EBook Gratis

APRENDIZAJE Xamarin.Android

Free unaffiliated eBook created from
Stack Overflow contributors.

#xamarin.android

Tabla de contenido

Acerca de

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

Capítulo 1: Empezando con Xamarin.Android

Observaciones

Xamarin.Android le permite crear aplicaciones nativas de Android utilizando los mismos controles de interfaz de usuario que en Java, excepto con la flexibilidad y elegancia de un lenguaje moderno (C #), el poder de la Biblioteca de clases base (BCL) .NET y dos IDEs de primera clase - Xamarin Studio y Visual Studio - al alcance de su mano.

Para obtener más información sobre la instalación de Xamarin.Android en su máquina Mac o Windows, consulte las guías de introducción en el centro de desarrolladores de Xamarin

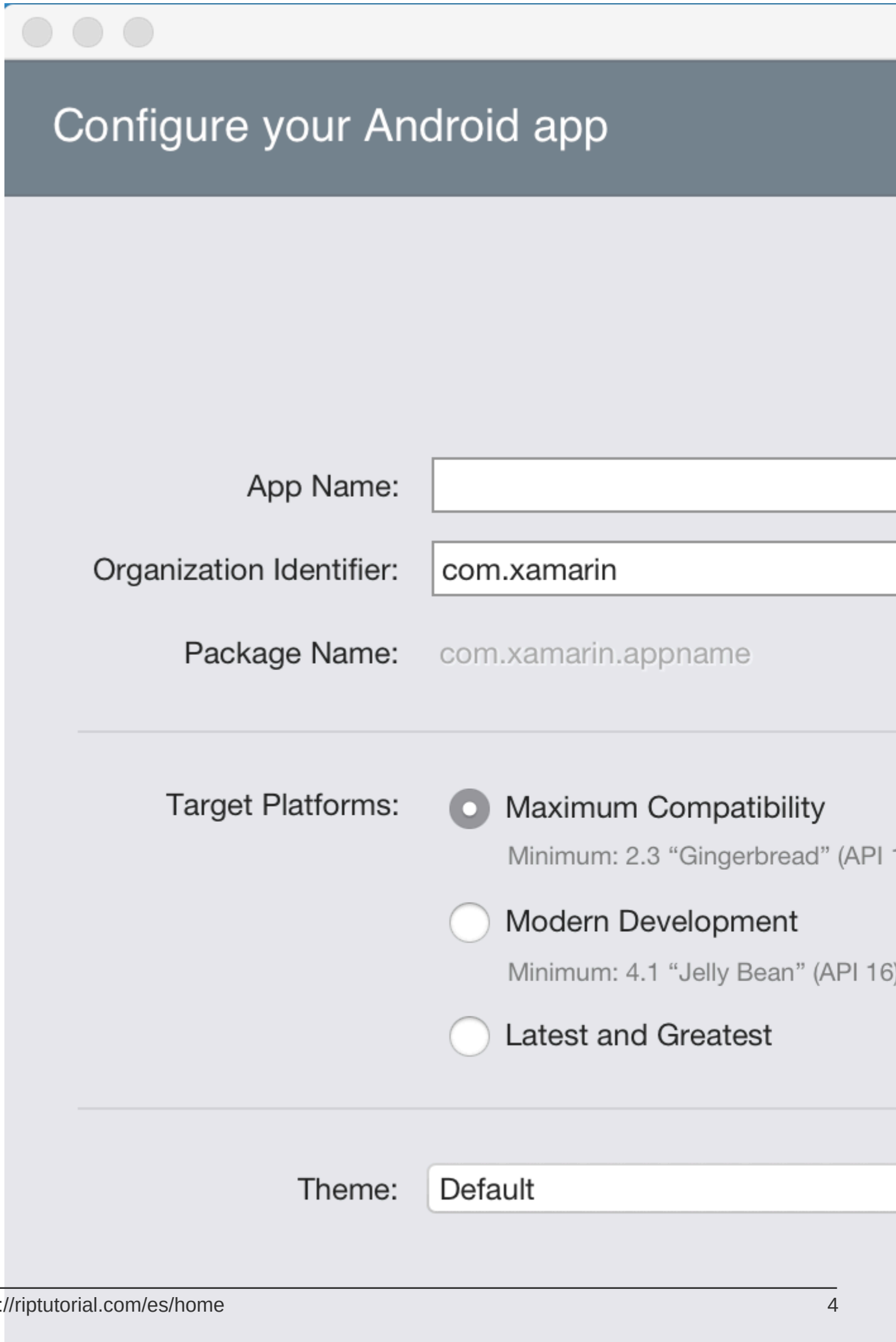
Versiones

Versión	Nombre clave	Nivel de API	Fecha de lanzamiento
1.0	Ninguna	1	2008-09-23
1.1	Ninguna	2	2009-02-09
1.5	Magdalena	3	2009-04-27
1.6	Rosquilla	4	2009-09-15
2.0-2.1	Eclair	5-7	2009-10-26
2.2-2.2.3	Froyo	8	2010-05-20
2.3-2.3.7	Pan de jengibre	9-10	2010-12-06
3.0-3.2.6	Panal	11-13	2011-02-22
4.0-4.0.4	Sandwich De Helado	14-15	2011-10-18
4.1-4.3.1	Frijol de jalea	16-18	2012-07-09
4.4-4.4.4, 4.4W-4.4W.2	Kit Kat	19-20	2013-10-31
5.0-5.1.1	Pirulí	21-22	2014-11-12
6.0-6.0.1	Malvavisco	23	2015-10-05
7.0	Turrón	24	2016-08-22

Examples

Empieza en Xamarin Studio

1. Busque **Archivo> Nuevo> Solución** para abrir el cuadro de diálogo del nuevo proyecto.
2. Seleccione la **aplicación de Android** y presione **Siguiente** .
3. Configure su aplicación configurando su nombre de aplicación y su ID de organización. Seleccione la plataforma de destino más adecuada para sus necesidades, o déjela como la predeterminada. Presione Siguiente:

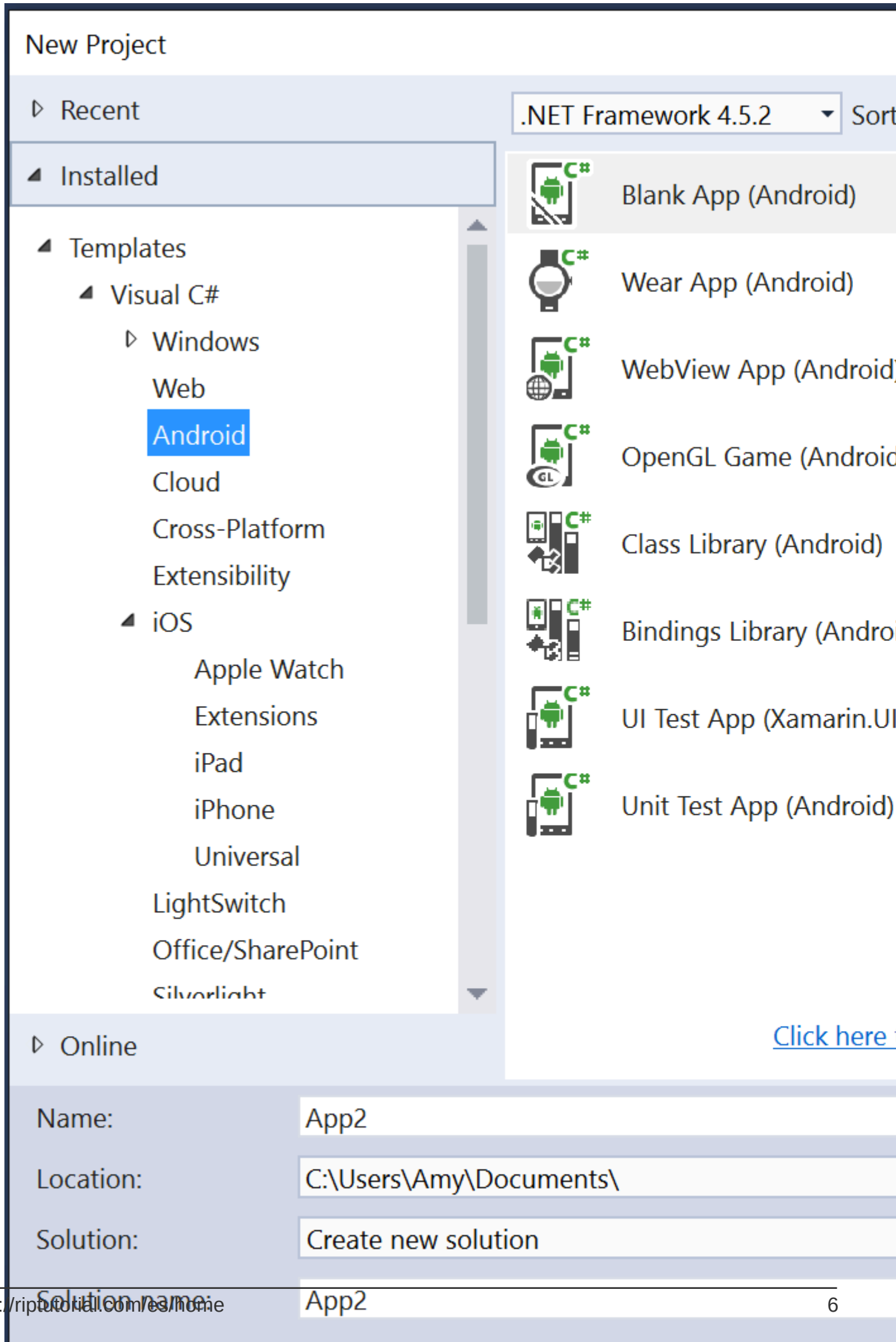

Configure your Android app
App Name:
Organization Identifier:
com.xamarin
Package Name:
com.xamarin.appname
Target Platforms:
Maximum Compatibility
Minimum: 2.3 “Gingerbread” (API
Modern Development
Minimum: 4.1 “Jelly Bean” (API 16)
Latest and Greatest
Theme:
Default

4. Establezca el nombre de su proyecto y el nombre de la solución, o déjelo como el nombre predeterminado. Haga clic en Crear para crear su proyecto.
5. Configure su dispositivo para la implementación , o configure un emulador
6. Para ejecutar su aplicación, seleccione la configuración de **depuración** y presione el botón

   Reproducir:
   y presione el botón Reproducir:

Empezar en Visual Studio

1. Vaya a **Archivo> Nuevo> Proyecto** para abrir el cuadro de diálogo Nuevo proyecto.
2. Navegue a **Visual C #> Android** y seleccione la aplicación en blanco:


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
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)
Online
Click here
Name:
App2
Location:
C:\Users\Amy\Documents\
Solution:
Create new solution
Solution name:
App2

3. Asigne un **nombre a** su aplicación y presione **OK** para crear su proyecto.
4. Configure su dispositivo para la implementación , o configure un emulador
5. Para ejecutar su aplicación, seleccione la configuración de **depuración** y presione el botón

Inicio :

Inicio : 

Lea Empezando con Xamarin.Android en línea: https://riptutorial.com/es/xamarin-android/topic/403/empezando-con-xamarin-android

Capítulo 2: Ciclo de vida de la aplicación - Xamarin.Andorid

Introducción

El ciclo de vida de la aplicación Xamarin.Android es el mismo que la aplicación normal de Android. Al hablar sobre el ciclo de vida, necesitamos hablar sobre: Ciclo de vida de la aplicación, Ciclo de vida de la actividad y Ciclo de vida de los fragmentos.

A continuación, trataré de proporcionar una buena descripción y la forma de usarlos. Obtuve esta documentación de la documentación oficial de Android y Xamarin y muchos recursos web útiles proporcionados en la sección de comentarios a continuación.

Observaciones

Enlaces interesantes para ampliar su conocimiento sobre el ciclo de vida de la aplicación de 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 de vida de la aplicación

En primer lugar, debe saber que puede extender la clase de aplicación Android.Aplication para que pueda acceder a dos métodos importantes relacionados con el ciclo de vida de la aplicación:

- OnCreate: se llama cuando se inicia la aplicación, antes de que se hayan creado otros objetos de la aplicación (como MainActivity).
- OnTerminate: este método es para uso en entornos de procesos emulados. Nunca se llamará a un dispositivo Android de producción, donde los procesos se eliminan simplemente matándolos; No se ejecuta ningún código de usuario (incluida esta devolución de llamada) al hacerlo. De la documentación:

https://developer.android.com/reference/android/app/Application.html#onTerminate ()

En la aplicación Xamarin.Android, puede extender la clase de aplicación de la forma que se presenta a continuación. Agregue una nueva clase llamada "MyApplication.cs" a su proyecto:

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

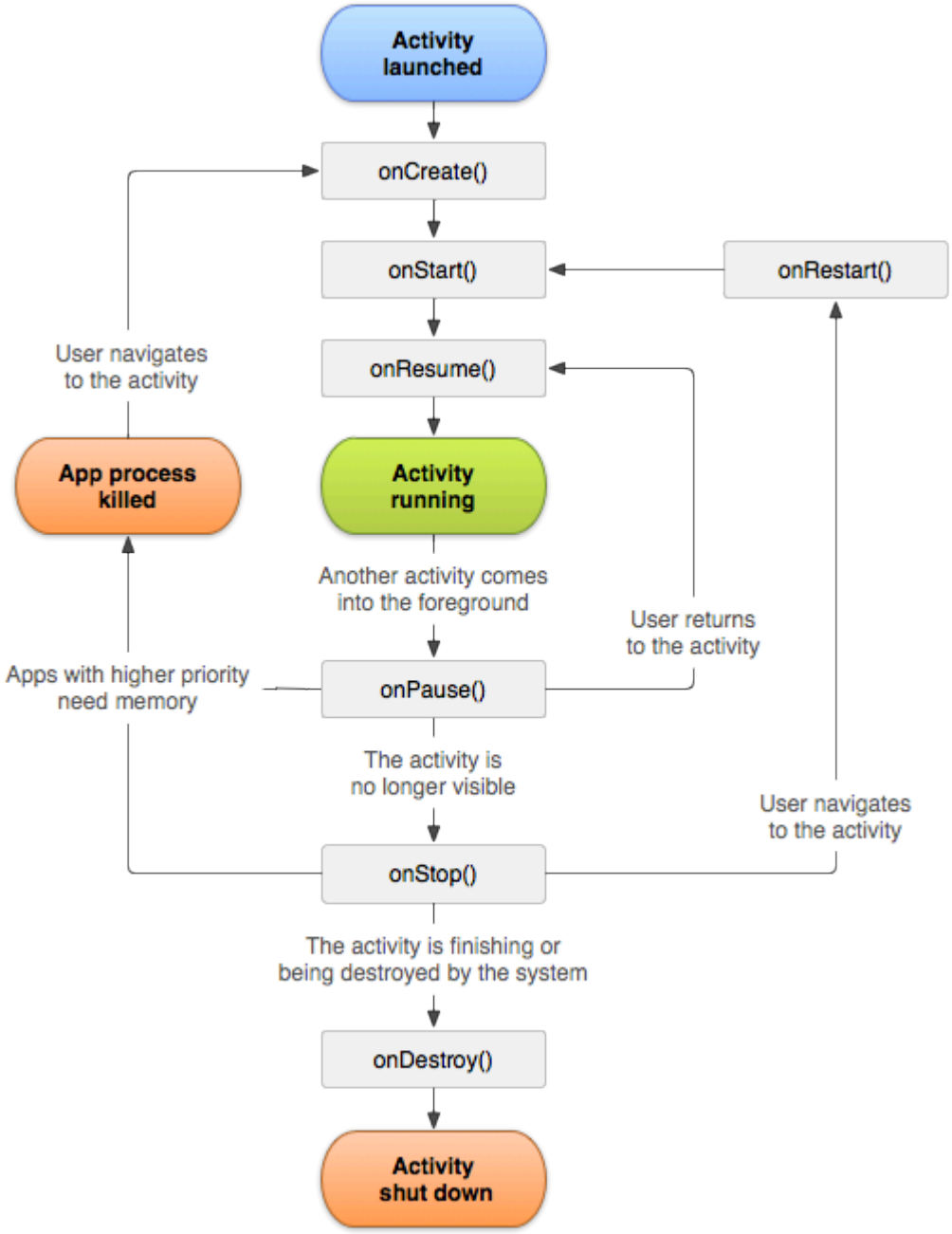
Como escribiste anteriormente puedes usar el método OnCreate. Por ejemplo, puede inicializar la base de datos local aquí o configurar alguna configuración adicional.

