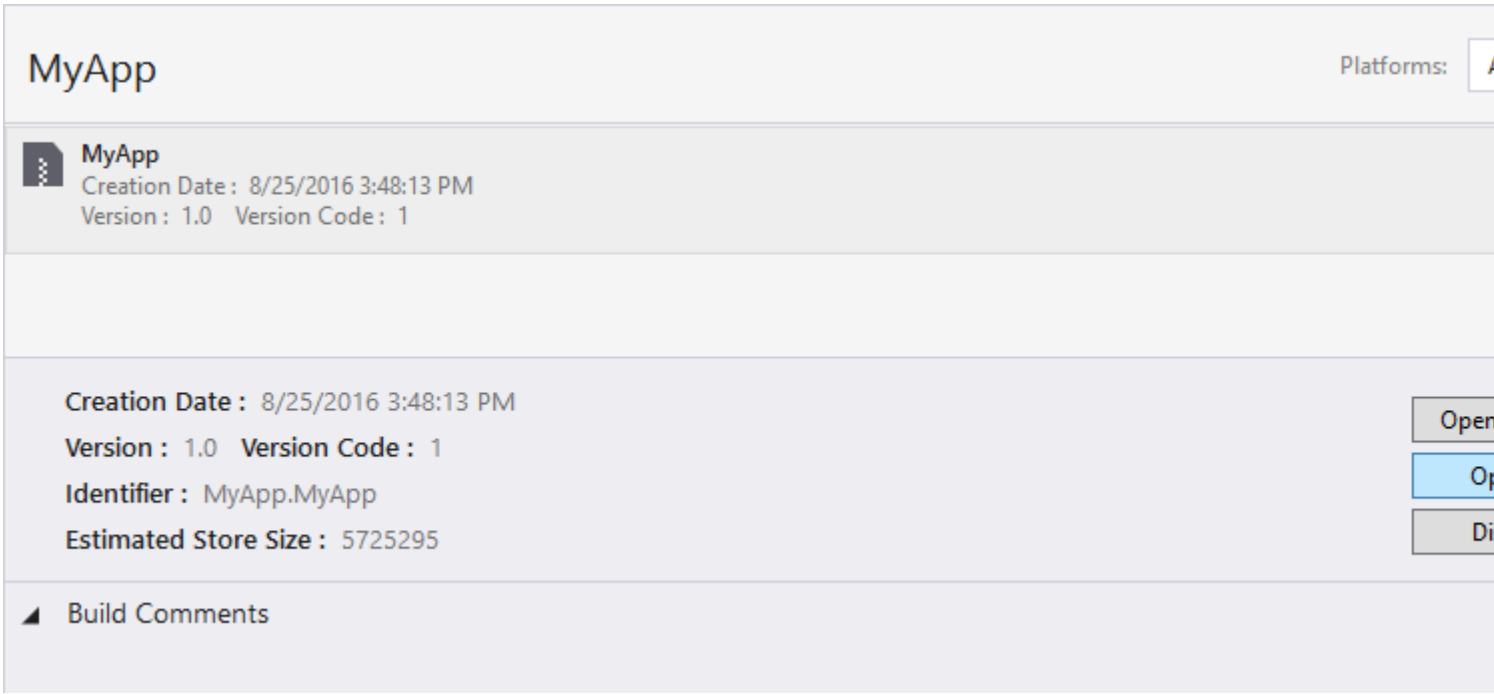
Per creare l'APK, fare clic su Salva con nome. Potrebbe essere richiesto di digitare la password dell'archivio chiavi.

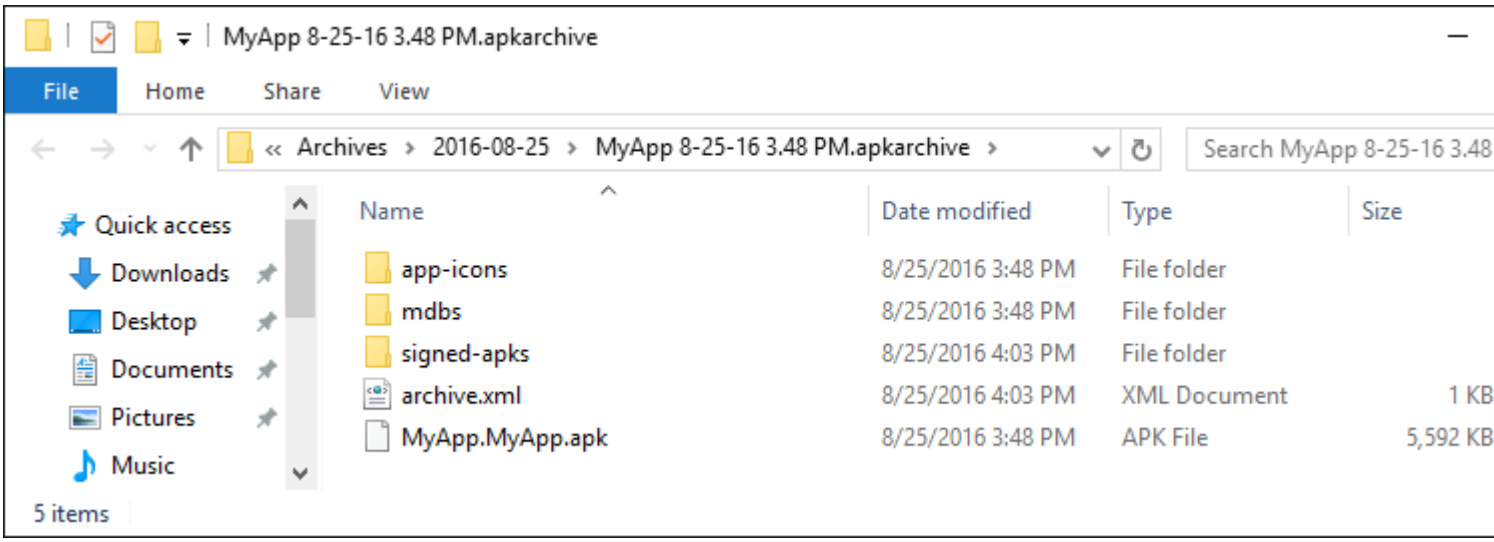

Distribute
App Details
MyApp
Creation Date: 8/25/2016
Version: 1.0
Select Channel
Ad Hoc
Signing Identity
Signing Identity
Search
Name
Expiration
chimp
Sat Aug 18 15:59:13 PDT 2046
Import...
Specify a Time Stamping Authority:
http://example.timestampauth.com
Why do I need a Key Store?
Back
Save As


Save As
typhon-dev
Documents
Search Documents
Organize
New folder
Quick access
Downloads
Desktop
Documents
Name
Date modified
Type
Size
Visual Studio
8/25/2016 2:36 PM
File folder
File name:
MyApp.MyApp.apk
Save as type:
Output APK file (.apk) (*.apk)
Hide Folders
Save



Al termine, puoi fare clic su Apri cartella nella schermata Archivi per visualizzare il file APK generato.





Abilitazione di MultiDex nel tuo APK Xamarin.Android

MultiDex è una libreria nell'APK Android che consente all'app di avere più di 65.536 metodi.

Gli APK Android hanno file eseguibili Dalvik (.dex) che contengono i bytecode generati dal codice Java. Ogni file .dex può contenere fino a 65.536 metodi (2 ^ 16).