También hay más métodos que se pueden anular, como: OnConfigurationChanged o OnLowMemory.

Ciclo de vida de la actividad

El ciclo de vida de la actividad es bastante más complejo. Como saben, Activity es una página única en la aplicación de Android donde el usuario puede interactuar con ella.

En el diagrama a continuación puedes ver cómo se ve el ciclo de vida de la actividad de Android:



Como puede ver, hay un flujo específico del ciclo de vida de la actividad. En la aplicación móvil, tienes métodos de curso en cada clase de actividad que manejan un fragmento de ciclo de vida específico:

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

Hay una buena descripción en la documentación oficial de Android:

- Toda la vida útil de una actividad transcurre desde la primera llamada a onCreate (Bundle) hasta una única llamada final a onDestroy (). Una actividad realizará toda la configuración del estado "global" en onCreate () y liberará todos los recursos restantes en onDestroy (). Por ejemplo, si tiene un hilo ejecutándose en segundo plano para descargar datos de la red, puede crear ese hilo en onCreate () y luego detener el hilo en onDestroy ().

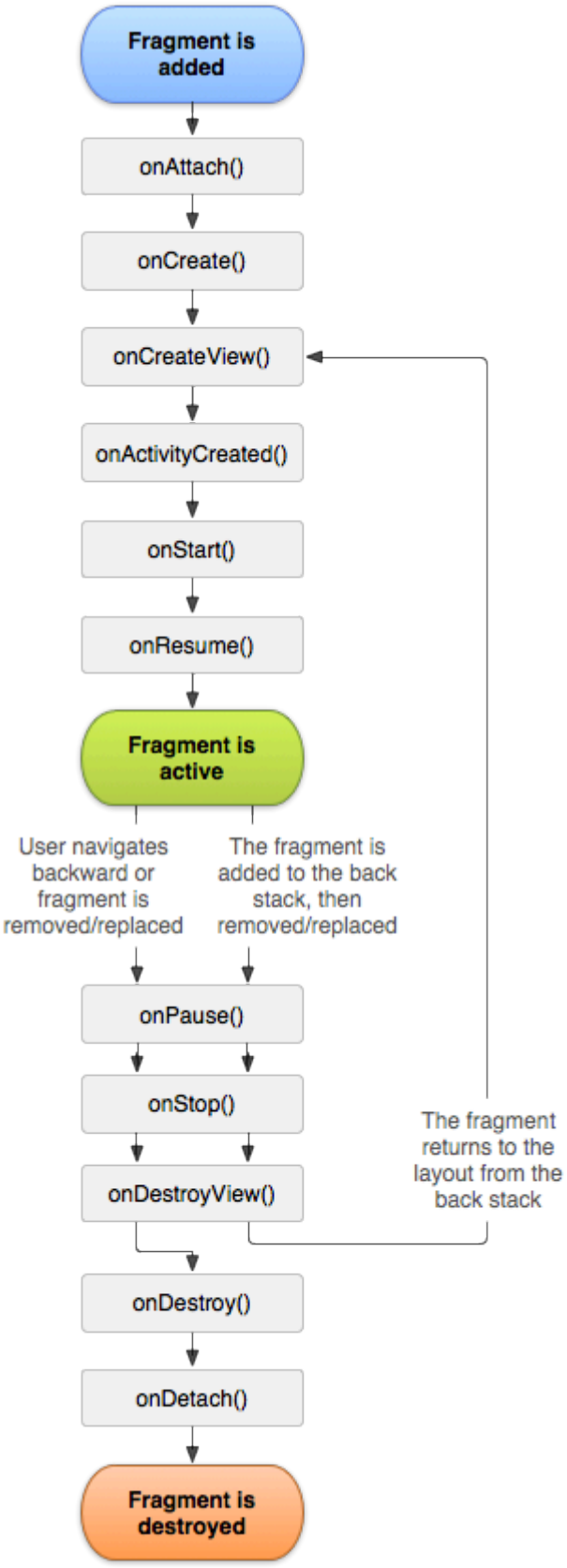
- El tiempo de vida visible de una actividad ocurre entre una llamada a onStart () hasta una llamada correspondiente a onStop (). Durante este tiempo, el usuario puede ver la actividad en pantalla, aunque es posible que no esté en primer plano y no interactúe con el usuario. Entre estos dos métodos puede mantener los recursos necesarios para mostrar la actividad al usuario. Por ejemplo, puede registrar un BroadcastReceiver en onStart () para monitorear los cambios que afectan su UI, y anular el registro en onStop () cuando el usuario ya no vea lo que está mostrando. Los métodos onStart () y onStop () se pueden llamar varias veces, a medida que la actividad se hace visible y oculta para el usuario.

- El tiempo de vida de una actividad en primer plano ocurre entre una llamada a onResume () hasta una llamada correspondiente a onPause (). Durante este tiempo, la actividad está delante de todas las demás actividades e interactúa con el usuario. Con frecuencia, una actividad puede ir entre los estados de reanudación y pausa, por ejemplo, cuando el dispositivo entra en modo de suspensión, cuando se entrega un resultado de actividad, cuando se entrega una nueva intención, por lo que el código de estos métodos debe ser bastante ligero.

Ciclo de vida del fragmento

Como usted sabe, puede tener una actividad pero diferentes fragmentos incrustados en ella. Es por eso que el ciclo de vida del fragmento también es importante para los desarrolladores.

En el diagrama a continuación, puede ver cómo se ve el ciclo de vida del fragmento de Android:



Como se describe en la documentación oficial de Android, debe implementar al menos tres métodos a continuación:

- OnCreate: el sistema lo llama al crear el fragmento. Dentro de su implementación, debe inicializar los componentes esenciales del fragmento que desea conservar cuando el fragmento se detiene o se detiene, y luego se reanuda.
- OnCreateView: el sistema lo llama cuando llega el momento de que el fragmento dibuje su interfaz de usuario por primera vez. Para dibujar una IU para su fragmento, debe devolver una Vista desde este método que es la raíz del diseño de su fragmento. Puede devolver nulo si el fragmento no proporciona una interfaz de usuario.
- OnPause: el sistema llama a este método como la primera indicación de que el usuario está dejando el fragmento (aunque no siempre significa que se está destruyendo el fragmento). Por lo general, aquí es donde debe realizar cualquier cambio que deba persistir más allá de la sesión del usuario actual (porque es posible que el usuario no regrese).

Aquí está la implementación de ejemplo en 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();
    }
}
```

Por supuesto, puede agregar métodos adicionales aquí si desea manejar diferentes estados.

Muestra completa en GitHub

Si desea obtener el proyecto base con los métodos que se describen a continuación, puede

descargar la plantilla de la aplicación Xamarin.Android desde mi GitHub. Puedes encontrar ejemplos para:

- Métodos de ciclo de vida de la aplicación.
- Métodos del ciclo de vida de la actividad.
- Métodos del ciclo de vida del fragmento.

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

Lea Ciclo de vida de la aplicación - Xamarin.Andorid en línea: https://riptutorial.com/es/xamarin-android/topic/8842/ciclo-de-vida-de-la-aplicacion---xamarin-andorid

Capítulo 3: Cómo corregir la orientación de una imagen capturada desde un dispositivo Android

Observaciones

1. Esta muestra de aplicación está disponible en mi GitHub a continuación:

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

2. La documentación del componente móvil de Xamarin está disponible a continuación:

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

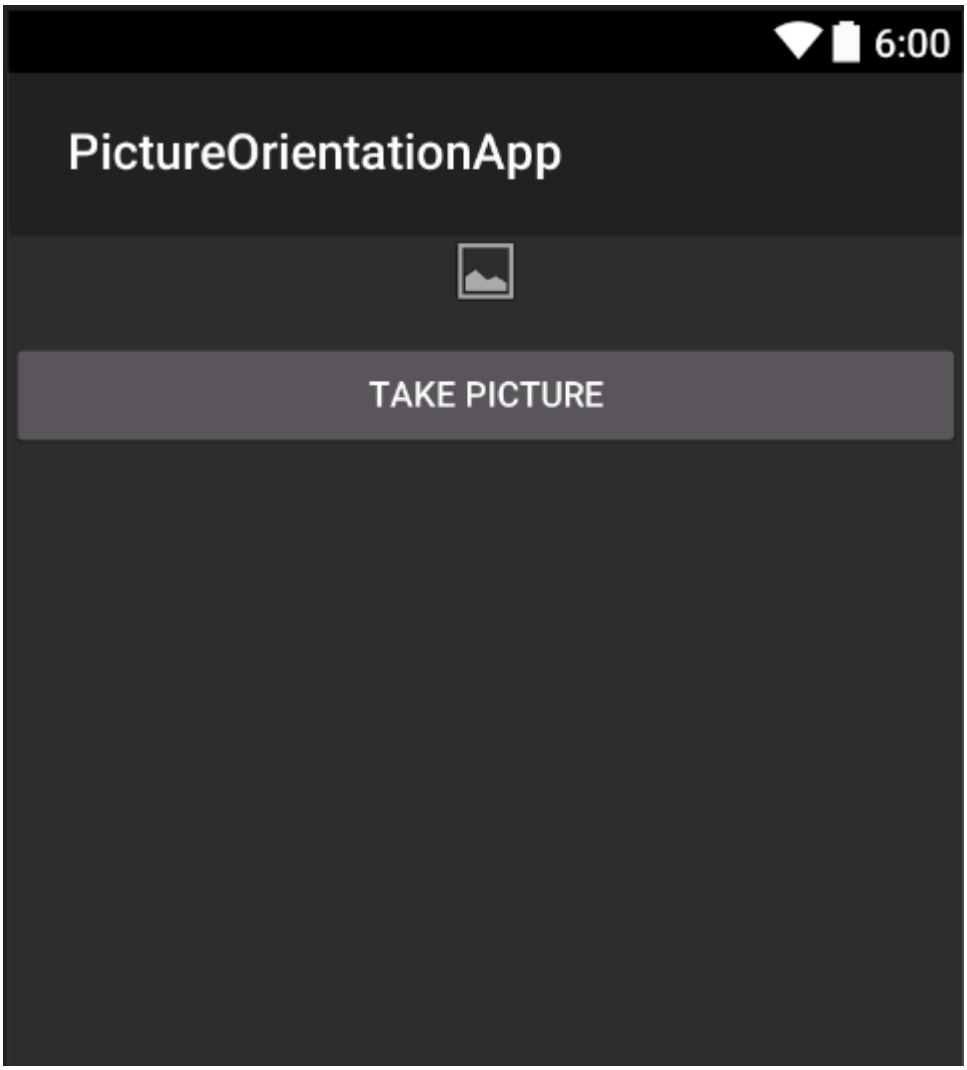
Examples

Cómo corregir la orientación de una imagen capturada desde un dispositivo Android

Este ejemplo muestra cómo tomar una imagen y mostrarla correctamente en el dispositivo Android.

Primero, tenemos que crear una aplicación de ejemplo con un botón y una vista de imagen. Una vez que el usuario hace clic en el botón, la cámara se inicia y una vez que el usuario selecciona la imagen, se mostrará con la orientación correcta en la pantalla.

1. Agregue el botón llamado "TakePictureButton" y la vista de imagen llamada "TakenPictureImageView":

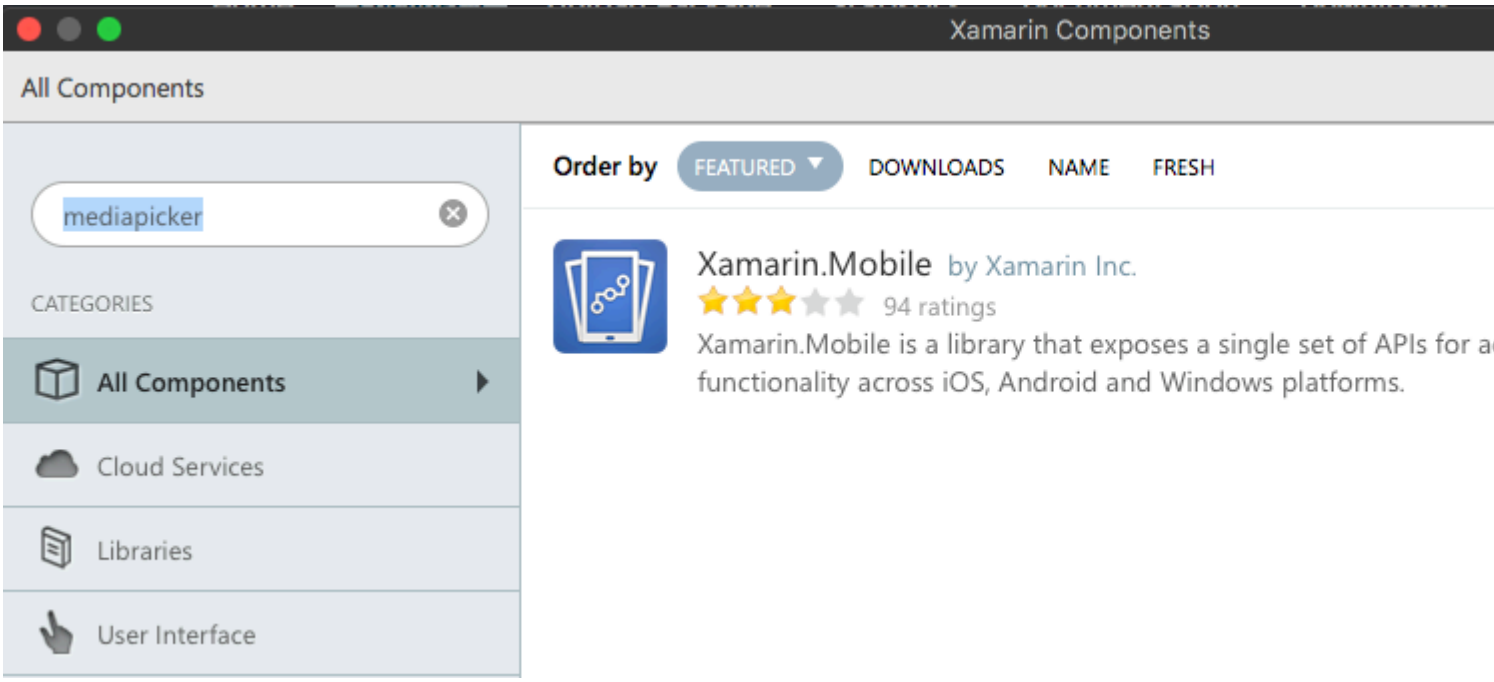


2. Ahora abre el código de actividad detrás:

Aquí en primer lugar obtener referencia a sus controles:

```
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. En nuestra aplicación, usaremos el componente de Xamarin Mobile disponible en la tienda de componentes:



4. Una vez que lo agregues al proyecto podemos seguir adelante. Agregue el siguiente código que es responsable de lanzar la cámara. Este método debe invocarse en el botón de clic como se puede ver en el código anterior:

```
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 vez que el usuario toma la foto, debemos mostrarla en la orientación correcta. Para hacerlo utiliza el siguiente método. Es responsable de recuperar la información exif de la imagen tomada (incluida la orientación durante el momento de tomar la fotografía) y de crear un mapa de bits con la orientación correcta:

```
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;
```

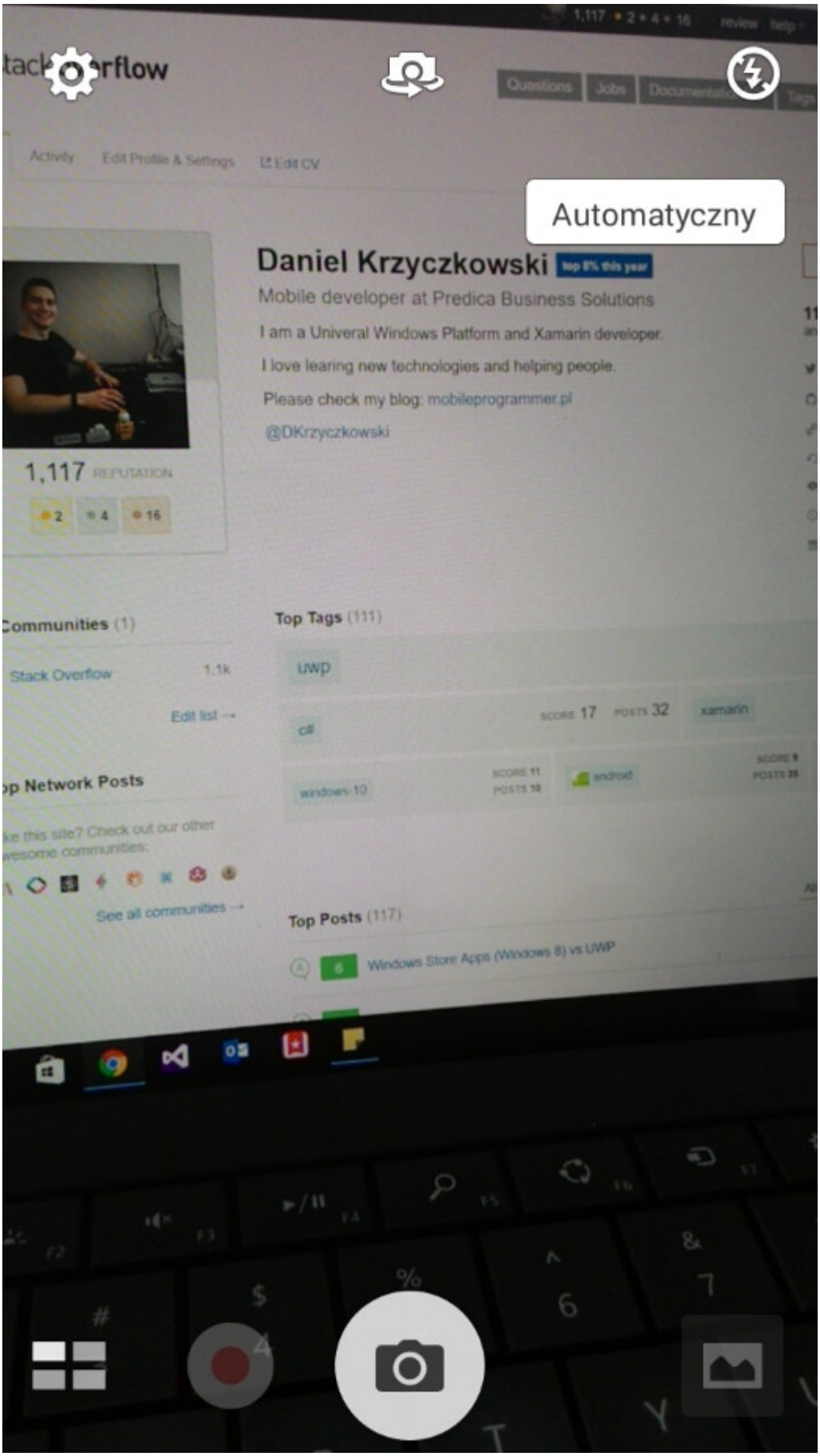
```
        height_tmp /= 2;
        scale++;
    }

    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;
```

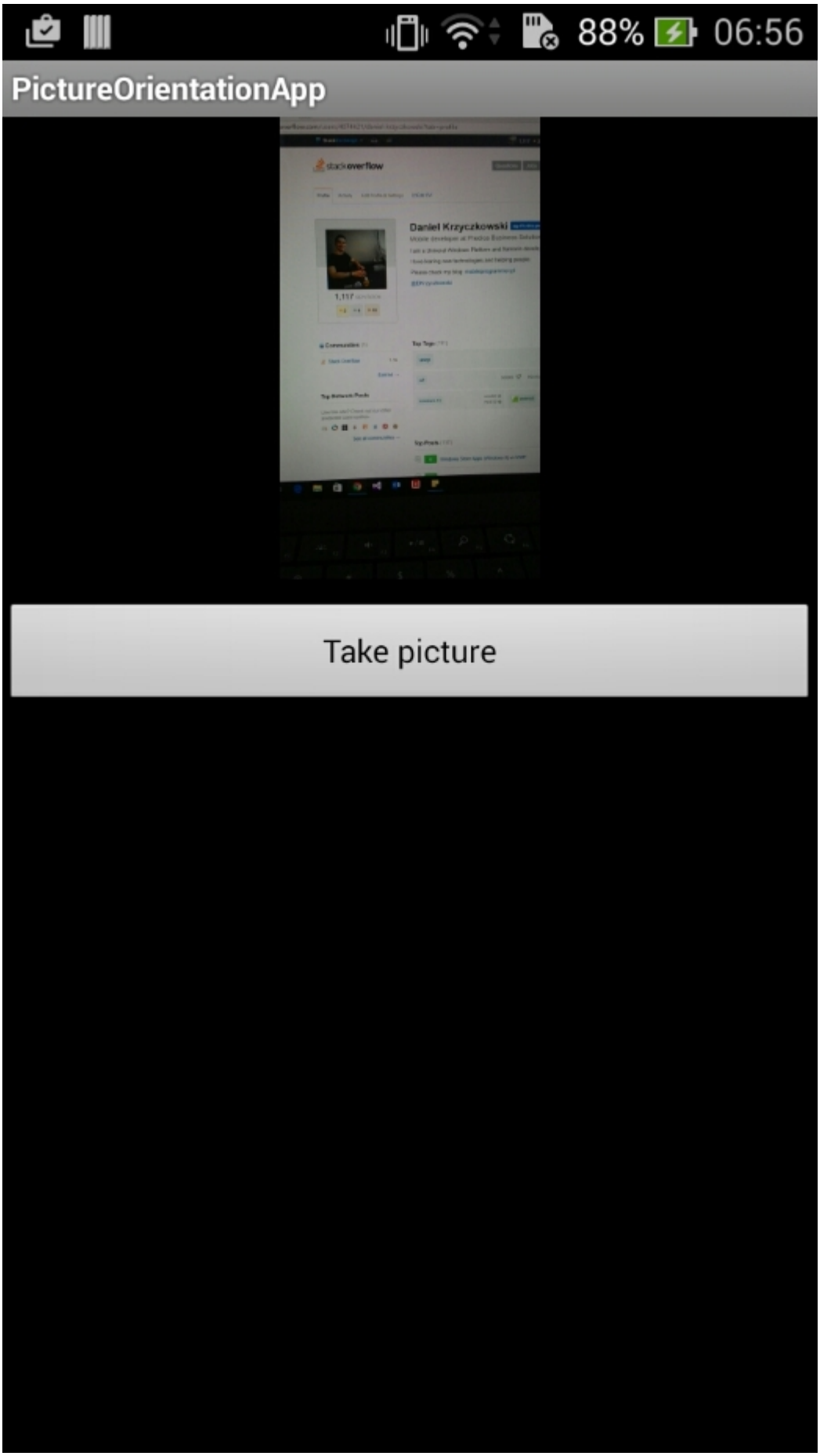
```
        }

        return resizedBitmap;
    }

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

6. El método anterior debe invocarse en el método OnActivityResult invocado después de que el usuario tome la foto:

```
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. Iniciar la aplicacion. Tome una foto y vea el resultado:


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
android
Top Posts (117)
Windows Store Apps (Windows 8) vs UWP



Eso es. Ahora tendrá toda la fotografía que tomó en la orientación correcta.

Lea Cómo corregir la orientación de una imagen capturada desde un dispositivo Android en línea: https://riptutorial.com/es/xamarin-android/topic/6683/como-corregir-la-orientacion-de-una-imagen-capturada-desde-un-dispositivo-android

Capítulo 4: Diálogos

Observaciones

Configuración del `Context` del diálogo

Al crear un `Dialog` desde una `Activiy` , podemos usar `this` como contexto.

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

Con `Fragments` utilizamos la propiedad `Context` .

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

Tipos de botones

`SetNeutralButton()` se puede utilizar para una simple notificación y confirmación de que se ha leído la notificación. `SetPositiveButton()` se puede usar para una confirmación, por ejemplo: "¿Está seguro de que desea eliminar este elemento?" `SetNegativeButton()` es para `SetNegativeButton()` el diálogo y cancelar su acción.

Desactivar cancelar desde botón trasero

Si queremos asegurarnos de que el usuario no puede descartar el cuadro de diálogo con el botón de retroceso, podemos llamar a `SetCanceable(false)` . Esto solo funciona para el botón de retroceso.

Rotación

Si se gira la pantalla mientras se muestra un cuadro de diálogo, se cerrará y no se activarán las acciones Aceptar y Cancelar. Deberá manejar esto dentro de su actividad y volver a mostrar el diálogo después de que la actividad haya sido recargada.

Para evitar esto usa un `DialogFragment` en `DialogFragment` lugar.

Examples

Diálogo de alerta

Creando un diálogo de alerta

```
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();
```

Lea Diálogos en línea: https://riptutorial.com/es/xamarin-android/topic/2510/dialogos

Capítulo 5: Diálogos

Parámetros

Método público de uso común	Utilizar
SetTitle (String)	Establece el título para el diálogo
SetIcon (Dibujable)	Establecer icono para el diálogo de alerta
SetMessage (cadena)	Establezca el mensaje para mostrar.
SetNegativeButton (String, EventHandler)	Configure un oyente para que se invoque cuando se presiona el botón negativo del diálogo.
SetPositiveButton (String, EventHandler)	Configure un oyente para que se invoque cuando se presiona el botón positivo del diálogo.
SetNeutralButton (String, EventHandler)	Configure un oyente para que se invoque cuando se presiona el botón neutral del diálogo.
SetOnCancelListener (IDialogInterfaceOnCancelListener)	Establece la devolución de llamada que se llamará si se cancela el diálogo.
SetOnDismissListener (IDialogInterfaceOnDismissListener)	Establece la devolución de llamada que se llamará cuando el cuadro de diálogo se cierre por cualquier motivo.
Show()	Crea un AlertDialog con los argumentos proporcionados a este constructor y Dialog.Show es el diálogo.

Observaciones

Requerimientos

Espacio de nombres: Android.App

Ensamblaje: Mono.Android (en Mono.Android.dll)

Versiones de montaje: 0.0.0.0

Constructores publicos

AlertDialog.Builder (Contexto): -

El constructor utiliza un contexto para este constructor y el AlertDialog que crea.

AlertDialog.Builder (Context, Int32): -

El constructor utiliza un contexto y un tema para este constructor y el AlertDialog que crea.

Usando Material Design AlertDialog

Para utilizar el moderno AlertDialog:

1. Instale la biblioteca AppCompat de Support v7 desde los paquetes de NuGet
2. Reemplace AlertDialog con Android.Support.V7.App.AlertDialog o agregue la siguiente declaración en la parte superior para hacer que su diálogo brille.

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

Examples

AlertDialog

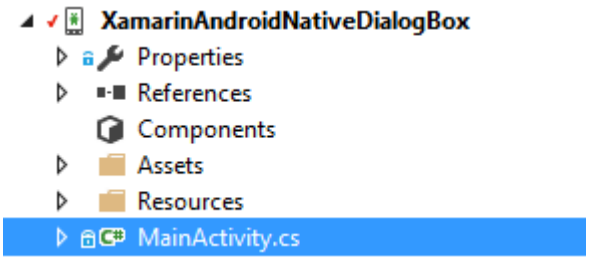
```
// 1. Instantiate an AlertDialog.Builder with its constructor
// the parameter this is the context (usually your activity)
AlertDialog.Builder builder = new AlertDialog.Builder(this);

// 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();
```

Ejemplo de diálogo de alerta simple

Crearemos un simple diálogo de alerta en Xamarin.Android

Ahora teniendo en cuenta que ha seguido la guía de introducción de la documentación.

Debes tener la estructura del proyecto así:



Tu actividad principal debe verse así:

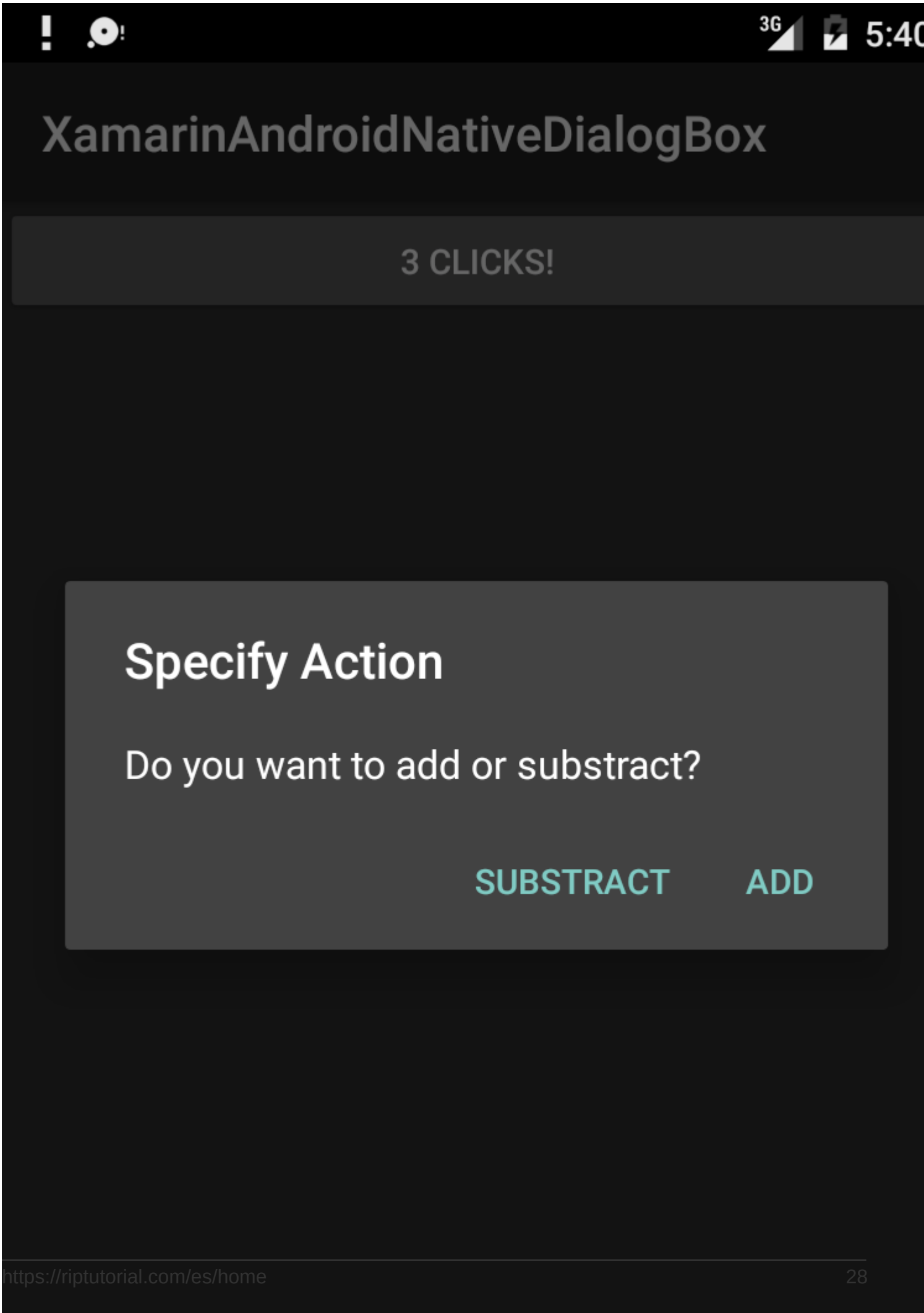
```
public class MainActivity : Activity
{
int count = 1;

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++); };
}
}
```

Ahora lo que haremos es, en lugar de agregar uno al contador al hacer clic en el botón, le preguntaremos al usuario si desea agregar o restar uno en un simple diálogo de alerta.

Y al hacer clic en el botón Positivo o negativo, tomaremos la acción.