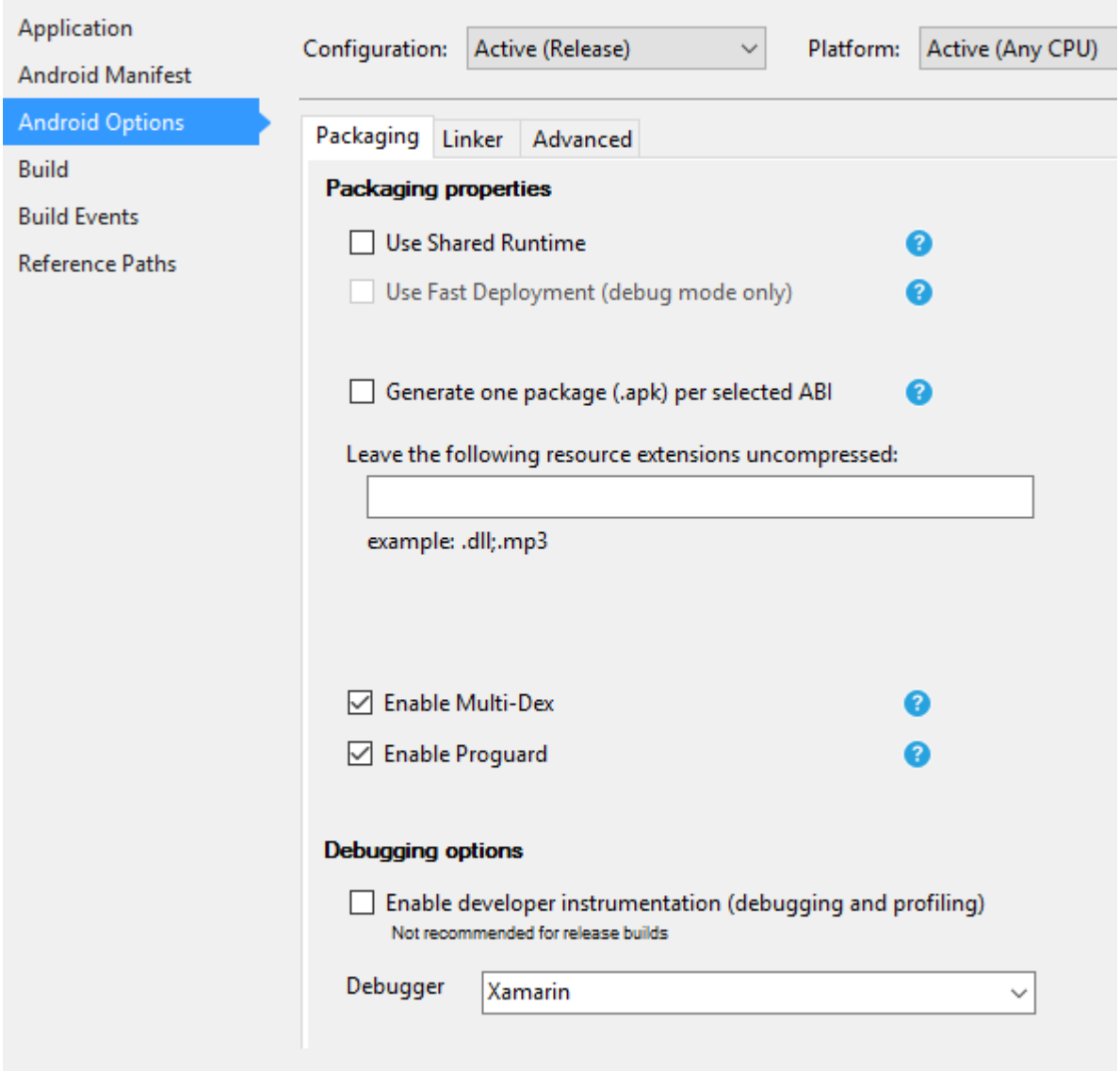
Le versioni del sistema operativo Android precedenti a Android 5.0 Lollipop (API 21) utilizzano il runtime Dalvik, che supporta solo un file .dex per APK, limitando a 65.536 metodi per APK. A partire da Android 5.0, il sistema operativo Android utilizza ART runtime, che può supportare più di un file .dex per APK, evitando il limite.

Per superare il limite dei 65k metodi nelle versioni Android sotto l'API 21, gli sviluppatori devono utilizzare la libreria di supporto MultiDex. Il MultiDex crea file classes.dex aggiuntivi (classes2.dex, classes3.dex, ...) facendoli riferimento nel file classes.dex. Quando l'app inizia il caricamento, utilizza una classe MultiDexApplication per caricare i file .dex aggiuntivi.

Se la tua app Android punta a una versione minima dell'SDK uguale o superiore all'API 21 (Android 5.0 Lollipop), non è necessario utilizzare la libreria MultiDex, poiché il sistema operativo gestisce in modo nativo i file .dex aggiuntivi. Tuttavia, se per ragioni di compatibilità lo sviluppatore vuole supportare il precedente sistema operativo Android, allora lui / lei dovrebbe usare la libreria MultiDex.

Come usare MultiDex nella tua app Xamarin.Android

Innanzitutto, per attivare MultiDex nella tua app Xamarin.Android, vai al tuo progetto Proprietà -> Opzioni Android -> Packaging -> Attiva Multi-Dex, come nella schermata di stampa qui sotto:



Quindi, devi creare una classe MultiDexApplication nella tua app. Nella root del progetto, creare una nuova classe (in Esplora soluzioni, fare clic con il tasto destro del mouse nel progetto, Aggiungi .. -> Classe o Maiusc + Alt + C). Nel nuovo file di classe, copia il seguente codice, sostituendo lo spazio dei nomi Sample con il nome del tuo spazio dei nomi del progetto Xamarin.Android.

```
using System;
using Android.App;
using Android.Runtime;
using Java.Interop;

namespace Sample
{
    [Register("android/support/multidex/MultiDexApplication", DoNotGenerateAcw = true)]
    public class MultiDexApplication : Application
    {
        internal static readonly JniPeerMembers _members =
        new XAPeerMembers("android/support/multidex/MultiDexApplication", typeof
(MultiDexApplication));

        internal static IntPtr java_class_handle;

        private static IntPtr id_ctor;
```

```
        [Register(".ctor", "()V", "", DoNotGenerateAcw = true)]
        public MultiDexApplication()
        : base(IntPtr.Zero, JniHandleOwnership.DoNotTransfer)
        {
            if (Handle != IntPtr.Zero)
                return;

            try
            {
                if (GetType() != typeof (MultiDexApplication))
                {
                    SetHandle(
                        JNIEnv.StartCreateInstance(GetType(), "()V"),
                        JniHandleOwnership.TransferLocalRef);
                        JNIEnv.FinishCreateInstance(Handle, "()V");
                    return;
                }

                if (id_ctor == IntPtr.Zero)
                    id_ctor = JNIEnv.GetMethodID(class_ref, "<init>", "()V");
                SetHandle(
                    JNIEnv.StartCreateInstance(class_ref, id_ctor),
                    JniHandleOwnership.TransferLocalRef);
                JNIEnv.FinishCreateInstance(Handle, class_ref, id_ctor);
            }
            finally
            {
            }
        }

        protected MultiDexApplication(IntPtr javaReference, JniHandleOwnership transfer)
            : base(javaReference, transfer)
        {
        }

        internal static IntPtr class_ref
        {
            get { return JNIEnv.FindClass("android/support/multidex/MultiDexApplication", ref
java_class_handle); }
        }

        protected override IntPtr ThresholdClass
        {
            get { return class_ref; }
        }

        protected override Type ThresholdType
        {
            get { return typeof (MultiDexApplication); }
        }
    }
}
```

Codice sorgente qui.

Se stai sviluppando in Visual Studio per Windows, c'è anche un bug negli strumenti di build dell'SDK di Android che devi correggere per creare correttamente i file classes.dex quando costruisci il tuo progetto.

Vai alla cartella Android SDK, apri la cartella build-tools e ci saranno cartelle con i numeri dei compilatori di Android SDK, come ad esempio:

C: \ android-sdk \ build-tools \ 23.0.3 \

C: \ android-sdk \ build-tools \ 24.0.1 \

C: \ android-sdk \ build-tools \ 25.0.2 \

All'interno di ciascuna di queste cartelle, c'è un file chiamato **mainClassesDex.bat** , uno script batch utilizzato per creare i file classes.dex. Apri ogni file mainClassesDex.bat con un editor di testo (Notepad o Notepad ++) e, nel suo script, trova e sostituisce il blocco:

```
if DEFINED output goto redirect
call "%java_exe%" -Djava.ext.dirs="%frameworkdir%" com.android.multidex.MainDexListBuilder
"%disableKeepAnnotated%" "%tmpJar%" "%params%"
goto afterClassReferenceListBuilder
:redirect
call "%java_exe%" -Djava.ext.dirs="%frameworkdir%" com.android.multidex.MainDexListBuilder
"%disableKeepAnnotated%" "%tmpJar%" "%params%" 1>"%output%"
:afterClassReferenceListBuilder
```

Con il blocco:

```
SET params=%params:'=%
if DEFINED output goto redirect
call "%java_exe%" -Djava.ext.dirs="%frameworkdir%" com.android.multidex.MainDexListBuilder
%disableKeepAnnotated% "%tmpJar%" %params%
goto afterClassReferenceListBuilder
:redirect
call "%java_exe%" -Djava.ext.dirs="%frameworkdir%" com.android.multidex.MainDexListBuilder
%disableKeepAnnotated% "%tmpJar%" %params% 1>"%output%"
:afterClassReferenceListBuilder
```

Fonte qui.

Salva ogni mainClassesDex.bat nell'editor di testo dopo le modifiche.

Dopo i passaggi precedenti, dovresti essere in grado di creare con successo la tua app Xamarin.Android con MultiDex.