```
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();
};
```

captura de pantalla:


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

Capítulo 6: Escaneado de códigos de barras utilizando la biblioteca ZXing en aplicaciones Xamarin

Introducción

La biblioteca Zxing es bien conocida por el procesamiento de imágenes. Zxing se basó en java y el módulo .Net también está disponible y se puede usar en aplicaciones de xamarin. Haga clic aquí para consultar la documentación oficial. http://zxingnet.codeplex.com/

Recientemente he usado este libarry.

Paso 1: Agregue el componente ZXing.Net.Mobile a la solución.

paso 2: en la actividad que necesitamos para mostrar el escáner de código de barras, en esa actividad inicialice MobileBarcodeScanner.

Paso 3: escriba el siguiente código cuando se toca en cualquier vista para comenzar a escanear.

Examples

Código de muestra

```
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;
}
```

Lea Escaneado de códigos de barras utilizando la biblioteca ZXing en aplicaciones Xamarin en línea: https://riptutorial.com/es/xamarin-android/topic/9526/escaneado-de-codigos-de-barras-utilizando-la-biblioteca-zxing-en-aplicaciones-xamarin

Capítulo 7: Fijaciones

Examples

Quitando tipos

Es posible dar instrucciones al generador de enlaces de Xamarin.Android para que ignore un tipo de Java y no lo enlace. Esto se hace agregando un elemento XML de `remove-node` al archivo metadata.xml:

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

Implementando interfaces Java

Si una biblioteca java contiene interfaces que deben ser implementadas por el usuario (por ejemplo, escuchas de clic como `View.IOnClickListener` o callbacks), la clase implementadora debe heredar, directa o indirectamente, de `Java.Lang.Object` o `Java.Lang.Throwable` . Este es un error común, porque los pasos del paquete simplemente imprimen una advertencia que se pasa por alto fácilmente:

> El tipo 'MyListener' implementa Android.Runtime.IJavaObject pero no hereda de Java.Lang.Object. No es compatible.

Incorrecto

El uso de esta implementación resultará en un comportamiento inesperado.

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

    public void Dispose()
    {
    }


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

Correcto

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

```
    {
        // ...
    }
}
```

Las bibliotecas de enlaces pueden renombrar métodos e interfaces.

No todo en una biblioteca de enlaces tendrá el mismo nombre en C # que en Java.

En C #, los nombres de las interfaces comienzan con "I", pero Java no tiene tal convención. Cuando importa una biblioteca Java, una interfaz llamada `SomeInterface` se convertirá en `ISomeInterface` .

Del mismo modo, Java no tiene propiedades como C # tiene. Cuando una biblioteca está vinculada, los métodos de obtención y establecimiento de Java se pueden refactorizar como propiedades. Por ejemplo, el siguiente código de Java

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

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

puede ser refactora como

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

cuando esta atado

Lea Fijaciones en línea: https://riptutorial.com/es/xamarin-android/topic/771/fijaciones

Capítulo 8: ListView personalizado

Examples

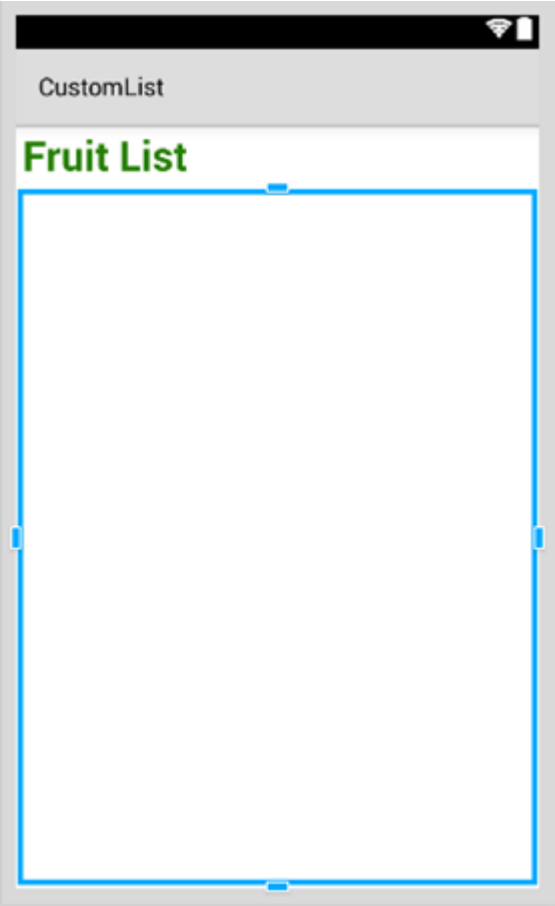
La vista de lista personalizada consta de filas diseñadas según las necesidades de los usuarios.

Para el diseño sobre su archivo customrow.axml es como se muestra a continuación

```
    <?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>
```

Luego puede diseñar su main.axml, que contiene una vista de texto para el encabezado y una vista de lista.



Espero que sea fácil ...

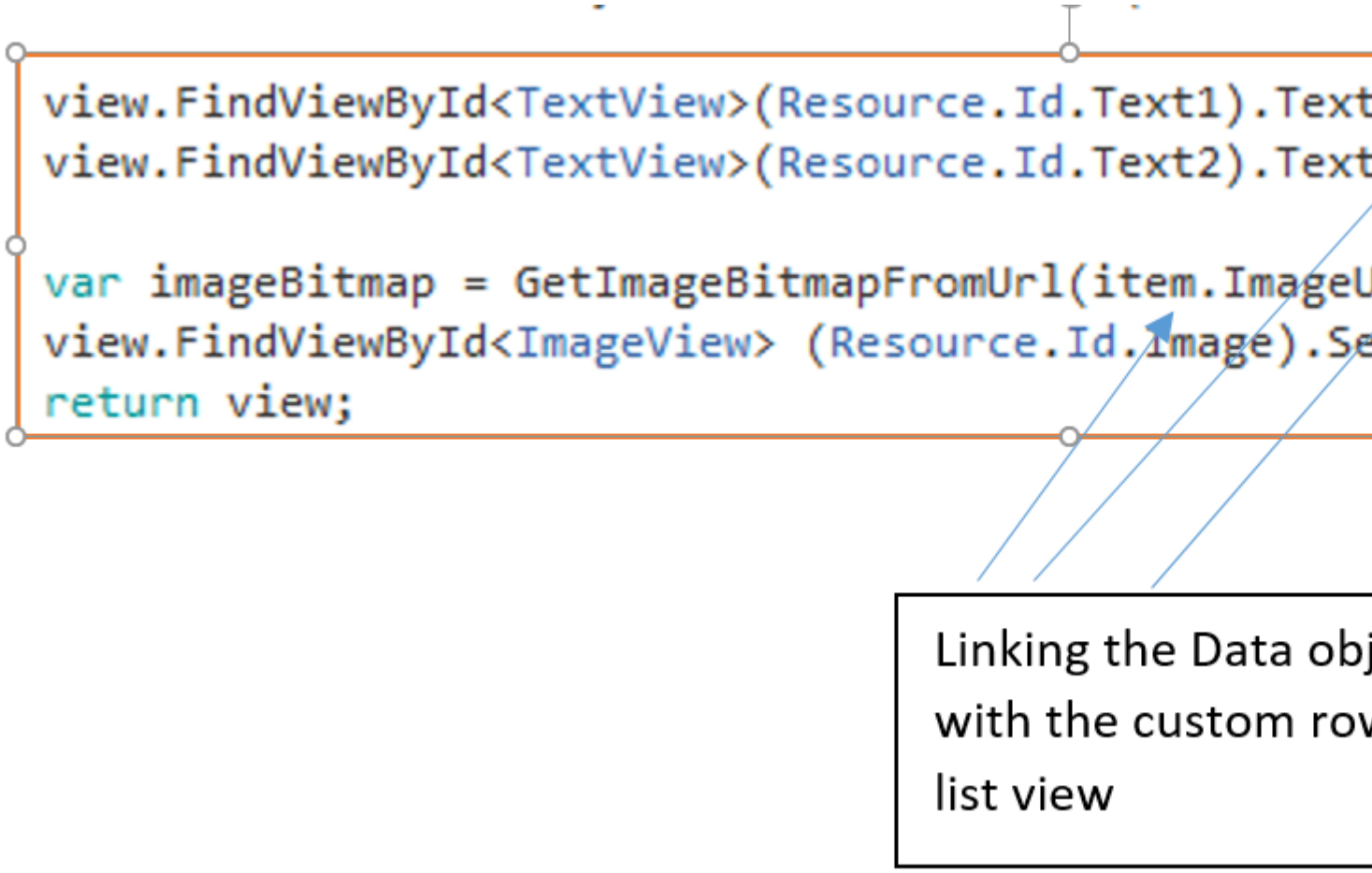
A continuación, crea tu clase Data.cs que representará tus objetos de fila

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

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

A continuación, necesita la clase DataAdapter.cs, los adaptadores vinculan sus datos con la vista subyacente

```
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 más importante está dentro de la función GetView, aquí es donde vincula su objeto a su fila personalizada.



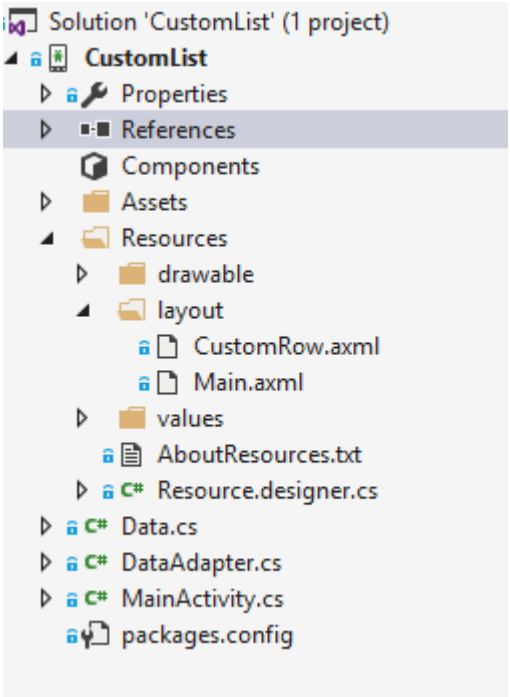
GetImageBitmapFromUrl no es parte del adaptador de datos, pero lo he puesto aquí por simplicidad.

Por fin llegamos a la MainActivity.cs.

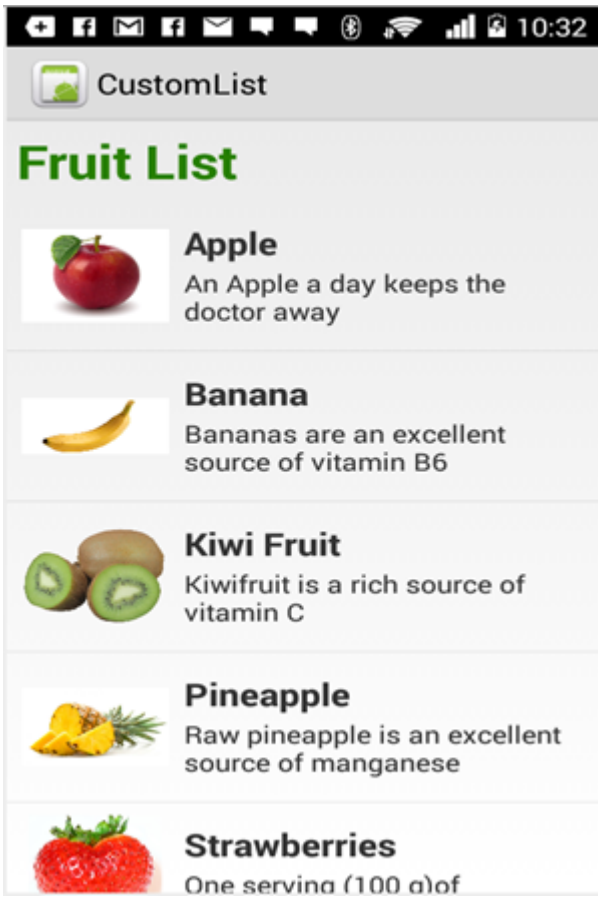
```
public class MainActivity : Activity
{

    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);

    }
```

Su estructura final del proyecto es como se muestra a continuación.



Si todo está bien, debería ver la salida como se muestra.



Lea ListView personalizado en línea: https://riptutorial.com/es/xamarin-android/topic/6406/listview-personalizado

Capítulo 9: Publicando tu Xamarin.Android APK

Introducción

Este tema muestra información sobre cómo preparar su aplicación Xamarin.Android para el modo de lanzamiento y cómo optimizarla.

Examples

Preparando tu APK en el Visual Studio

Terminaste tu aplicación, la probaste en modo de depuración y está funcionando perfectamente. Ahora, desea prepararla para publicar en Google Play Store.

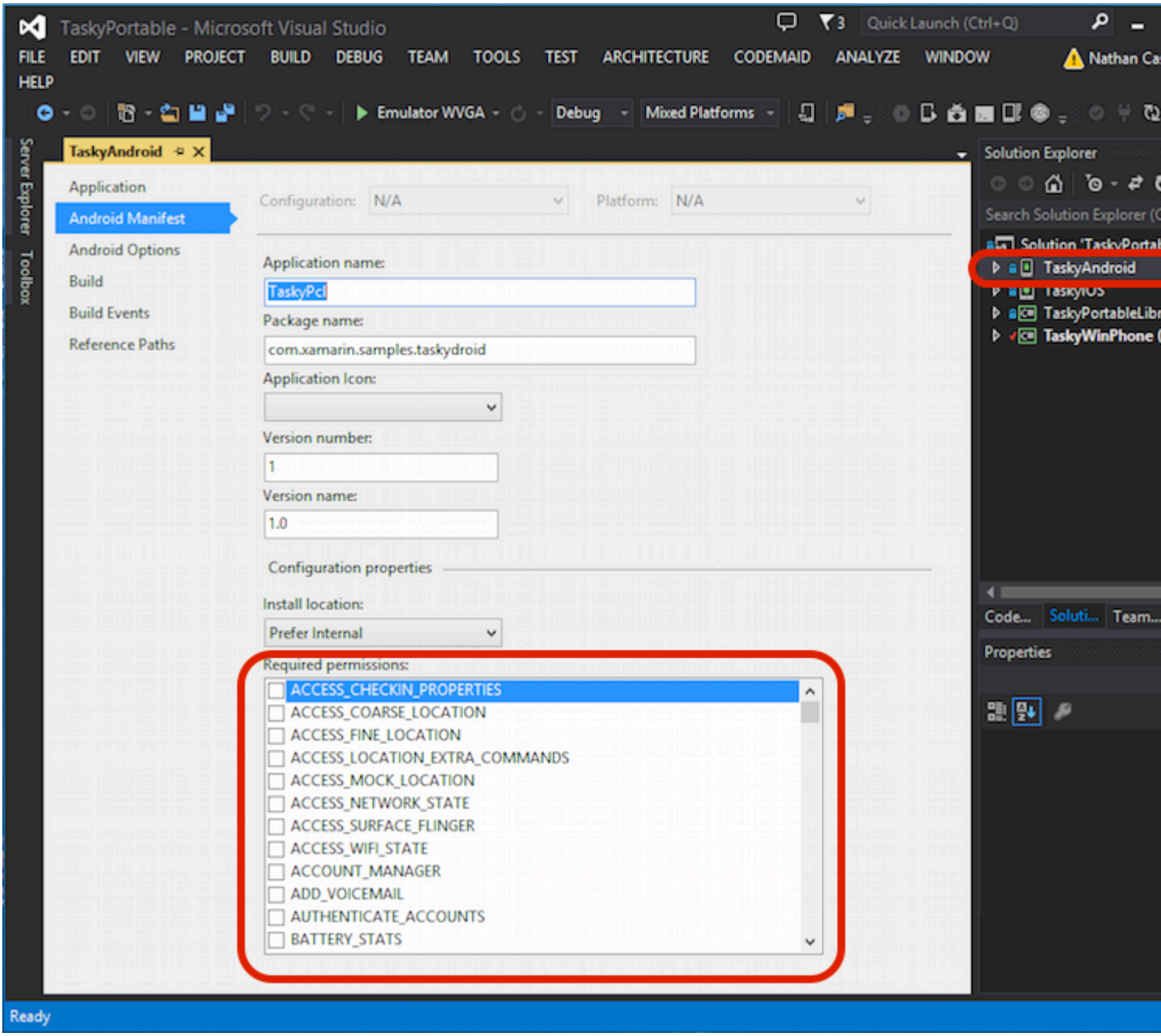
La documentación de Xamarin proporciona buena información aquí:

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

---

Manifiesto de Android

Primero, en Visual Studio, haga clic con el botón derecho en su proyecto Xamarin.Android en el Explorador de soluciones y seleccione Propiedades. Luego, vaya a la pestaña Manifiesto de Android, para ver esta pantalla:



A diferencia de Android Studio o Eclipse, no necesita la configuración del archivo AndroidManifest.xml al escribir; Xamarin y Visual Studio lo hacen por ti. Las actividades, BroadcastReceivers y Services se insertan en el manifiesto de Android declarando atributos específicos en sus clases .