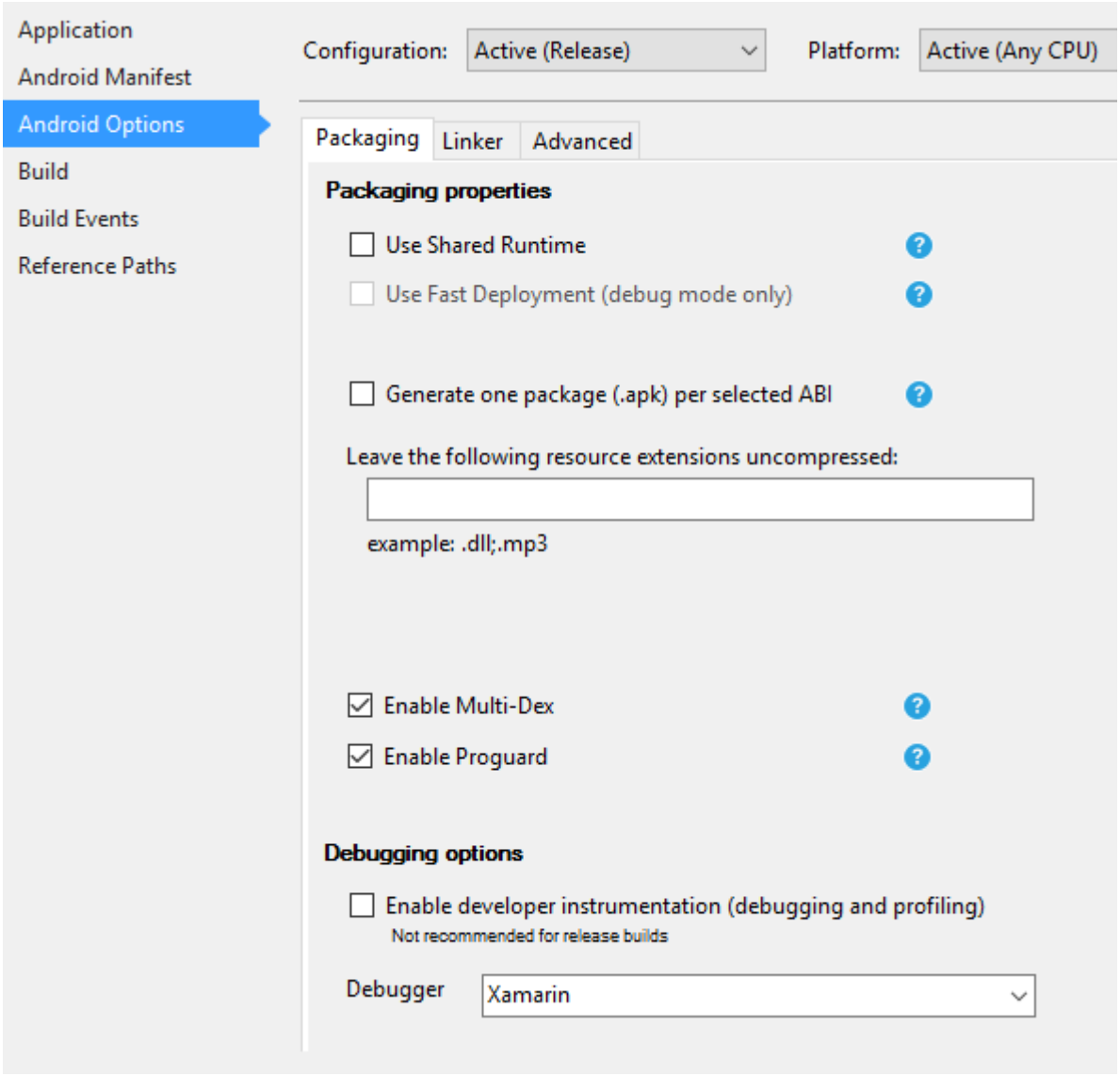
Abilitazione di ProGuard nel tuo APK Xamarin.Android

ProGuard è uno strumento utilizzato nel processo di costruzione per ottimizzare e offuscare il codice Java del tuo APK e rimuovere anche le classi non utilizzate. L'APK risultante quando si utilizza ProGuard avrà una dimensione inferiore e sarà più difficile da decodificare (decompilazione).

ProGuard può essere utilizzato anche in Xamarin.App Android e ridurrà anche la dimensione del file APK e offuscerà il codice Java. Essere consapevoli, tuttavia, che l'offuscamento di ProGuard si applica solo al codice Java. Per offuscare il codice .NET, lo sviluppatore dovrebbe usare Dotfuscator o strumenti simili.

Come usare ProGuard nella tua app Xamarin.Android

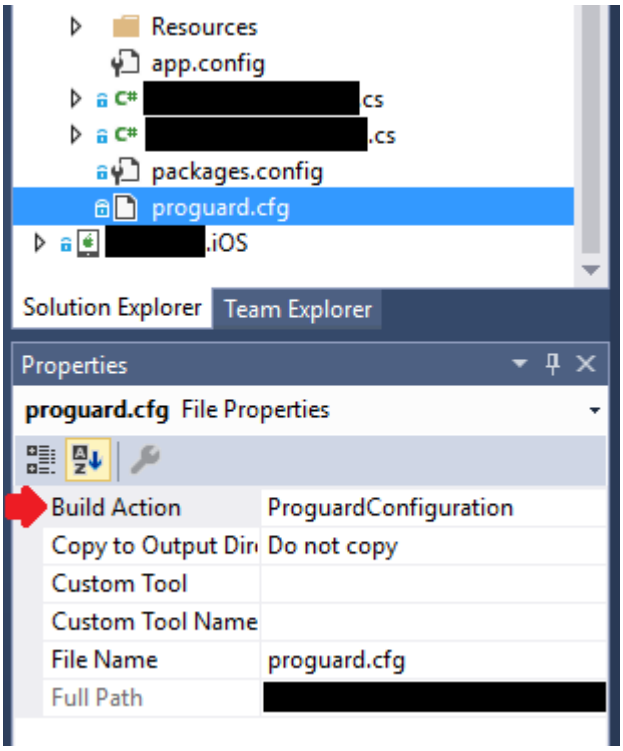
Innanzitutto, per abilitare ProGuard nella tua app Xamarin.Android, vai al tuo progetto Proprietà -> Opzioni Android -> Packaging -> Abilita ProGuard, come nella seguente schermata di stampa:



Ciò consente a ProGuard di creare la tua app.

Xamarin.Android, per impostazione predefinita, imposta le proprie configurazioni per ProGuard, che possono essere trovate all'interno delle cartelle `obj/Debug/proguard` o `obj/Release/proguard` , nei file `proguard_project_primary.cfg` , `proguard_project_references.cfg` e `proguard_xamarin.cfg` . I tre file sono combinati come configurazioni per ProGuard e vengono creati automaticamente da Xamarin durante la creazione.

Se lo sviluppatore desidera personalizzare ulteriormente le opzioni di ProGuard, può creare un file nella radice del progetto denominato `proguard.cfg` (anche gli altri nomi sono validi, purché l'estensione sia .cfg) e impostando la relativa azione di compilazione su ProguardConfiguration, come nella foto qui sotto:



Nel file, è possibile inserire opzioni personalizzate di ProGuard, come `-dontwarn` , `-dontwarn -keep class` e altri .

Importante

A partire da ora (aprile / 2017), l'SDK di Android che viene solitamente scaricato ha una vecchia versione di ProGuard, che può causare errori durante la creazione dell'app utilizzando Java 1.8. Durante la creazione, l'Elenco errori mostra il seguente messaggio:

```
Error
Can't read [C:\Program Files (x86)\Reference
Assemblies\Microsoft\Framework\MonoAndroid\v7.0\mono.android.jar]
(Can't process class [android/app/ActivityTracker.class]  (Unsupported class version number
[52.0] (maximum 51.0, Java 1.7))) [CREATEMULTIDEXMAINDEXCLASSLIST]
```

Fonte qui.

Per risolvere questo problema, è necessario scaricare la versione più recente di ProGuard (qui) e copiare il contenuto del file .zip in `android-sdk\tools\proguard\` . Questo aggiornerà ProGuard e il processo di costruzione dovrebbe funzionare senza problemi.

Dopodiché, dovresti essere in grado di creare con successo la tua app Xamarin.Android con ProGuard.

Bug "Misteriosi" relativi a ProGuard e Linker

Hai realizzato un'ottima app e l'hai testata in Debug, con buoni risultati. Tutto stava funzionando bene!

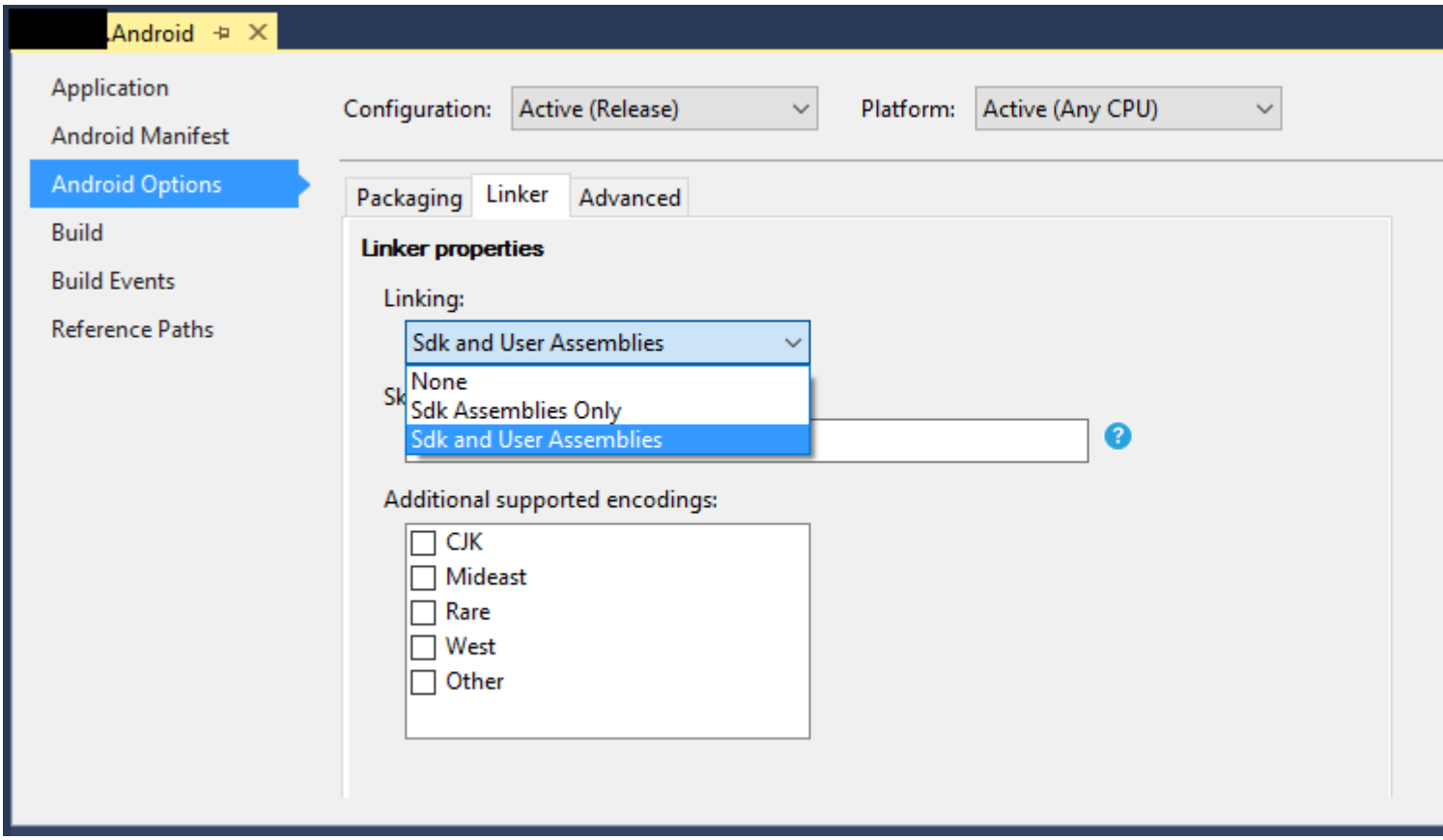
Ma poi, hai deciso di preparare la tua app per il rilascio. Hai configurato MultiDex, ProGuard e