En esta pantalla, las opciones son:

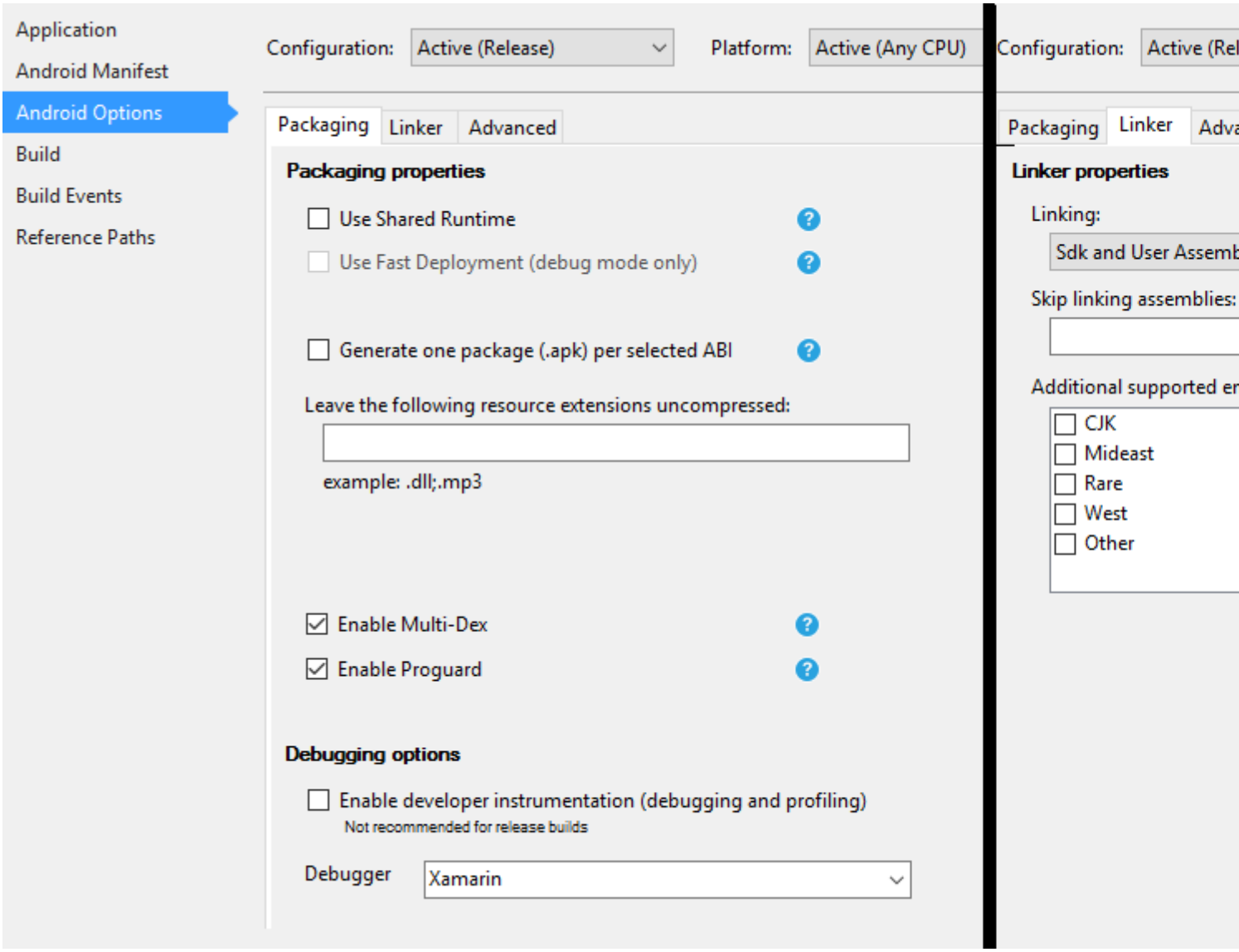
- **Nombre de la aplicación** : este es el nombre de la aplicación que estará visible para el usuario.
- **Nombre del paquete** : Este es el nombre del paquete. Debe ser único, lo que significa que no debe usar el mismo nombre de paquete de otras aplicaciones en Google Play Store.
- **Icono de la aplicación** : este es el icono que será visible para el usuario, equivalente al @ drawable / ic_launcher utilizado en los proyectos de Android Studio o Eclipse.
- **Número de versión** : Google Play utiliza el número de versión para el control de versiones. Cuando desee publicar un APK para una versión actualizada de su aplicación, debe agregar

1 a este número para cada nueva actualización.

- **Nombre de la versión** : este es el nombre de la versión que se mostrará al usuario.
- **Ubicación de instalación** : Esto determina dónde se instalará su APK, en el almacenamiento del dispositivo o en la tarjeta SD.
- **Permisos requeridos** : Aquí usted determina qué permisos son necesarios para su aplicación.

Opciones de Android

En la pantalla de abajo, puede configurar las opciones del compilador. El uso de las opciones correctas aquí puede reducir mucho el tamaño de su APK y también evitar errores.



- **Configuración** : **Activa (Release)** .
- **Plataforma** : **Activa (Cualquier CPU)** . Estos son necesarios para construir tu APK para Google Play Store. Si la configuración está configurada para depurar, Google Play no la aceptará.
- **Utilizar Tiempo de ejecución compartido** : **falso** . Si lo establece en verdadero, el APK utilizará Mono Runtime para ejecutarse. Mono Runtime se instala automáticamente cuando se realiza la depuración a través de USB, pero no en la versión Release APK. Si Mono

Runtime no está instalado en el dispositivo y esta opción se establece en verdadero en la versión Release APK, la aplicación se bloqueará.

- **Genere un paquete (.apk) por ABI seleccionado** : **falso** . Crea tu APK para tantas plataformas como sea posible, por razones de compatibilidad.
- **Habilite Multi-Dex** : **verdadero** , pero puede configurarlo en falso si su aplicación no es muy compleja (es decir, tiene menos de 65536 métodos, consulte aquí).
- **Habilitar Proguard** : **verdadero** . Esto habilita la herramienta Proguard que oculta el código Java en su aplicación. Tenga en cuenta que no se aplica al código .NET; Si desea ofuscar el código .NET, debe usar Dotfuscator . Más información sobre Proguard para Xamarin.Android se puede encontrar aquí .
- **Habilitar instrumentación de desarrollador (depuración y creación de perfiles)** : **falso** para Release APK.
- **Enlace** : **SDK y ensamblajes de usuarios** . Esto hará que el Xamarin Linker elimine todas las clases no utilizadas del SDK y su código, reduciendo el tamaño del APK.

Importante

Xamarin.Linker a veces puede eliminar clases que su código no parece utilizar, especialmente si están en el Core del proyecto (biblioteca PCL). Para evitar eso, puede establecer el enlace a "Solo ensamblajes SDK" o usar el atributo Preservar en sus clases, por ejemplo:

PreserveAttribute.cs

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

En una clase

```
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;
        }
    }
```

```
}
```

- **Arquitecturas soportadas** : **Seleccionar todas** , por razones de compatibilidad.

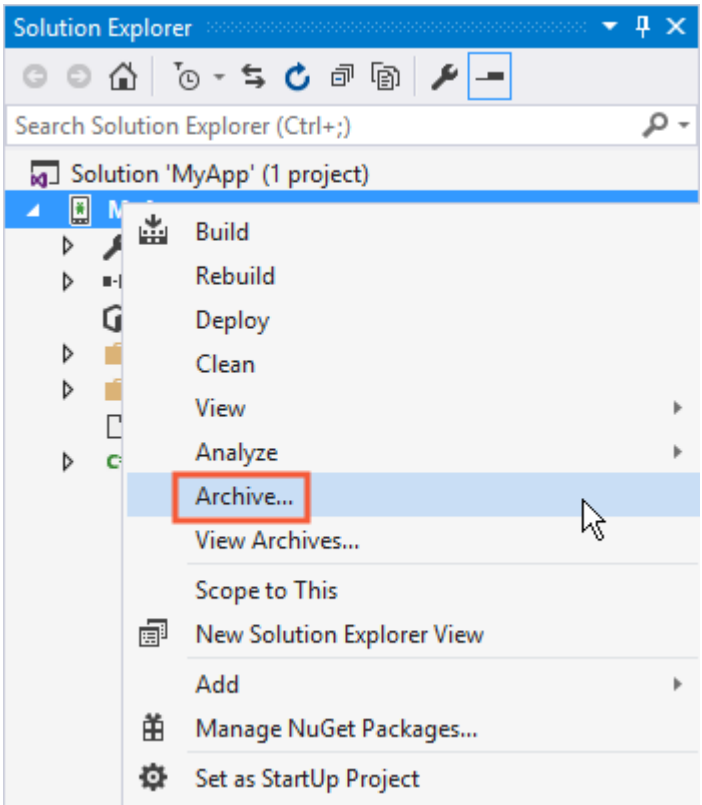
Después de configurar todo, Reconstruya el proyecto para asegurarse de que se compile correctamente.

Creando el APK para el modo Release

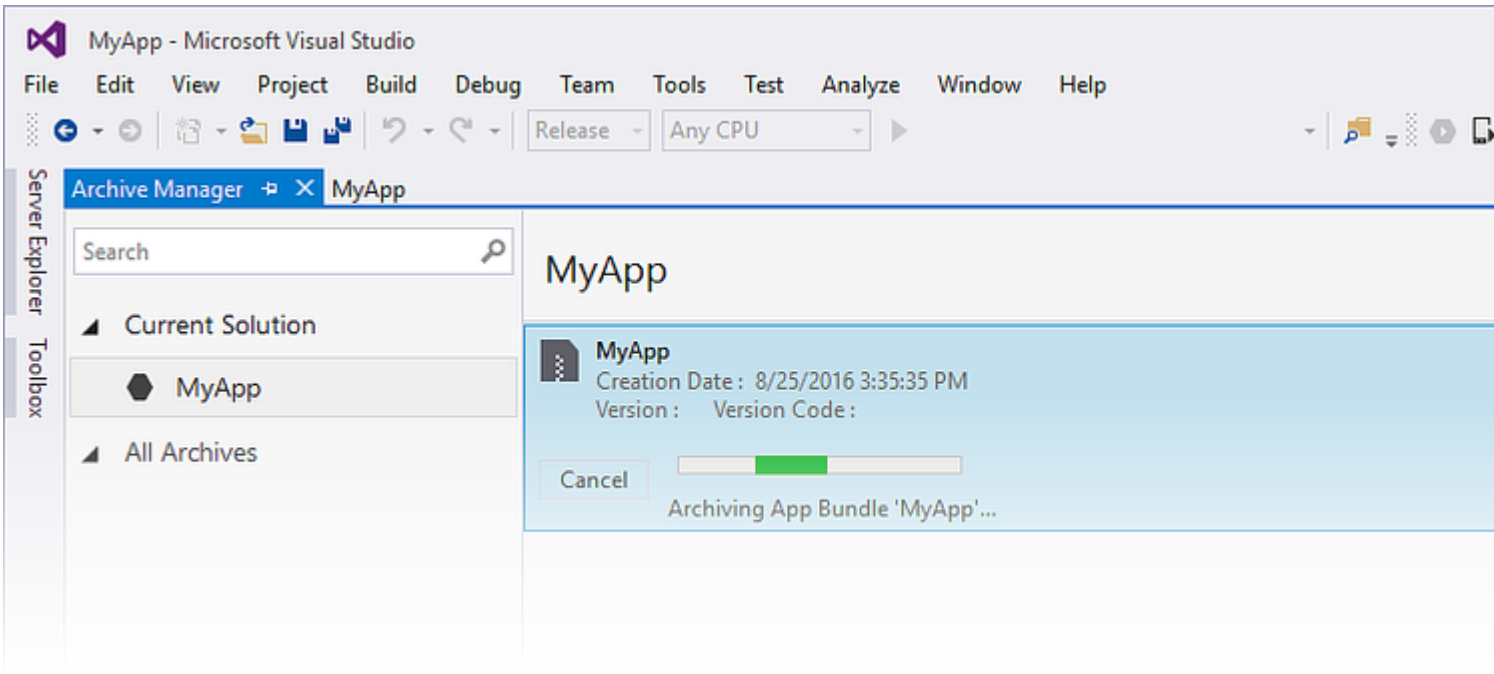
Has terminado de configurar tu proyecto de Android para su lanzamiento. El tutorial a continuación muestra cómo generar el APK en Visual Studio. Un tutorial completo de la documentación de Xamarin se puede encontrar aquí:

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

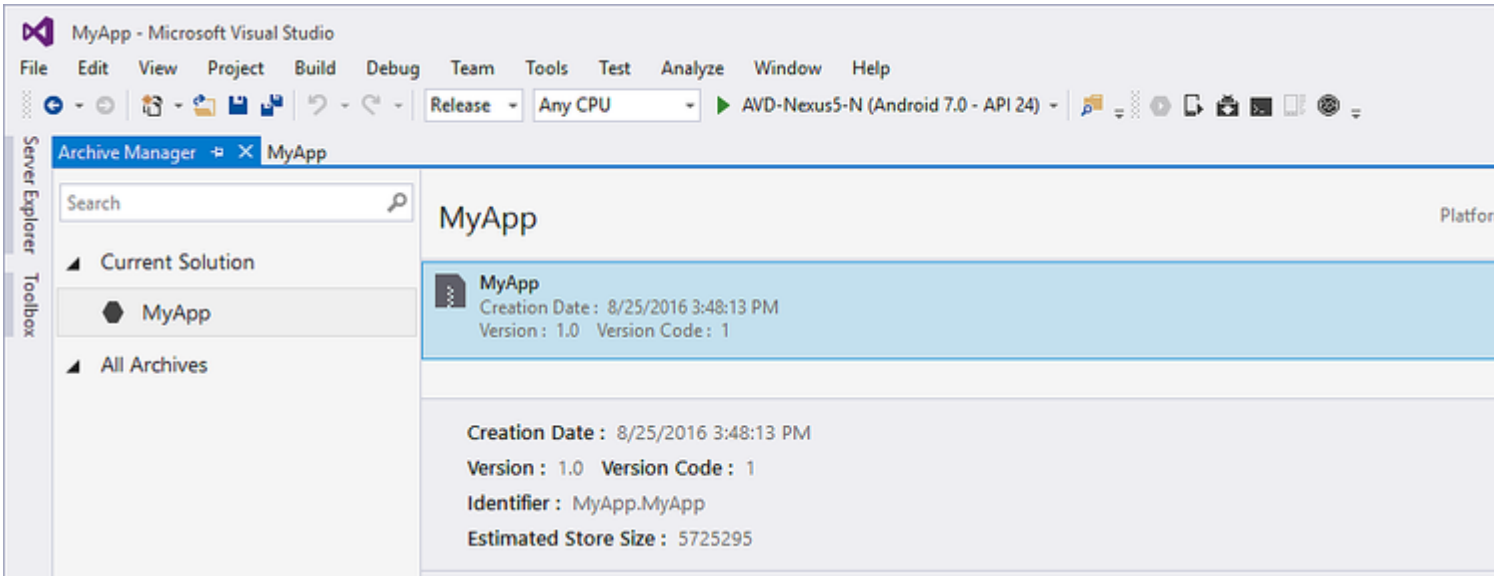
Para crear el archivo APK, haga clic derecho en el proyecto Xamarin.Android en el Explorador de soluciones y seleccione Archivar ...



Esto abrirá el administrador de archivos y comenzará a archivar el proyecto, preparándose para crear el archivo APK.