Linker, e poi ha smesso di funzionare.

Questo tutorial ha lo scopo di aiutarti a scoprire problemi comuni relativi a ProGuard e Linker che possono causare bug misteriosi.

Capire Xamarin.Linker

Xamarin.Linker è uno strumento nel processo di costruzione che rimuove codice e classi non utilizzati **dal codice .NET (non dal codice Java)** . Nelle Proprietà del tuo progetto -> Opzioni Android -> Linker, ci sarà una casella di selezione Collegamento con le opzioni:



Nessuno : nessun codice rimosso.

Solo assembly Sdk : questa opzione consente a Xamarin.Linker di verificare la presenza di codice non utilizzato solo nelle librerie Xamarin. **Questa opzione è sicura.**

Sdk e gruppi utente : questa opzione consente a Xamarin.Linker di verificare la presenza di codice inutilizzato nelle librerie Xamarin e nel codice del progetto (inclusi PCL, componenti Xamarin e pacchetti NuGet). **Questa opzione non è sempre sicura!**

Quando si utilizza l'opzione Sdk e User Assembly, Xamarin.Linker potrebbe pensare che parti del codice non siano utilizzate quando effettivamente sono molto utilizzate! Ciò potrebbe causare il blocco corretto di alcune librerie e causare errori nella tua app.

Per fare in modo che Xamarin.Linker non rimuova il codice, ci sono 3 opzioni:

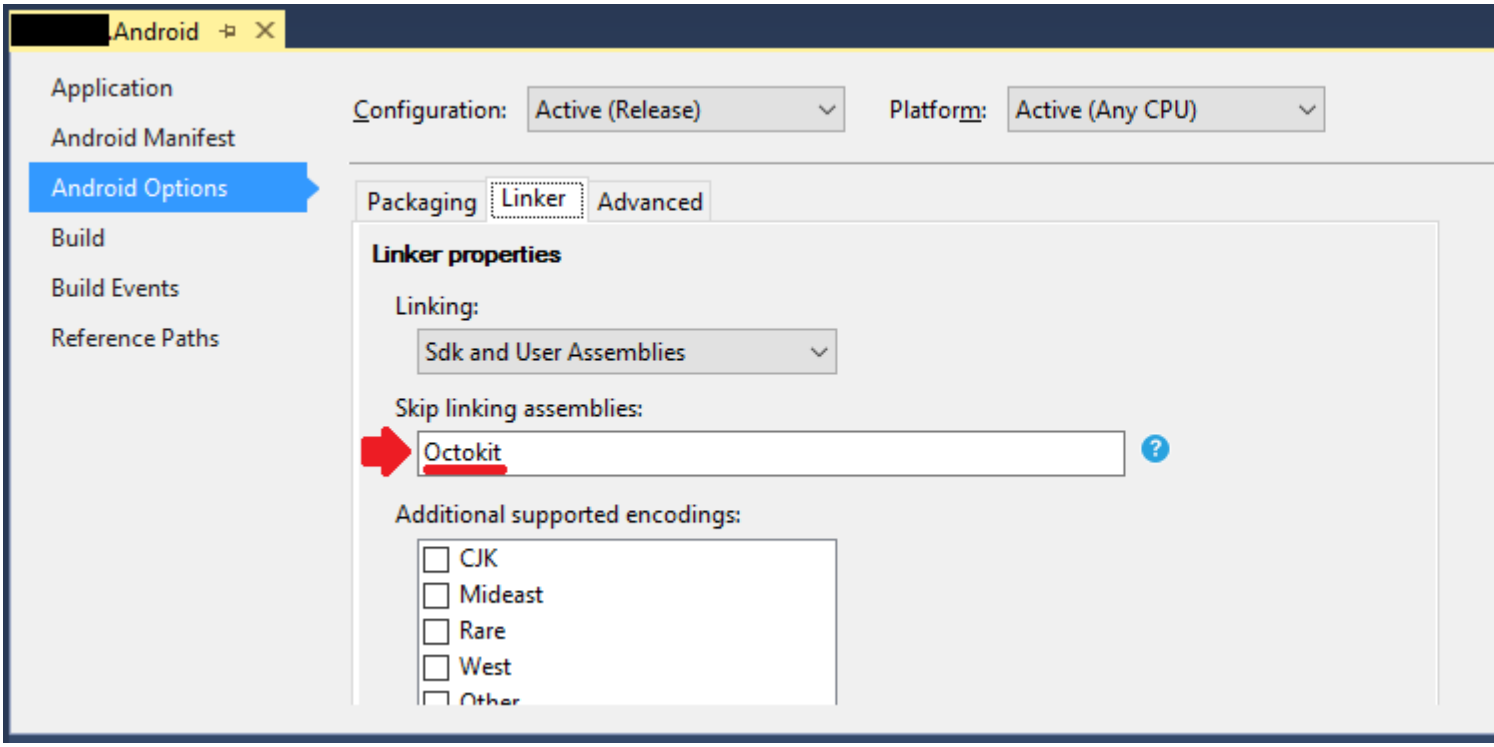
1. Impostazione dell'opzione di collegamento su Nessuno o Solo gruppi Sdk;
2. Salta il collegamento di assiemi;
3. Utilizzo dell'attributo Preserve.

Esempio per 2. Salta il collegamento degli assiemi:

Nell'esempio seguente, l'uso di Xamarin.Linker ha causato un pacchetto NuGet (Octokit) che funziona correttamente per non funzionare più, perché non potrebbe più connettersi a Internet:

```
[0:] ERROR
[0:] SOURCE: mscorlib
[0:] MESSAGE: Object reference not set to an instance of an object.
[0:] STACK TRACE:   at Octokit.PocoJsonSerializerStrategy.DeserializeObject (System.Object
value, System.Type type) [0x003d8] in D:\repos\octokit.net\Octokit\SimpleJson.cs:1472
  at Octokit.Internal.SimpleJsonSerializer+GitHubSerializerStrategy.DeserializeObject
(System.Object value, System.Type type) [0x001c3] in
D:\repos\octokit.net\Octokit\Http\SimpleJsonSerializer.cs:165
  at Octokit.SimpleJson.DeserializeObject (System.String json, System.Type type,
Octokit.IJsonSerializerStrategy jsonSerializerStrategy) [0x00007] in
D:\repos\octokit.net\Octokit\SimpleJson.cs:583
  at Octokit.SimpleJson.DeserializeObject[T] (System.String json,
Octokit.IJsonSerializerStrategy jsonSerializerStrategy) [0x00000] in
D:\repos\octokit.net\Octokit\SimpleJson.cs:595
  at Octokit.Internal.SimpleJsonSerializer.Deserialize[T] (System.String json) [0x00000] in
D:\repos\octokit.net\Octokit\Http\SimpleJsonSerializer.cs:21
  at Octokit.Internal.JsonHttpPipeline.DeserializeResponse[T] (Octokit.IResponse response)
[0x000a7] in D:\repos\octokit.net\Octokit\Http\JsonHttpPipeline.cs:62
  at Octokit.Connection+<Run>d__54`1[T].MoveNext () [0x0009c] in
D:\repos\octokit.net\Octokit\Http\Connection.cs:574
--- End of stack trace from previous location where exception was thrown ---
```

Per far funzionare nuovamente la libreria, è stato necessario aggiungere il nome di riferimento del pacchetto nel campo Ignora collegamenti assiemi, situato in progetto -> Proprietà -> Opzioni Android -> Linker, come nella seguente immagine:



Successivamente, la libreria ha iniziato a funzionare senza problemi.

Esempio per 3. Uso dell'attributo Preserve:

Xamarin.Linker percepisce come codice inutilizzato principalmente codice dalle classi di modelli nel nucleo del progetto.

Per mantenere la classe preservata durante il processo di collegamento, è possibile utilizzare l'attributo Preserve.

Innanzitutto, crea nella classe del tuo progetto una classe denominata **PreserveAttribute.cs** , inserisci il seguente codice e sostituisci lo spazio dei nomi con lo spazio dei nomi del tuo progetto:

PreserveAttribute.cs:

```
namespace My_App_Core.Models
{
    public sealed class PreserveAttribute : System.Attribute
    {
        public bool AllMembers;
        public bool Conditional;
    }
}
```

In ogni classe di modello del core del progetto, inserisci l'attributo Preserve come nell'esempio seguente:

Country.cs:

```
using System;
using System.Collections.Generic;

namespace My_App_Core.Models
{
    [Preserve(AllMembers = true)]
    public class Country
    {
        public String name { get; set; }
        public String ISOcode { get; set; }

        [Preserve(AllMembers = true)]
        public Country(String name, String ISOCode)
        {
            this.name = name;
            this.ISOCode = ISOCode;
        }
    }
}
```

Successivamente, il processo di collegamento non rimuoverà più il codice conservato.

Capire ProGuard

ProGuard è uno strumento nel processo di costruzione che rimuove codice inutilizzato e classi **dal tuo codice Java** . Inoltre offusca e ottimizza il codice.

Tuttavia, a volte ProGuard può rimuovere il codice che percepisce come non utilizzato, quando non lo è. Per evitare ciò, lo sviluppatore deve eseguire il debug dell'applicazione (in Monitoraggio dispositivi Android e nel Debug di Visual Studio) e rilevare quale classe è stata rimossa, per poi configurare il file di configurazione di ProGuard per mantenere la classe.

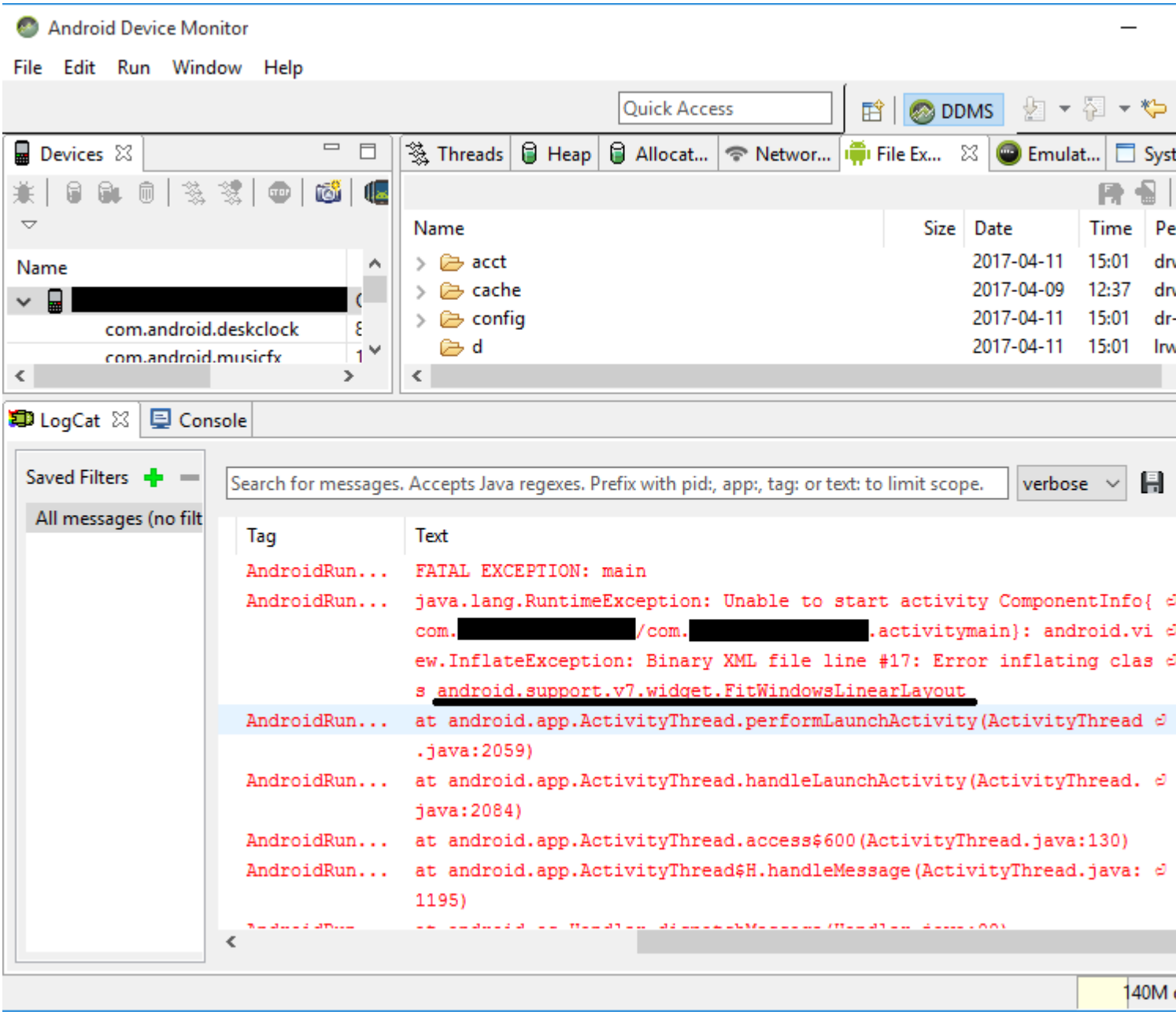
Esempio

Nell'esempio seguente, ProGuard ha rimosso due classi (Android.Support.V7.Widget.FitWindowsLinearLayout e Android.Support.Design.Widget.AppBarLayout) utilizzate nei file di layout AXML, ma che erano state percepite come non utilizzate nel codice. La rimozione ha causato ClassNotFoundException nel codice Java durante il rendering del layout dell'attività:

layout_activitymain.axml:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activitymain_drawerlayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true" <!-- ### HERE ### -->
    tools:openDrawer="start">
    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:fitsSystemWindows="true">
        <!-- ### HERE ## -->
        <android.support.design.widget.AppBarLayout
            android:id="@+id/activitymain_appbarlayout"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:theme="@style/AppTheme.AppBarOverlay">
...
```

LogCat che mostra errore durante la creazione del layout in SetContentView:



Per correggere questo errore, è stato necessario aggiungere le seguenti righe al file di configurazione di ProGuard del progetto:

```
-keep public class android.support.v7.widget.FitWindowsLinearLayout
-keep public class android.support.design.widget.AppBarLayout
```

Successivamente, non sono stati più visualizzati errori durante la creazione del layout.

Avvertenze ProGuard

A volte ProGuard mostra avvisi nella Lista degli errori dopo aver creato il tuo progetto. Anche se sollevano una domanda se la tua app è OK o no, non tutti i loro avvertimenti indicano problemi, specialmente se la tua app ha successo.

Un esempio è quando si utilizza la libreria Picasso : quando si utilizza ProGuard, è possibile che `okio.Okio: can't find referenced class (...)` visualizzati avvisi come `okio.Okio: can't find`

`referenced class (...)` o `can't write resource [META-INF/MANIFEST.MF] (Duplicate zip entry [okhttp.jar:META-INF/MANIFEST.MF]) (...)` , ma l'app si costruisce e la libreria funziona senza problemi.

Leggi Pubblicare il tuo APK Xamarin.Android online: https://riptutorial.com/it/xamarin-android/topic/9601/pubblicare-il-tuo-apk-xamarin-android

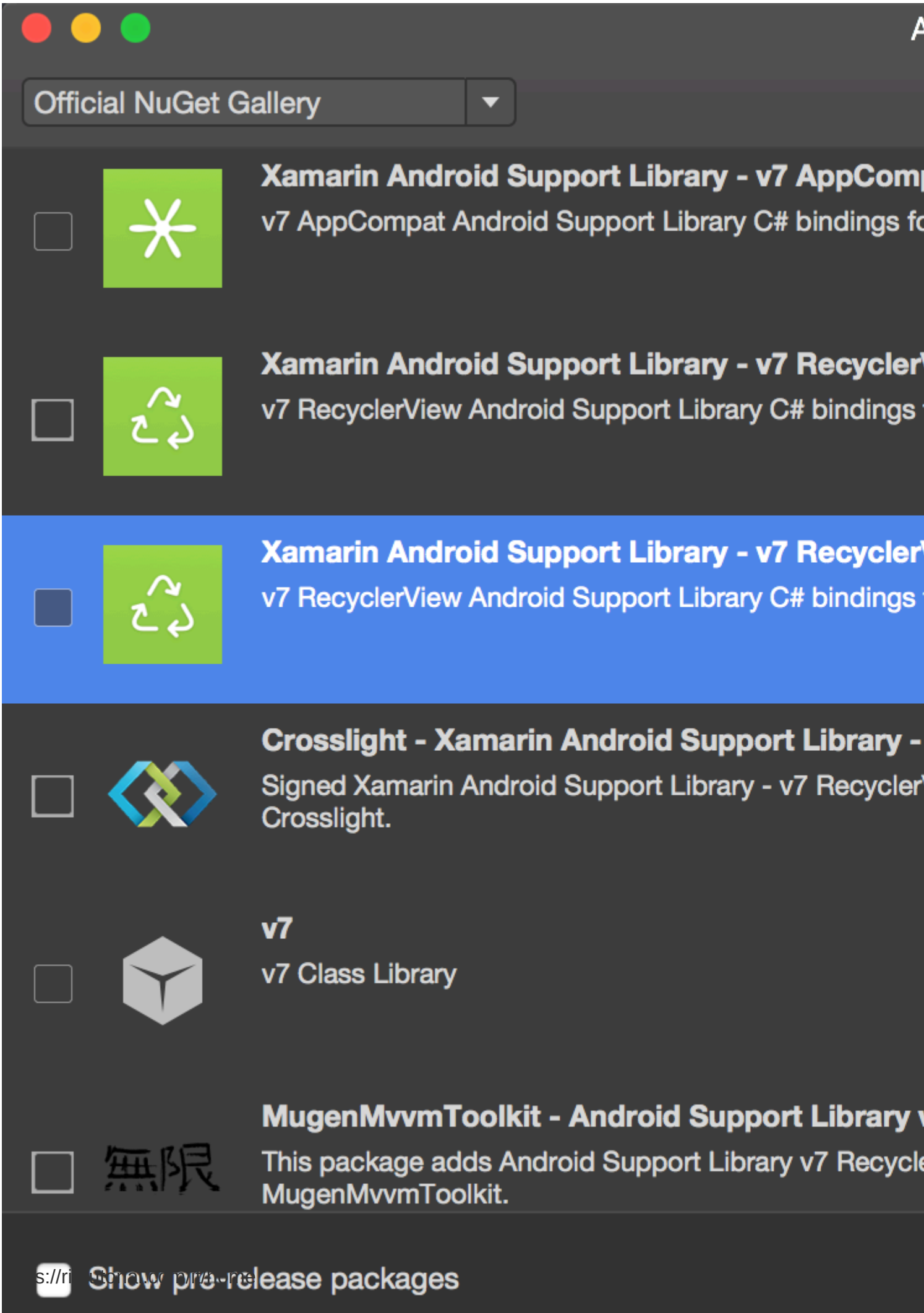
Capitolo 10: RecyclerView

Examples

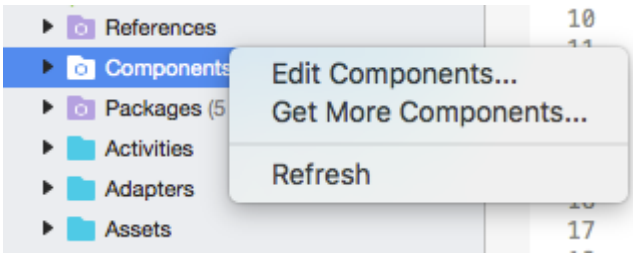
Informazioni di base su RecyclerView

Questo è un esempio di utilizzo di `Android Support Library V7 RecyclerView` . Le librerie di supporto sono generalmente raccomandate perché forniscono versioni compatibili con le versioni precedenti di nuove funzionalità, forniscono utili elementi dell'interfaccia utente che non sono inclusi nel framework e forniscono una gamma di utilità che le app possono utilizzare.

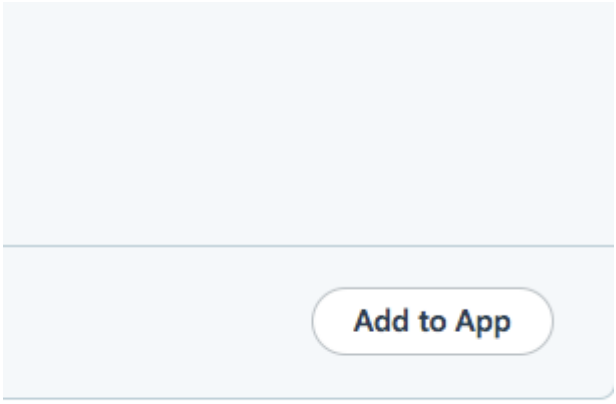
Per ottenere il `RecyclerView` , installeremo i pacchetti necessari di Nuget. In primo luogo, cercheremo la `v7 recyclerview` . Scorri verso il basso fino a visualizzare la `Xamarin Android Support Library - v7 RecyclerView` . Selezionalo e fai clic su **Aggiungi pacchetto** .


Official NuGet Gallery
Xamarin Android Support Library - v7 AppCom
v7 AppCompat Android Support Library C# bindings fo
Xamarin Android Support Library - v7 Recycler
v7 RecyclerView Android Support Library C# bindings
Xamarin Android Support Library - v7 Recycler
v7 RecyclerView Android Support Library C# bindings
Crosslight - Xamarin Android Support Library -
Signed Xamarin Android Support Library - v7 Recycler
Crosslight.
v7
v7 Class Library
MugenMvvmToolkit - Android Support Library
This package adds Android Support Library v7 Recycle
MugenMvvmToolkit.
Show pre-release packages

è disponibile come componente Xamarin. Per aggiungere il componente, fare clic con il tasto destro del mouse su `Components` all'interno del progetto Android in Esplora soluzioni e fare clic su `Get More Components` .



All'interno della finestra Component Store visualizzata, cercare RecyclerView. Nell'elenco di ricerca, selezionare `Android Support Library V7 RecyclerView` . Quindi fare clic su `Add to App` . Il componente viene aggiunto al progetto.



Il prossimo passo è aggiungere il RecyclerView a una pagina. All'interno del file `axml` (layout), possiamo aggiungere `RecyclerView` come di seguito.

```
<android.support.v7.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:scrollbars="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
```

RecyclerView richiede l'impostazione di almeno due classi helper per l'implementazione dello standard di base: `Adapter` e `ViewHolder` . `Adapter` gonfia i layout degli articoli e associa i dati alle viste visualizzate in RecyclerView. ViewHolder cerca e memorizza i riferimenti delle viste. Il titolare della vista aiuta anche a rilevare i clic di visualizzazione delle voci.

Ecco un esempio di base di Adapter Class

```
public class MyAdapter : RecyclerView.Adapter
{
    string [] items;

    public MyAdapter (string [] data)
    {
        items = data;
```

```
    }

    // Create new views (invoked by the layout manager)
    public override RecyclerView.ViewHolder OnCreateViewHolder (ViewGroup parent, int
viewType)
    {
        // set the view's size, margins, paddings and layout parameters
        var tv = new TextView (parent.Context);
        tv.SetWidth (200);
        tv.Text = "";

        var vh = new MyViewHolder (tv);
        return vh;
    }

    // Replace the contents of a view (invoked by the layout manager)
    public override void OnBindViewHolder (RecyclerView.ViewHolder viewHolder, int position)
    {
        var item = items [position];

        // Replace the contents of the view with that element
        var holder = viewHolder as MyViewHolder;
        holder.TextView.Text = items[position];
    }

    public override int ItemCount {
        get {
            return items.Length;
        }
    }
}
```

Nel metodo `OnCreateViewHolder` , prima gonfiamo una vista e creiamo un'istanza della classe ViewHolder. Questa istanza deve essere restituita. Questo metodo viene richiamato dall'adapter quando richiede una nuova istanza di ViewHolder. Questo metodo non verrà invocato per ogni singola cella. Una volta che RecyclerView ha abbastanza celle per riempire la vista, riutilizzerà le vecchie celle che vengono scorse dalla vista per ulteriori celle.

Il callback `OnBindViewHolder` viene richiamato da Adapter per visualizzare i dati nella posizione specificata. Questo metodo dovrebbe aggiornare il contenuto dell'elementoView per riflettere l'elemento nella posizione specificata.

Poiché la cella contiene solo un singolo `TextView` , possiamo avere un semplice ViewHolder come sotto.

```
public class MyViewHolder : RecyclerView.ViewHolder
{
    public TextView TextView { get; set; }


    public MyViewHolder (TextView v) : base (v)
    {
        TextView = v;
    }
}
```

Il prossimo passo è collegare le cose in `Activity` .

```
RecyclerView mRecyclerView;
MyAdapter mAdapter;
protected override void OnCreate (Bundle bundle)
{
    base.OnCreate (bundle);
    SetContentView (Resource.Layout.Main);
    mRecyclerView = FindViewById<RecyclerView> (Resource.Id.recyclerView);

    // Plug in the linear layout manager:
    var layoutManager = new LinearLayoutManager (this) { Orientation =
LinearLayoutManager.Vertical };
    mRecyclerView.SetLayoutManager (layoutManager);
    mRecyclerView.HasFixedSize = true;

    var recyclerViewData = GetData();
    // Plug in my adapter:
    mAdapter = new MyAdapter (recyclerViewData);
    mRecyclerView.SetAdapter (mAdapter);
}

string[] GetData()
{
     string[] data;
     .
     .
     .
     return data;
}
```

La classe LayoutManager è responsabile della misurazione e del posizionamento delle viste degli oggetti all'interno di un RecyclerView e della determinazione del criterio di riciclo delle viste degli articoli che non sono più visibili all'utente. Prima di `RecyclerView` , dovevamo usare `ListView` per disporre le celle come in un elenco a scorrimento verticale e `GridView` per visualizzare gli elementi in una griglia scorrevole bidimensionale. Ma ora possiamo realizzare entrambi con RecyclerView impostando un diverso LayoutManger. `LinearLayoutManager` organizza le celle come in un ListView e `GridLayoutManager` dispone le celle in modalità Grid.

RecyclerView con eventi Click

Questo esempio mostra come impostare Click EventHandlers in Xamarin.Android RecyclerView.

In Android Java , il modo per impostare un listener per un clic è utilizzare un onClickListener per la vista su cui verrà fatto clic, in questo modo:

```
ImageView picture = findViewById(R.id.item_picture);
picture.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // do stuff
    }
});
```

Tuttavia, in Xamarin.Android, il modo per impostare un listener per un evento Click è **aggiungere** un EventHandler, nei seguenti modi:

1.