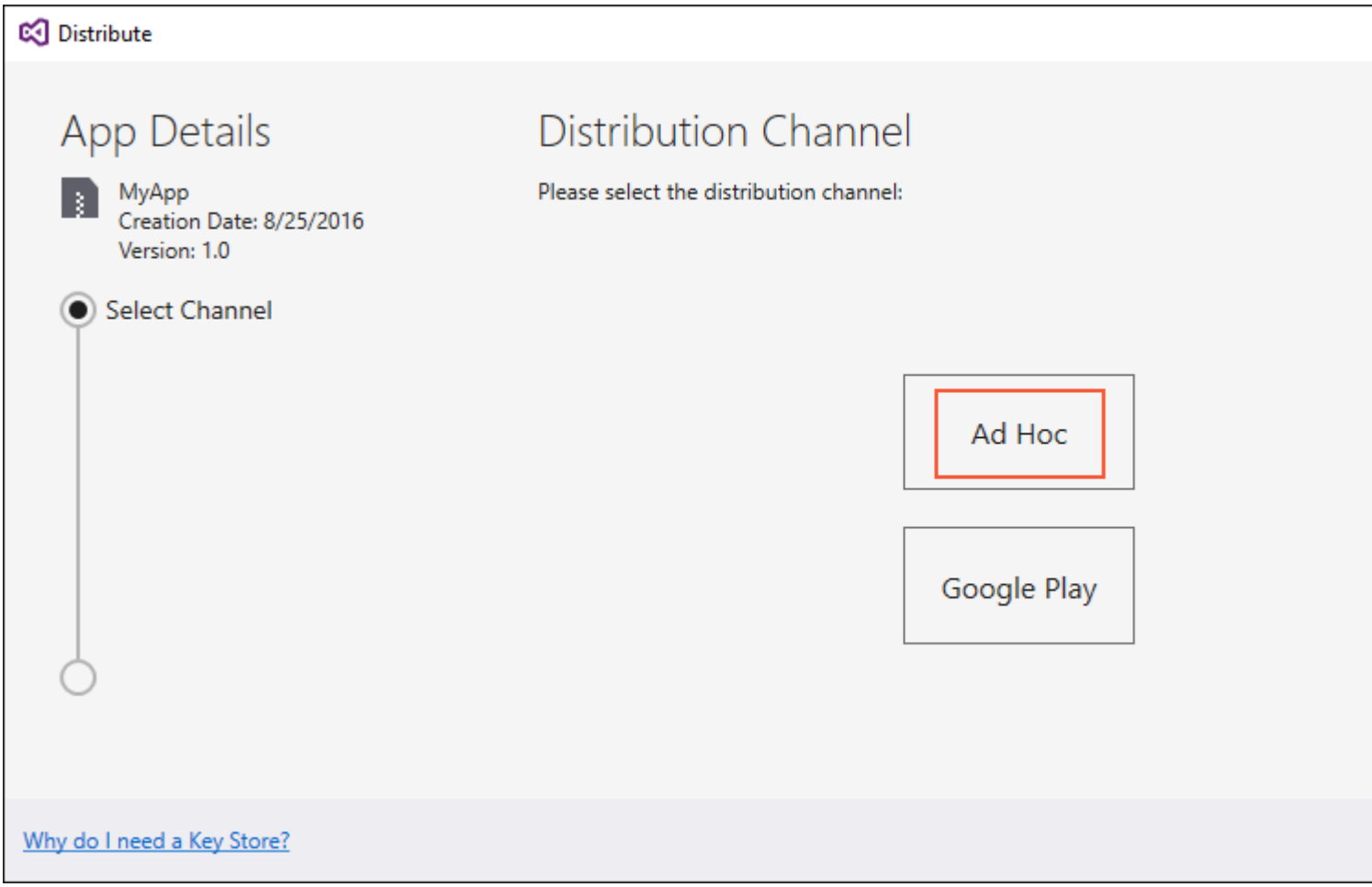


Cuando termine de archivar el proyecto, haga clic en Distribuir ... para continuar.

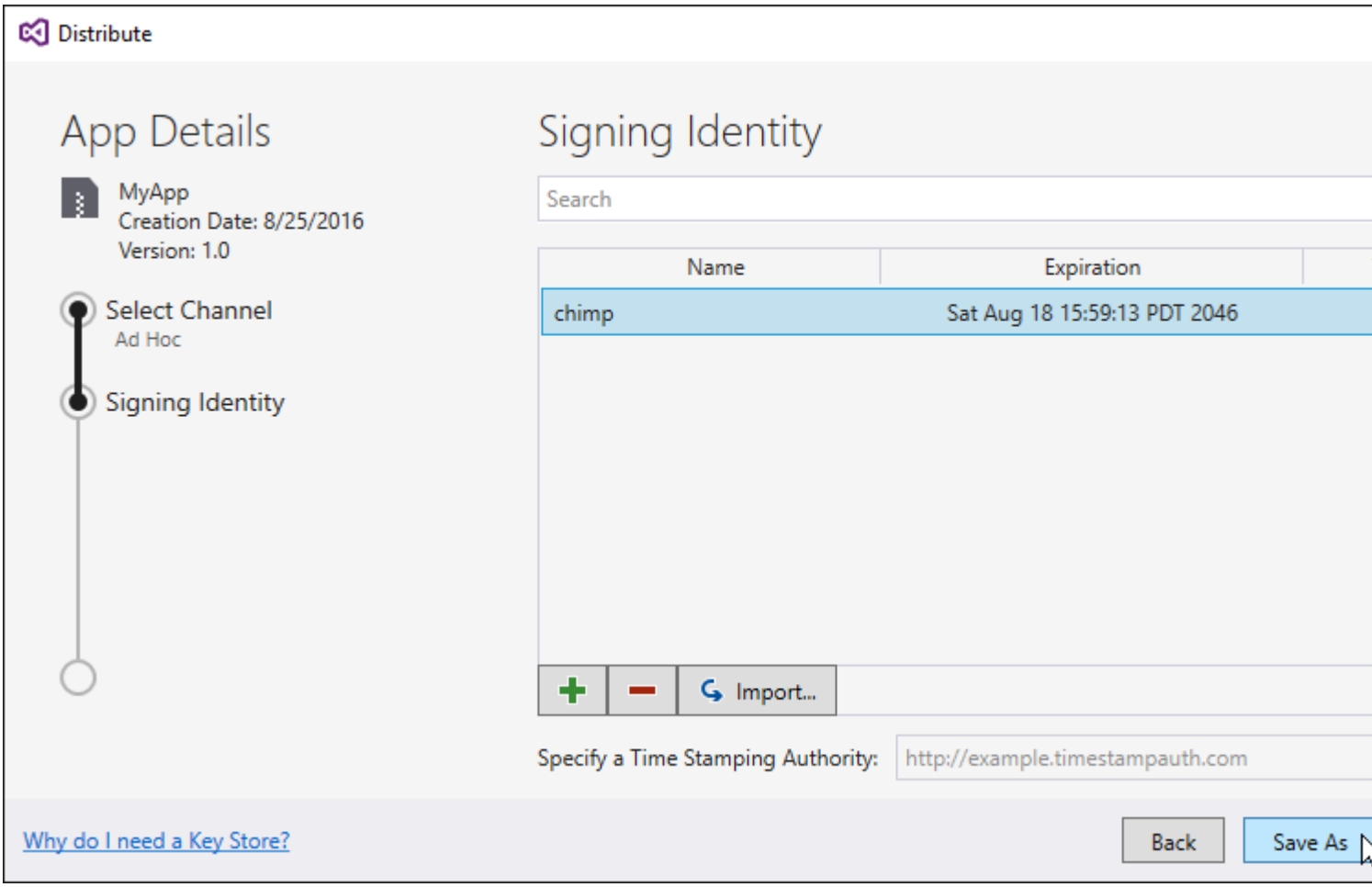


La pantalla de Distribución le presentará dos opciones: Ad-hoc y Google Play. El primero creará un APK y lo guardará en su computadora. El segundo publicará directamente la aplicación en Google Play.

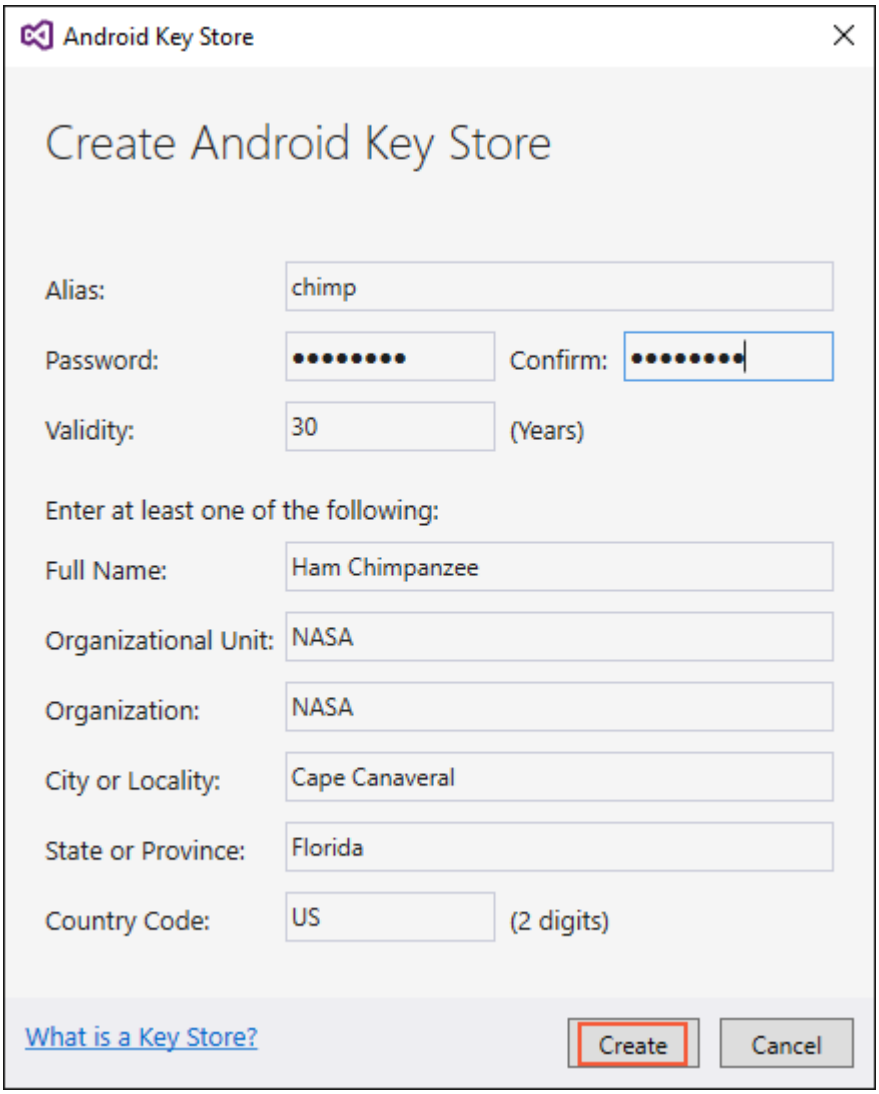
Se recomienda elegir el primero, para que pueda probar el APK en otros dispositivos si lo desea.



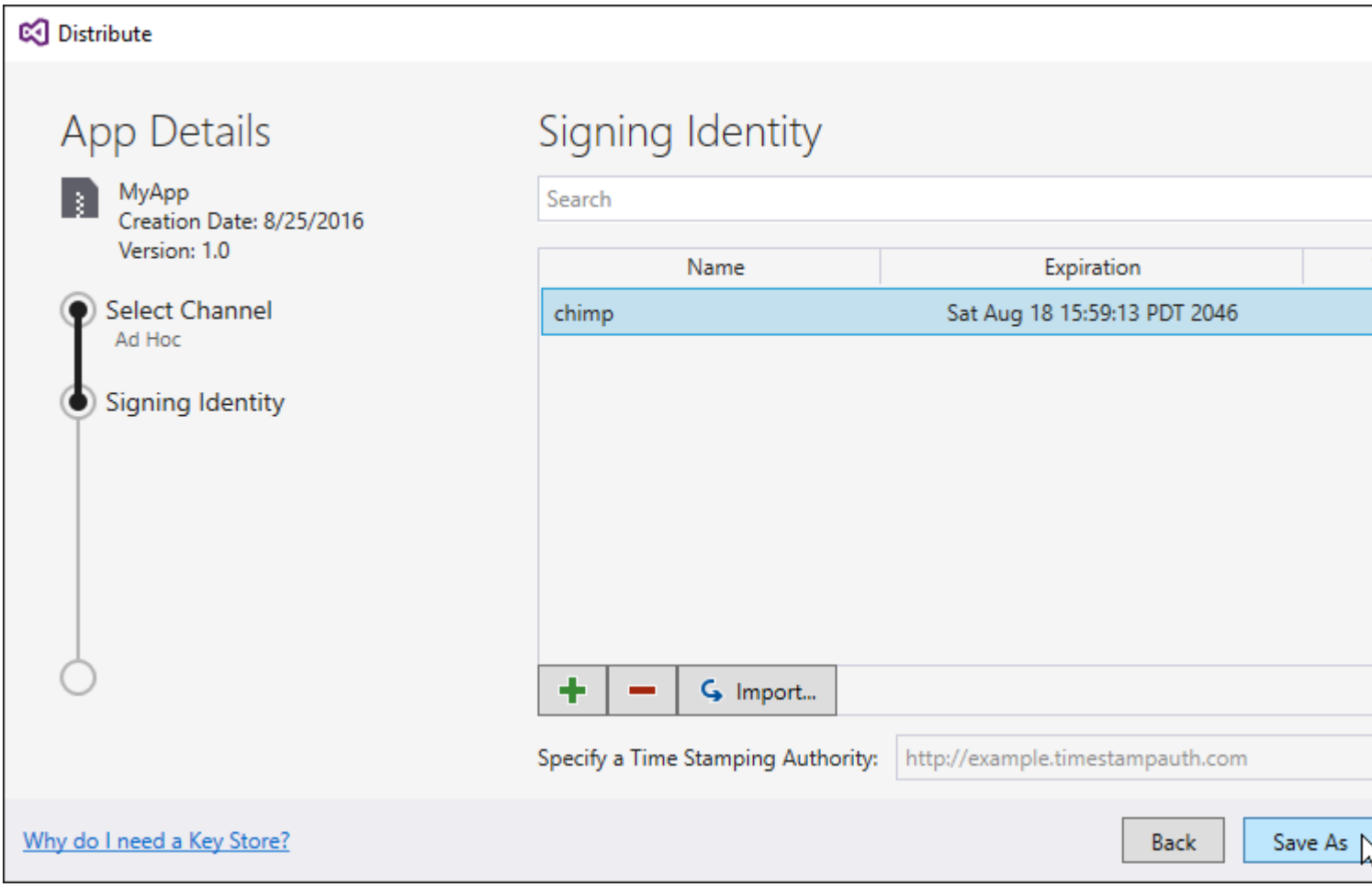
En la siguiente pantalla, se necesita un Android Key Store para firmar el APK. Si ya tiene uno, puede usarlo haciendo clic en Importar ...; Si no lo hace, puede crear un nuevo Android Key Store haciendo clic en +.