```
ImageView picture = FindViewById<ImageView>(Resource.Id.item_picture);
picture.Click += delegate {
    // do stuff
};
```

2.

```
ImageView picture = FindViewById<ImageView>(Resource.Id.item_picture);
picture.Click += async delegate {
    // await DoAsyncMethod();
    // do async stuff
};
```

3.

```
ImageView picture = FindViewById<ImageView>(Resource.Id.item_picture);
picture.Click += Picture_Click;
... // rest of your method

private void Picture_Click(object sender, EventArgs e)
{
    // do stuff
}
```

Si noti che **EventHandler viene aggiunto, non impostato.** Se ClickHandler viene aggiunto all'interno di un metodo GetView da un adattatore GridView / ListView o da un metodo OnBindViewHolder da RecyclerView.Adapter, ogni volta che viene creata la vista dell'elemento verrà aggiunto un nuovo EventHandler. Dopo averlo fatto scorrere più volte, verranno aggiunti più EventHandler e, quando la vista viene cliccata, tutti verranno attivati.

Per evitare questo problema, EventHandlers deve essere annullato e sottoscritto successivamente nei metodi GetView o OnBindViewHolder. Inoltre, devono utilizzare il modo numero **3.** per impostare EventHandler, altrimenti non sarà possibile annullare l'iscrizione a EventHandlers.

Di seguito è riportato un esempio di un RecyclerView.Adapter con eventi Click:

```
public class ViewHolderPerson : Android.Support.V7.Widget.RecyclerView.ViewHolder
{
    public View Item { get; private set; }
    public ImageView Picture { get; private set; }
    public TextView Name { get; private set; }

    public ViewHolderPerson(View itemView) : base(itemView)
    {
        this.Item = itemView;
        this.Picture = itemView.FindViewById<ImageView>(Resource.Id.Item_Person_Picture);
        this.Name = itemView.FindViewById<TextView>(Resource.Id.Item_Person_Name);
    }
}

public class AdapterPersons : Android.Support.V7.Widget.RecyclerView.Adapter
```

```
{
    private Context context;
    private Android.Support.V7.Widget.RecyclerView recyclerView;
    private List<Person> persons;

    public AdapterPersons(Context context, Android.Support.V7.Widget.RecyclerView
recyclerView, List<Person> persons)
    {
        this.context = context;
        this.recyclerView = recyclerView;
        this.persons = persons;
    }

    public override int ItemCount => persons.Count;

    public override void OnBindViewHolder(RecyclerView.ViewHolder holder, int position)
    {
        Person person = this.persons[position];
        ((ViewHolderPerson)holder).Name.Text = person.Name;
        ((ViewHolderPerson)holder).Picture.SetImageBitmap(person.Picture);

        // Unsubscribe and subscribe the method, to avoid setting multiple times.
        ((ViewHolderPerson)holder).Item.Click -= Person_Click;
        ((ViewHolderPerson)holder).Item.Click += Person_Click;
    }

    private void Person_Click(object sender, EventArgs e)
    {
        int position = this.recyclerView.GetChildAdapterPosition((View)sender);
        Person personClicked = this.persons[position];
        if(personClicked.Gender == Gender.Female)
        {
            Toast.MakeText(this.context, "The person clicked is a female!",
ToastLength.Long).Show();
        }
        else if(personClicked.Gender == Gender.Male)
        {
            Toast.MakeText(this.context, "The person clicked is a male!",
ToastLength.Long).Show();
        }
    }

    public override RecyclerView.ViewHolder OnCreateViewHolder(ViewGroup parent, int viewType)
    {
        View itemView =
LayoutInflater.From(parent.Context).Inflate(Resource.Layout.item_person, parent, false);
        return new ViewHolderPerson(itemView);
    }

}
```

Leggi RecyclerView online: https://riptutorial.com/it/xamarin-android/topic/3452/recyclerview

Capitolo 11: Scansione di codici a barre tramite la libreria ZXing in Xamarin Applications

introduzione

La libreria Zxing è ben nota per l'elaborazione delle immagini. Zxing era basato su java e il modulo .Net è anche disponibile e può essere utilizzato nelle applicazioni xamarin. clicca qui per consultare la documentazione ufficiale. http://zxingnet.codeplex.com/

Recentemente ho usato questo libarry.

Passaggio 1: aggiungere il componente ZXing.Net.Mobile alla soluzione.

step2: In qualsiasi attività dobbiamo mostrare lo scanner di codici a barre, in tale attività inizializzare MobileBarcodeScanner.

Passaggio 3: scrivere sotto il codice quando si tocca su qualsiasi vista per avviare la scansione.

Examples

Codice di esempio

```
button.Click +=async delegate
{
var MScanner = new MobileBarcodeScanner();
var Result = await MScanner.Scan();
if(Result == null)
{
    return;
}
//get the bar code text here
string BarcodeText = Result.text;
}
```

Leggi Scansione di codici a barre tramite la libreria ZXing in Xamarin Applications online: https://riptutorial.com/it/xamarin-android/topic/9526/scansione-di-codici-a-barre-tramite-la-libreria-zxing-in-xamarin-applications

Capitolo 12: Xamarin.Android - Come creare una barra degli strumenti

Osservazioni

Cara squadra,

Penso che sia utile menzionare la documentazione ufficiale di Android in cui il controllo della barra degli strumenti è spiegato nei dettagli:

https://developer.android.com/reference/android/support/v7/widget/Toolbar.html

Sono inoltre presenti contenuti interessati sulla libreria Android.Support.v7 utilizzata nell'esempio:

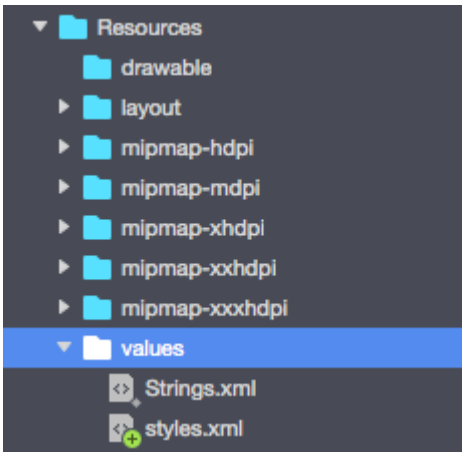
https://developer.android.com/training/appbar/index.html

Examples

Aggiungi la barra degli strumenti all'applicazione Xamarin.Android

Per prima cosa devi aggiungere la libreria Xamarin.Android.Support.V7.AppCompat per NuGet: https://www.nuget.org/packages/Xamarin.Android.Support.v7.AppCompat/

Nella cartella "values" in "Risorse" aggiungi un nuovo file xml chiamato "styles.xml":

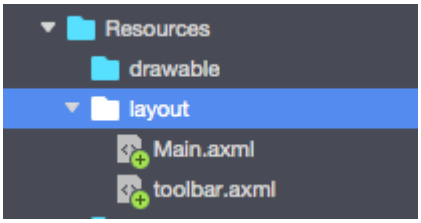


Il file "styles.xml" dovrebbe contenere sotto il codice:

```
<?xml version="1.0" encoding="utf-8" ?>
<resources>
<style name="MyTheme" parent="MyTheme.Base">
</style>

<!-- Base theme applied no matter what API -->
<style name="MyTheme.Base" parent="Theme.AppCompat.Light.DarkActionBar">
<item name="windowNoTitle">true</item>
<!--We will be using the toolbar so no need to show ActionBar-->
```

```
<item name="windowActionBar">false</item>
<!-- Set theme colors from http://www.google.com/design/spec/style/color.html#color-color-
palette-->
<!-- colorPrimary is used for the default action bar background -->
<item name="colorPrimary">#2196F3</item>
<!-- colorPrimaryDark is used for the status bar -->
<item name="colorPrimaryDark">#1976D2</item>
<!-- colorAccent is used as the default value for colorControlActivated
     which is used to tint widgets -->
<item name="colorAccent">#FF4081</item>

<item name="colorControlHighlight">#FF4081</item>
<!-- You can also set colorControlNormal, colorControlActivated
     colorControlHighlight and colorSwitchThumbNormal. -->
```

Il passaggio successivo consiste nell'aggiungere il file "toolbar.axml" che contiene la definizione del controllo della barra degli strumenti nella cartella "layout":



Aggiungi sotto il codice per definire la barra degli strumenti:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.Toolbar xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
android:id="@+id/toolbar"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:minHeight="?attr/actionBarSize"
android:background="?attr/colorPrimary"
android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />
```

Ora apri il file "Main.axml" e aggiungi sotto il codice appena sotto il tag di chiusura per il primo layout. Il tuo codice dovrebbe apparire come di seguito:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">

    <include android:id="@+id/toolbar" layout="@layout/toolbar" />

</LinearLayout>
```

Ora devi aggiungere informazioni sul tema che la tua app utilizza. Apri il file "AndroidManifest" e aggiungi le informazioni del tema al tag "application":

```
<application android:theme="@style/MyTheme" android:allowBackup="true"
```

```
android:icon="@mipmap/icon" android:label="@string/app_name">
```

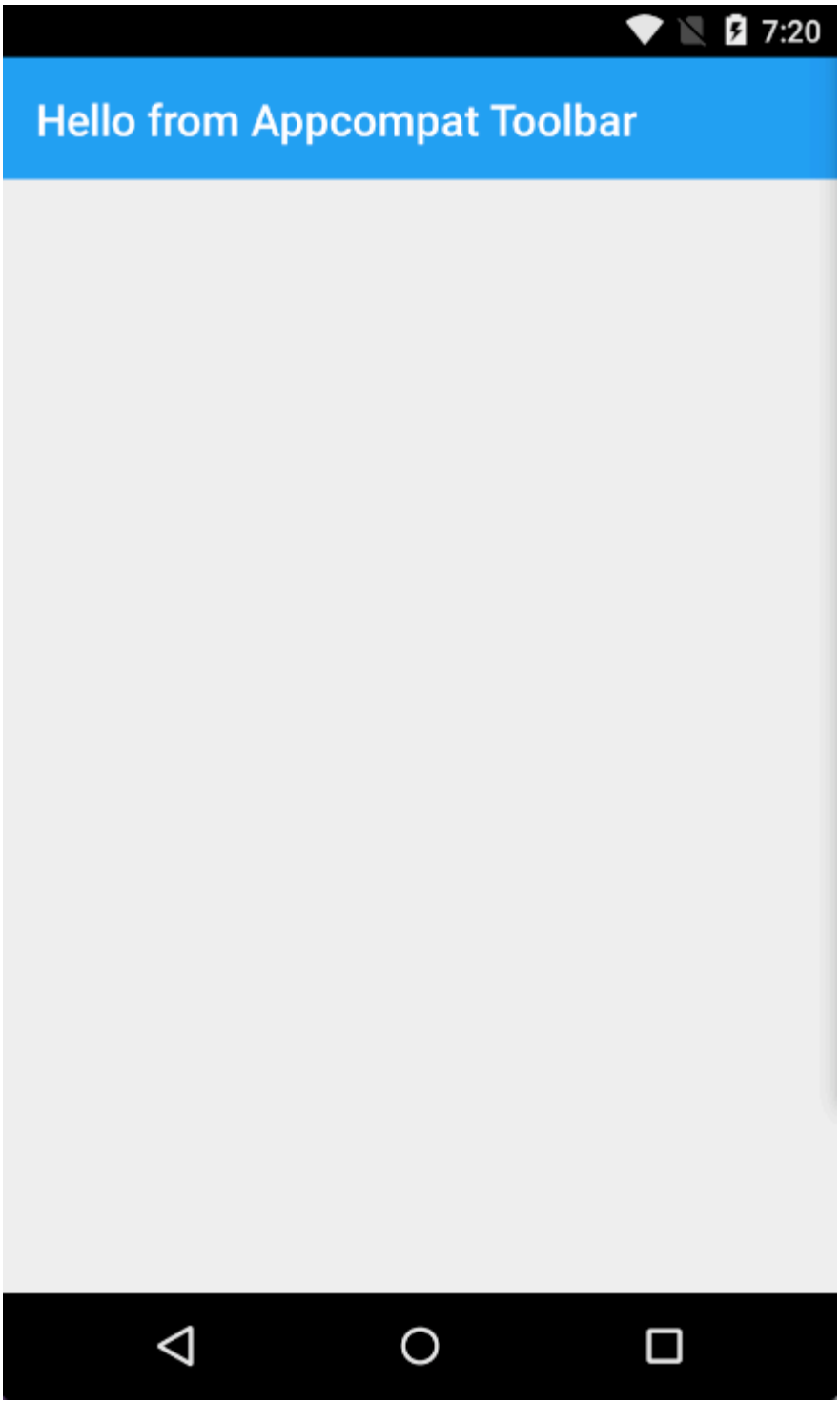
L'ultimo passaggio consiste nel connettere la barra degli strumenti nel file Attività. Aprire il file "MainActivity.cs". È necessario modificare la derivazione da "Attività" a "AppCompatActivity". Ora ottieni un riferimento alla barra degli strumenti e impostalo come barra degli strumenti predefinita per l'attività nel metodo "OnCreate". Puoi anche definire il titolo:

```
var toolbar = FindViewById<Android.Support.V7.Widget.Toolbar>(Resource.Id.toolbar);
        SetSupportActionBar(toolbar);
        SupportActionBar.Title = "Hello from Appcompat Toolbar";
```

Il metodo completo dovrebbe apparire come di seguito:

```
protected override void OnCreate(Bundle savedInstanceState)
    {
        base.OnCreate(savedInstanceState);
        SetContentView(Resource.Layout.Main);

        var toolbar = FindViewById<Android.Support.V7.Widget.Toolbar>(Resource.Id.toolbar);
        SetSupportActionBar(toolbar);
        SupportActionBar.Title = "Hello from Appcompat Toolbar";
    }
```

Ricrea il progetto e avvialo per vedere il risultato:



Leggi Xamarin.Android - Come creare una barra degli strumenti online: https://riptutorial.com/it/xamarin-android/topic/4755/xamarin-android---come-creare-una-barra-degli-strumenti

Capitolo 13: Xamarin.Android - Comunicazione Bluetooth

introduzione

In **Xamarin.Android** le proprietà **BluetoothSocket.InputStream** e **BluetoothSocket.OutputStream** sono progettate automaticamente in **System.IO.Stream** . Nel caso del cosiddetto protocollo di comunicazione interattiva, quando il server risponde solo quando il client parla con esso, System.IO.Stream non va bene perché non ha alcun metodo o proprietà per ottenere il numero di byte di risposta disponibili prima di leggere la risposta.

Parametri

Parametro	Dettagli
presa di corrente	Un'istanza dell'oggetto BluetoothSocket. Il socket deve essere aperto prima di chiamare questo metodo.
cmd	Comando come array di byte da inviare al dispositivo BT.
_mx	Poiché questo metodo utilizza una risorsa hardware, è preferibile chiamarlo da un thread di lavoro separato. Questo parametro è un'istanza dell'oggetto System.Threading.Mutex e viene utilizzato per sincronizzare il thread con altri thread che facoltativamente chiamano questo metodo.
tempo scaduto	Tempo di attesa in millisecondi tra operazioni di scrittura e lettura.

Examples

Invia e ricevi dati da e verso dispositivi Bluetooth tramite socket

L'esempio seguente utilizza i tipi Android.Runtime.InputStreamInvoker e Android.Runtime.OutputStreamInvoker per ottenere Java.IO.InputStream e Java.IO.OutputStream . Una volta che abbiamo un'istanza **Java.IO.InputStream** , possiamo usare il suo metodo **.Available ()** per ottenere il numero di byte di risposta disponibili che possiamo usare nel metodo **.Read ()** :

```
byte[] Talk2BTsocket(BluetoothSocket socket, byte[] cmd, Mutex _mx, int timeOut = 150)
{
    var buf = new byte[0x20];

    _mx.WaitOne();
    try
```

```
    {
        using (var ost = socket.OutputStream)
        {
            var _ost = (ost as OutputStreamInvoker).BaseOutputStream;
            _ost.Write(cmd, 0, cmd.Length);
        }

        // needed because when skipped, it can cause no or invalid data on input stream
        Thread.Sleep(timeOut);

        using (var ist = socket.InputStream)
        {
            var _ist = (ist as InputStreamInvoker).BaseInputStream;
            var aa = 0;
            if ((aa = _ist.Available()) > 0)
            {
                var nn = _ist.Read(buf, 0, aa);
                System.Array.Resize(ref buf, nn);
            }
        }
    }
    catch (System.Exception ex)
    {
        DisplayAlert(ex.Message);
    }
    finally
    {
        _mx.ReleaseMutex();     // must be called here !!!
    }

    return buf;
}
```

Leggi Xamarin.Android - Comunicazione Bluetooth online: https://riptutorial.com/it/xamarin-android/topic/10844/xamarin-android---comunicazione-bluetooth

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con Xamarin.Android	Amy Burns, Community, Jon Douglas, Kevin Montrose, Ryan Weaver
2	Attacchi	EJoshuaS, Jon Douglas, jonp, Matthew, Prashant C, Sven-Michael Stübe
3	brindisi	GONeale, Matthew, Piet, user2912553
4	Ciclo di vita dell'app - Xamarin.Andorid	CDrosos, Daniel Krzyczkowski, Steven Mark Ford
5	Come correggere l'orientamento di un'immagine catturata dal dispositivo Android	Daniel Krzyczkowski
6	Finestre di dialogo	JimBobBennett, Pilatus
7	ListView personalizzato	user3814750
8	Pubblicare il tuo APK Xamarin.Android	Alexandre
9	RecyclerView	Alexandre, Matthew, Ryan Alford, Sreeraj, Zverev Eugene
10	Scansione di codici a barre tramite la libreria ZXing in Xamarin Applications	GvSharma
11	Xamarin.Android - Come creare una barra degli strumenti	Daniel Krzyczkowski, tylerjgarland
12	Xamarin.Android - Comunicazione Bluetooth	Ladislav