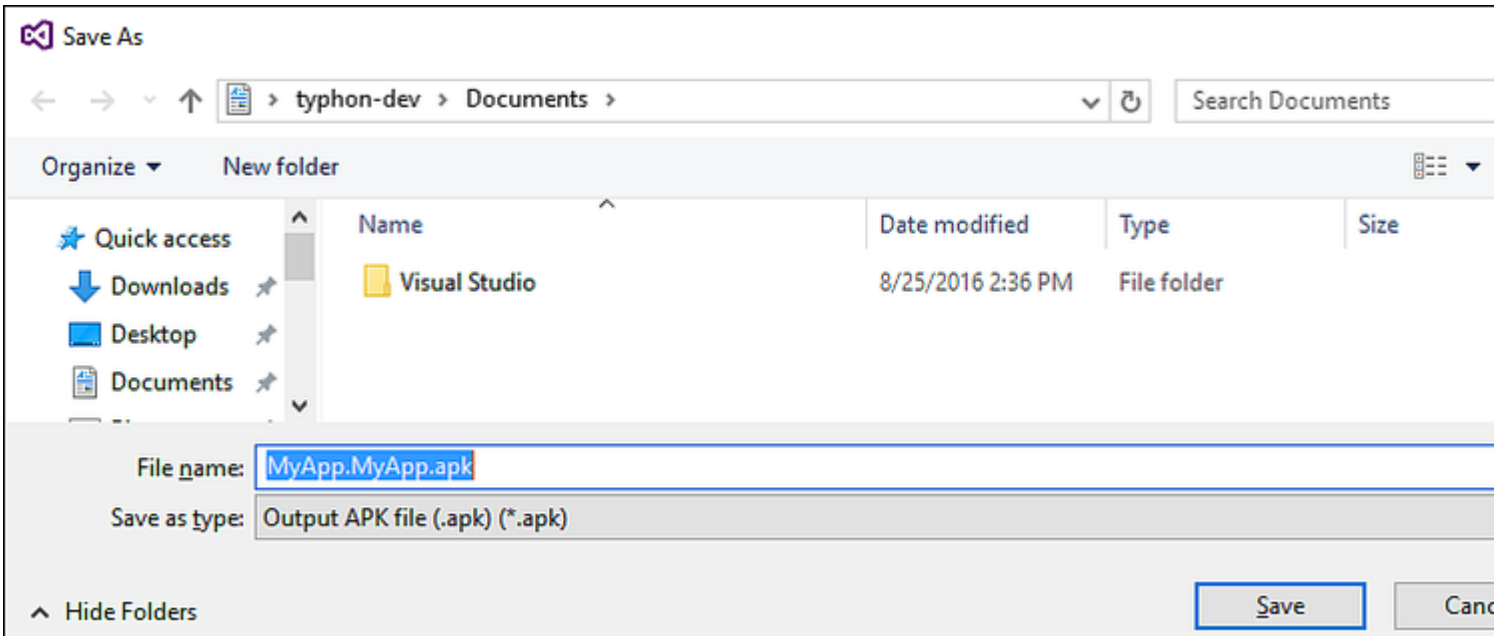


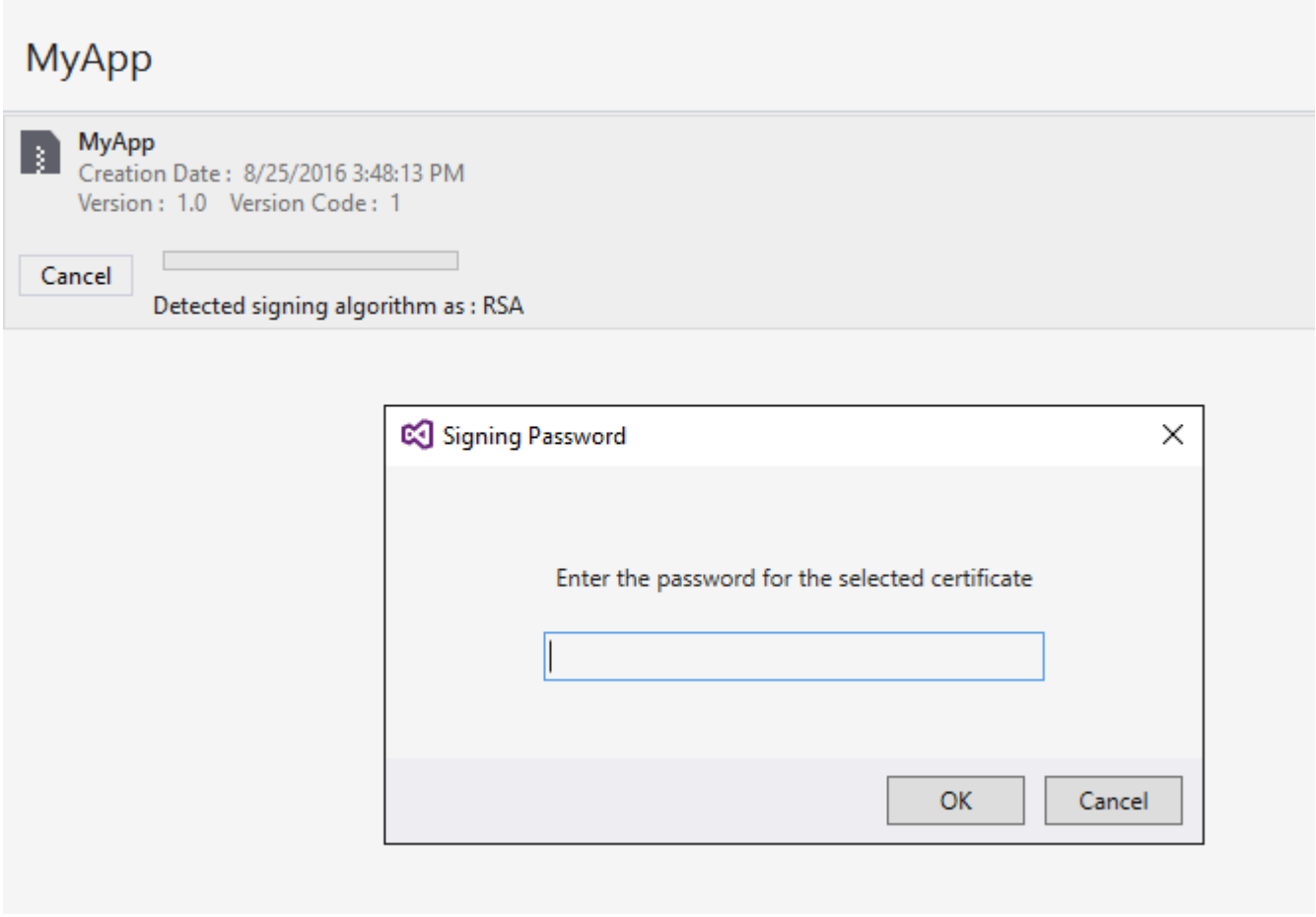
Creando una nueva pantalla de Android Key Store:



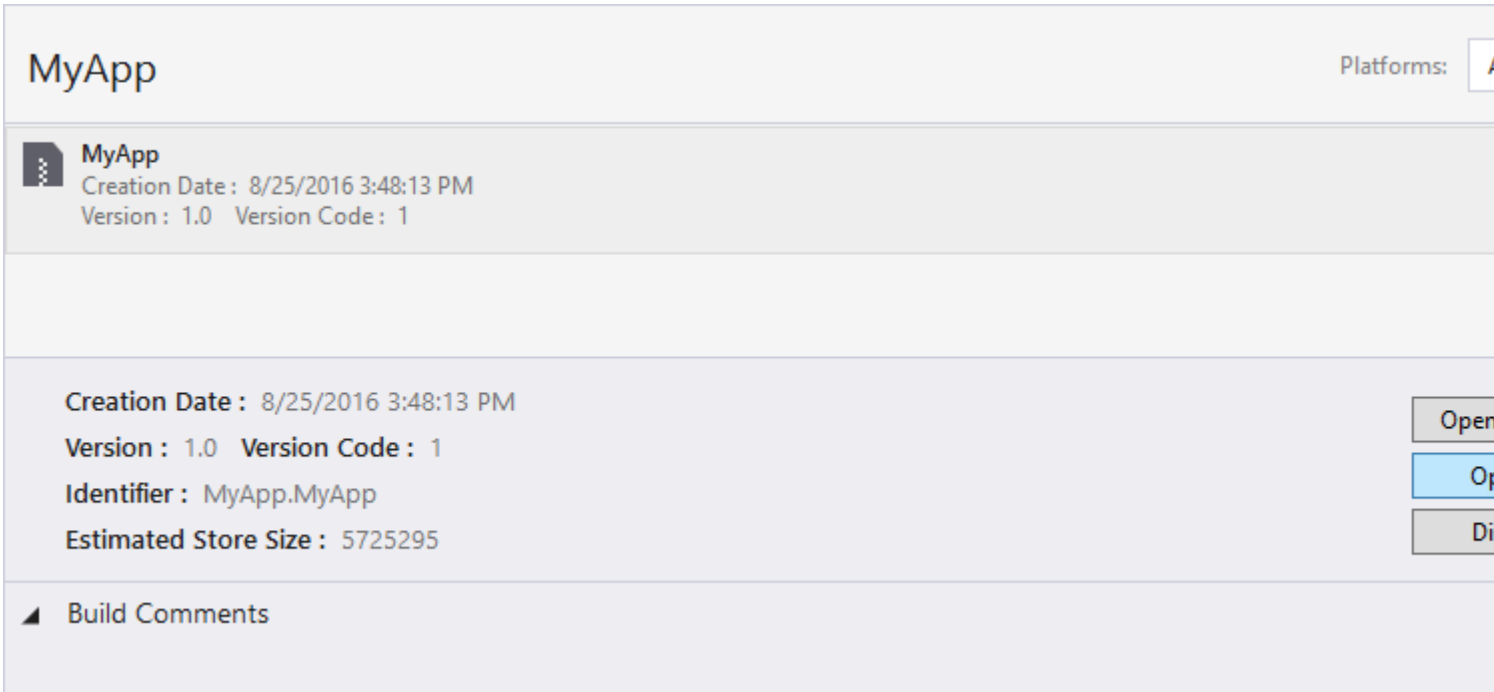
Para crear el APK, haga clic en Guardar como. Es posible que se le solicite que escriba la contraseña del almacén de claves.

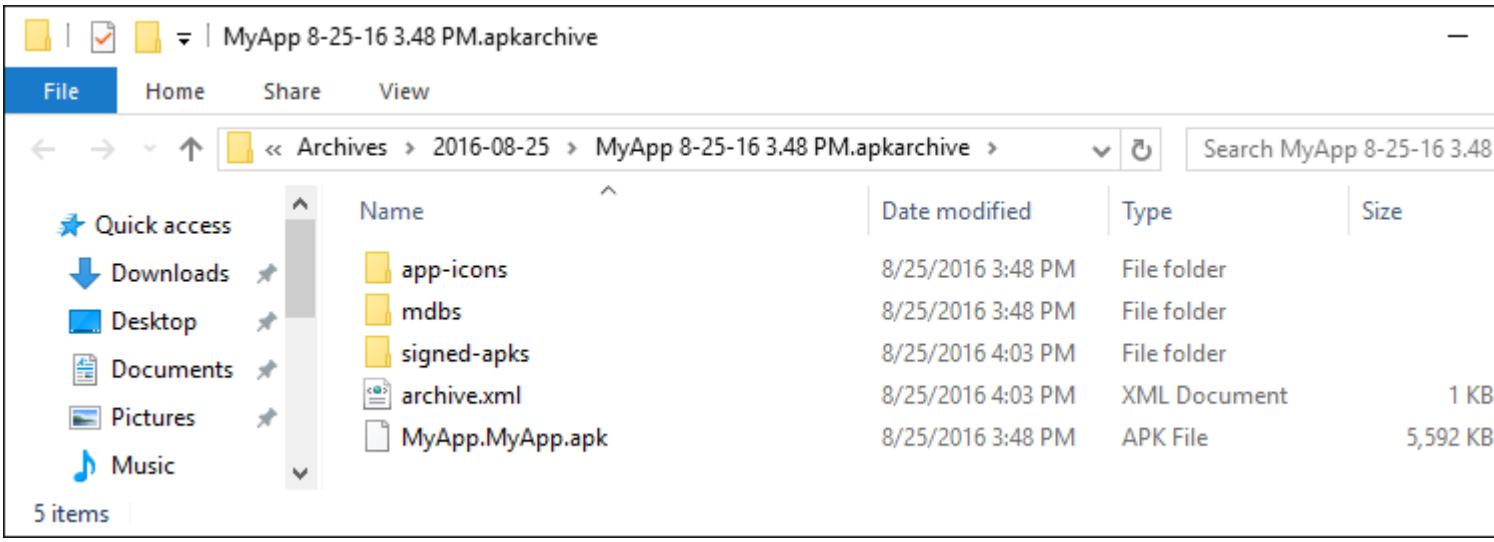

Distribute
App Details
MyApp
Creation Date: 8/25/2016
Version: 1.0
Select Channel
Ad Hoc
Signing Identity
Signing Identity
Search
Name
Expiration
chimp
Sat Aug 18 15:59:13 PDT 2046
Import...
Specify a Time Stamping Authority:
http://example.timestampauth.com
Why do I need a Key Store?
Back
Save As


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



Cuando se complete, puede hacer clic en Abrir carpeta en la pantalla Archivos para ver su archivo APK generado.





Habilitando MultiDex en tu Xamarin.Android APK

MultiDex es una biblioteca en el APK de Android que permite que la aplicación tenga más de 65,536 métodos.

Los archivos APK de Android tienen archivos ejecutables Dalvik (.dex) que contienen los códigos de bytes generados compilados a partir de su código Java. Cada archivo .dex puede contener hasta 65,536 métodos (2 ^ 16).

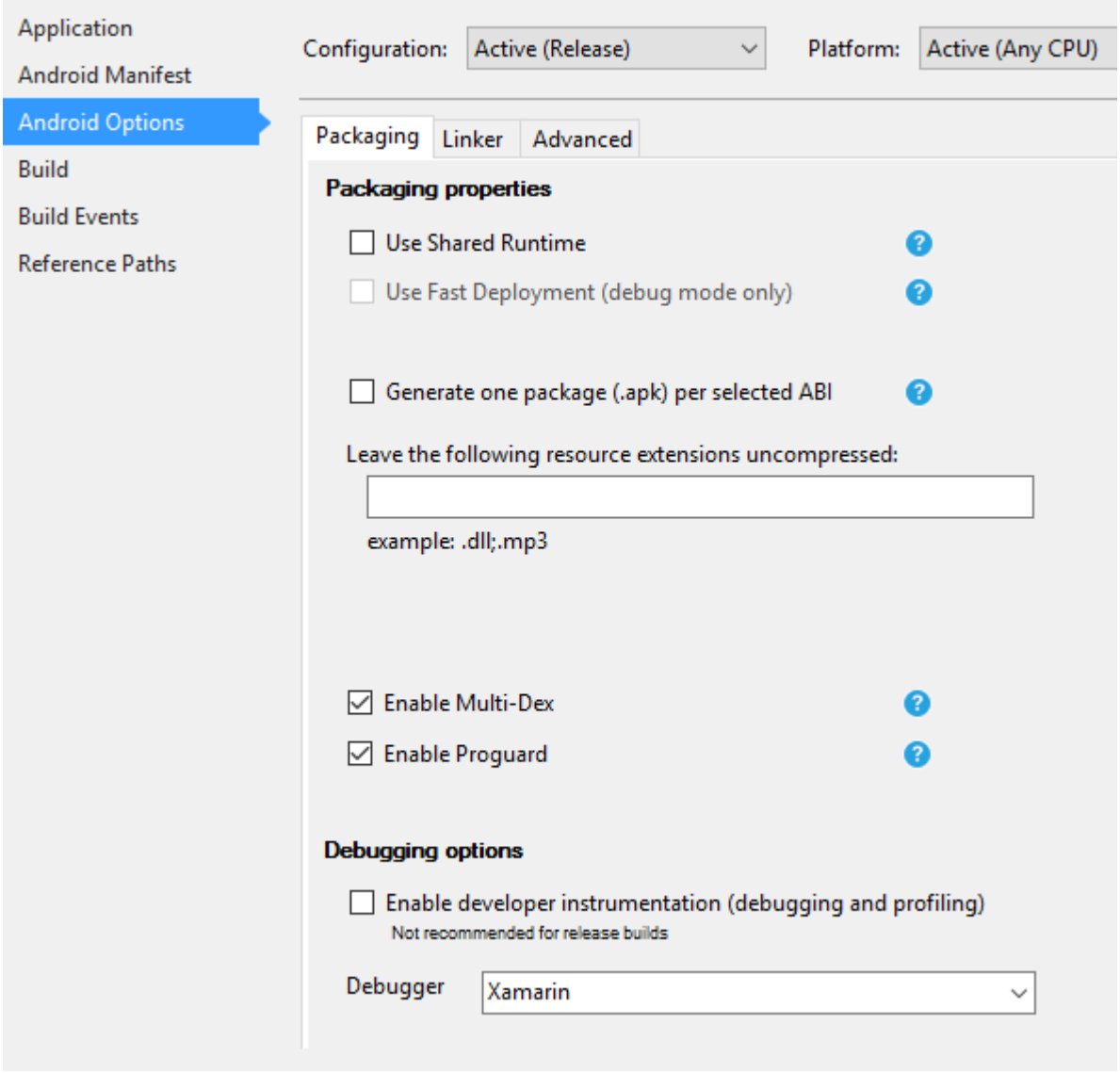
Las versiones de Android OS antes de Android 5.0 Lollipop (API 21) usan el tiempo de ejecución de Dalvik, que solo admite un archivo .dex por APK, limitando a 65,536 métodos por APK. A partir de Android 5.0, el sistema operativo Android usa el tiempo de ejecución ART, que puede admitir más de un archivo .dex por APK, evitando el límite.

Para superar el límite de métodos de 65k en las versiones de Android por debajo de API 21, los desarrolladores deben usar la biblioteca de soporte MultiDex. El MultiDex crea archivos adicionales classes.dex (classes2.dex, classes3.dex, ...) haciendo referencia a ellos en el archivo classes.dex. Cuando la aplicación comienza a cargarse, utiliza una clase MultiDexApplication para cargar los archivos .dex adicionales.

Si su aplicación de Android apunta a una versión mínima de SDK superior o igual a API 21 (Android 5.0 Lollipop), no es necesario usar la biblioteca MultiDex, porque el sistema operativo maneja de forma nativa los archivos .dex adicionales. Sin embargo, si por razones de compatibilidad el desarrollador desea admitir el sistema operativo Android más antiguo, entonces él / ella debe usar la biblioteca MultiDex.

Cómo usar MultiDex en tu aplicación Xamarin.Android

Primero, para habilitar MultiDex en su aplicación Xamarin.Android, vaya a Propiedades del proyecto -> Opciones de Android -> Empaque -> Habilite Multi-Dex, como en la pantalla de impresión a continuación:



Luego, debe crear una clase MultiDexApplication en su aplicación. En la raíz del proyecto, cree una nueva clase (en el Explorador de soluciones, haga clic con el botón derecho en el proyecto, Agregar .. -> Clase o Mayús + Alt + C). En el nuevo archivo de clase, copie el siguiente código, reemplazando la muestra del espacio de nombres con el nombre de su espacio de nombres del proyecto 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); }
        }
    }
}
```

Código fuente aquí.

Si está desarrollando en Visual Studio para Windows, también hay un error en las herramientas de compilación del SDK de Android que debe corregir para crear correctamente los archivos classes.dex al crear su proyecto.

Vaya a su carpeta SDK de Android, abra la carpeta de herramientas de construcción y habrá carpetas con los números de los compiladores del SDK de Android, como:

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

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

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

Dentro de cada una de esas carpetas, hay un archivo llamado **mainClassesDex.bat** , un script por lotes que se utiliza para crear los archivos classes.dex. Abra cada archivo mainClassesDex.bat con un editor de texto (Notepad o Notepad ++) y, en su script, busque y reemplace el bloque:

```
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 el bloque:

```
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
```

Fuente aquí.

Guarde cada mainClassesDex.bat en el editor de texto después de los cambios.

Después de los pasos anteriores, debería poder construir con éxito su aplicación Xamarin.Android con MultiDex.

Habilitando ProGuard en tu Xamarin.Android APK

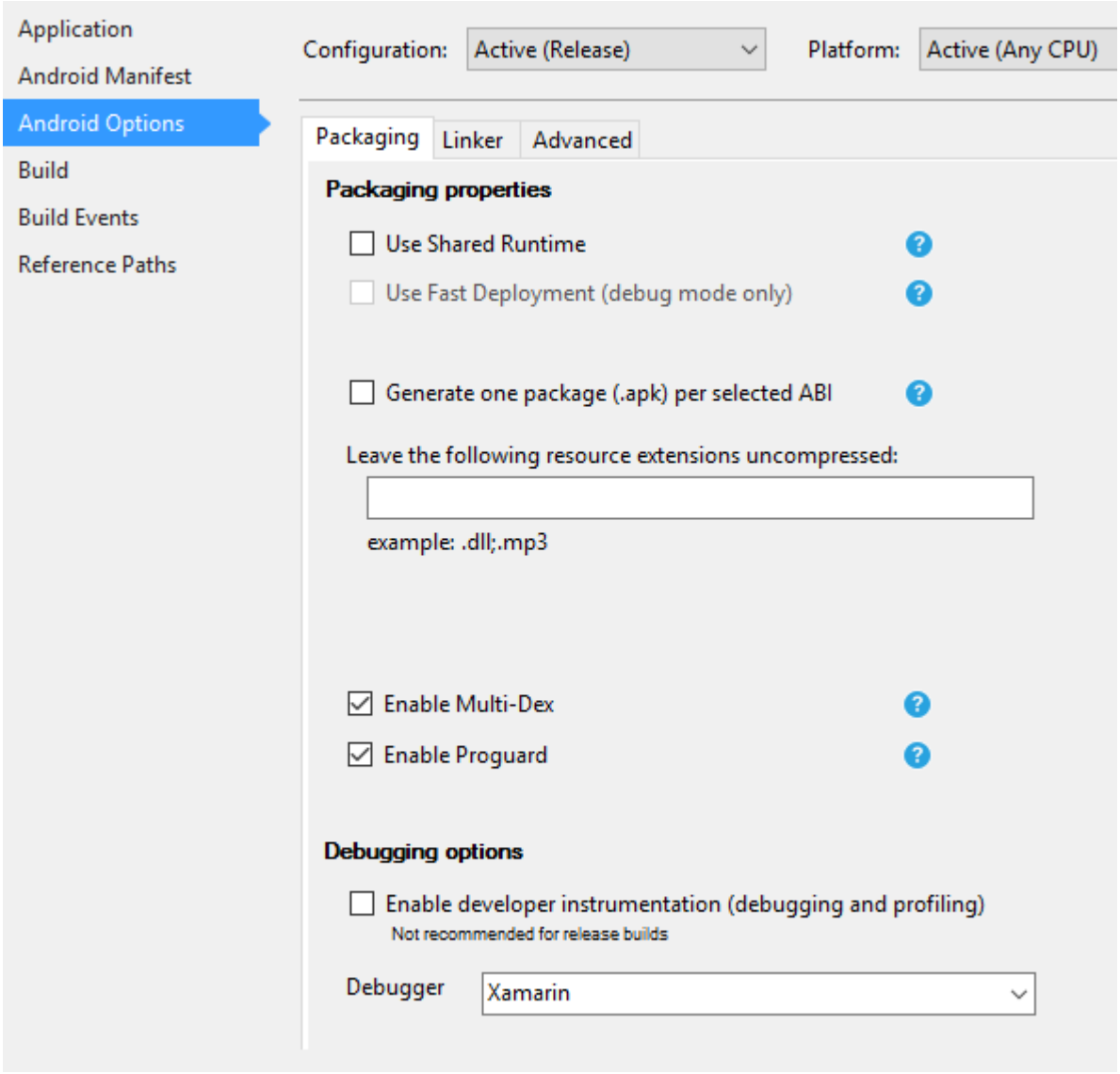
ProGuard es una herramienta que se utiliza en el proceso de construcción para optimizar y ofuscar el código Java de tu APK y también para eliminar las clases no utilizadas. El APK resultante cuando se usa ProGuard tendrá un tamaño más pequeño y será más difícil realizar ingeniería inversa (descompilación).

ProGuard se puede usar también en las aplicaciones de Xamarin.Android, y también reducirá el tamaño del archivo APK y ofuscará el código de Java. Sin embargo, tenga en cuenta que la ofuscación de ProGuard solo se aplica al código Java. Para ofuscar el código .NET, el

desarrollador debe usar Dotfuscator o herramientas similares.

Cómo usar ProGuard en tu aplicación Xamarin.Android

Primero, para habilitar ProGuard en su aplicación Xamarin.Android, vaya a Propiedades del proyecto -> Opciones de Android -> Empaque -> Habilite ProGuard, como en la pantalla de impresión a continuación:

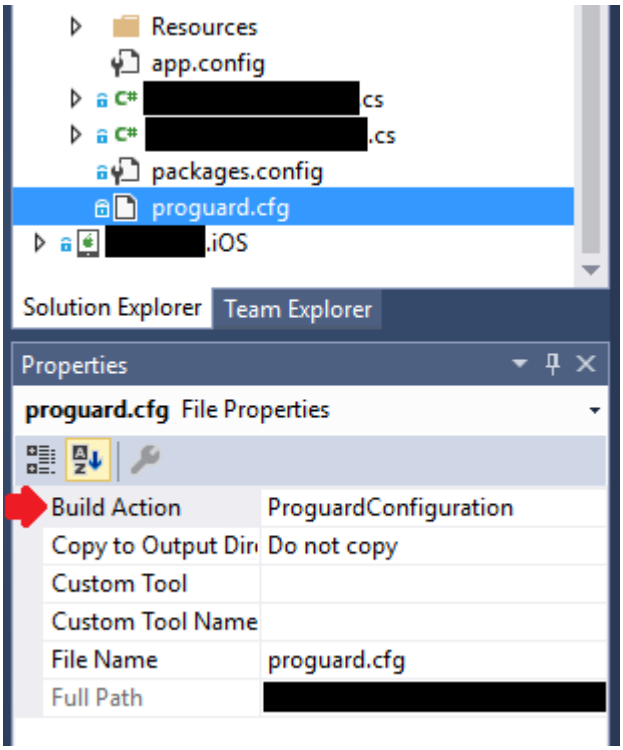


Esto permite ProGuard al construir su aplicación.

Xamarin.Android, por defecto, establece sus propias configuraciones para ProGuard, que se pueden encontrar dentro de las carpetas `obj/Debug/proguard` u `obj/Release/proguard` , en los archivos `proguard_project_primary.cfg` , `proguard_project_references.cfg` y `proguard_xamarin.cfg` . Los tres archivos se combinan como configuraciones para ProGuard y Xamarin los crea automáticamente al construirlos.

Si el desarrollador desea personalizar aún más las opciones de ProGuard, él / ella puede crear un archivo en la raíz del proyecto llamada `proguard.cfg` (otros nombres también son válidos, siempre

y cuando la extensión sea .cfg) y configurar su Acción de compilación en ProguardConfiguration, como en la imagen de abajo:



En el archivo, se pueden insertar opciones personalizadas de ProGuard, como `-dontwarn` , `-keep class` y otras .

Importante

A fecha de hoy (abril / 2017), el SDK de Android que generalmente se descarga tiene una versión antigua de ProGuard, que puede causar errores al crear la aplicación con Java 1.8. Al construir, la lista de errores muestra el siguiente mensaje:

```
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]
```

Fuente aquí.

Para solucionar este problema, debe descargar la versión más reciente de ProGuard (aquí) y copiar el contenido del archivo .zip en `android-sdk\tools\proguard\` . Eso actualizará ProGuard y el proceso de construcción debería ejecutarse sin problemas.

Después de eso, debería poder construir con éxito su aplicación Xamarin.Android con ProGuard.

Errores "misteriosos" relacionados con ProGuard y Linker

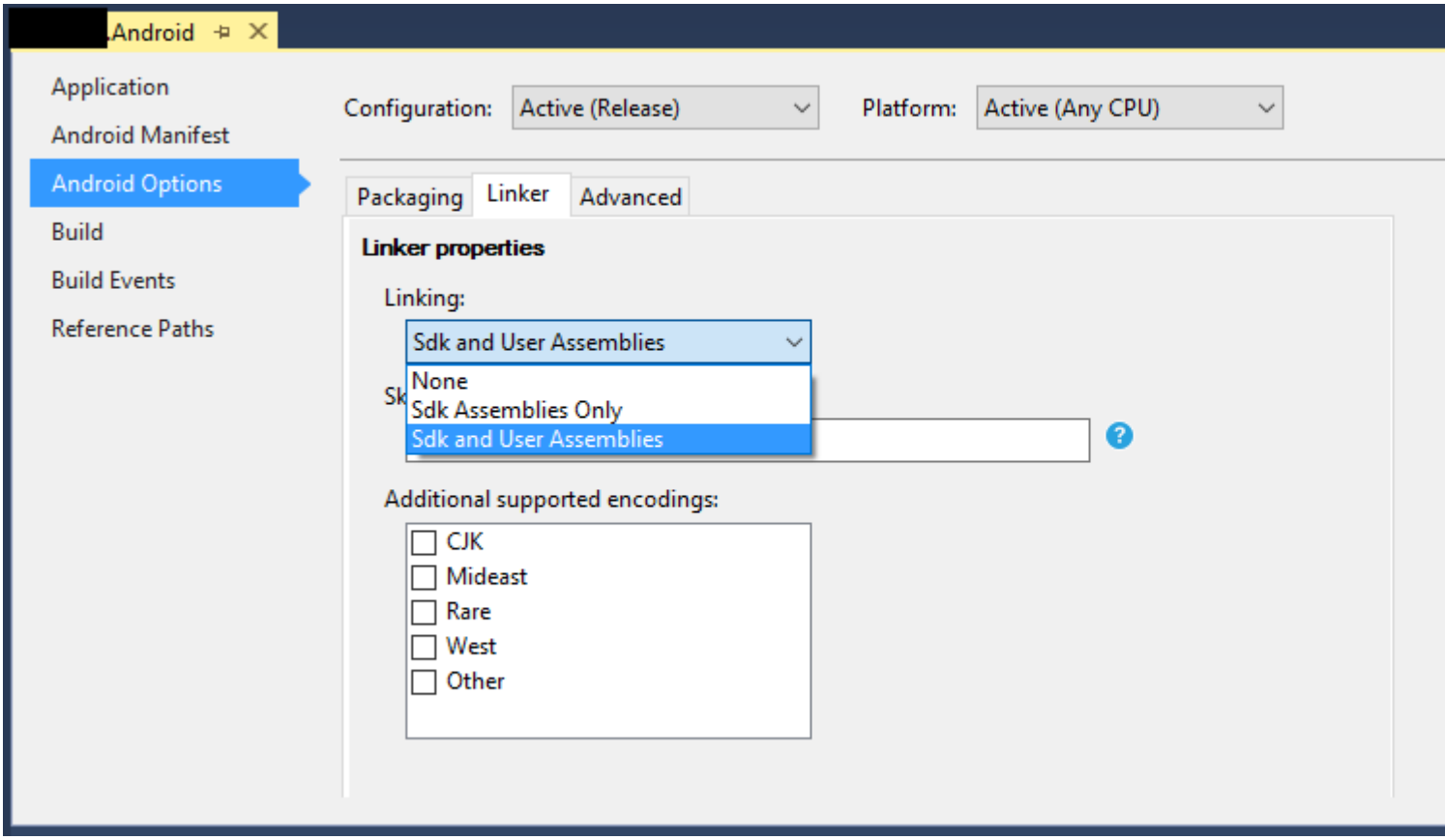
Hiciste una gran aplicación y la probaste en Debug, con buenos resultados. ¡Todo funcionaba bien!

Pero entonces, decidiste preparar tu aplicación para el lanzamiento. Configuró MultiDex, ProGuard y Linker, y luego, dejó de funcionar.

Este tutorial tiene como objetivo ayudarlo a encontrar problemas comunes relacionados con ProGuard y Linker que pueden causar errores misteriosos.

Entendiendo Xamarin.Linker

Xamarin.Linker es una herramienta en el proceso de construcción que elimina el código y las clases no utilizadas **de su código .NET (no el código Java)** . En las Propiedades de su proyecto -> Opciones de Android -> Vinculador, habrá un cuadro de selección Vincular con las opciones:



Ninguno : no se elimina ningún código.

Solo ensamblajes Sdk : esta opción hace que Xamarin.Linker compruebe el código no utilizado solo en las bibliotecas de Xamarin. **Esta opción es segura.**

Conjuntos de usuarios y SDK : esta opción hace que Xamarin.Linker compruebe el código no utilizado en las bibliotecas de Xamarin y en el código del proyecto (incluidos los PCL, los componentes de Xamarin y los paquetes de NuGet). **¡Esta opción no siempre es segura!**

Cuando se usa la opción Sdk and User Assemblies, Xamarin.Linker puede pensar que partes del código no se usan cuando en realidad se usan mucho. Esto puede hacer que algunas bibliotecas dejen de funcionar correctamente y causen errores en su aplicación.

Para hacer que el código Xamarin.Linker no elimine, hay 3 opciones:

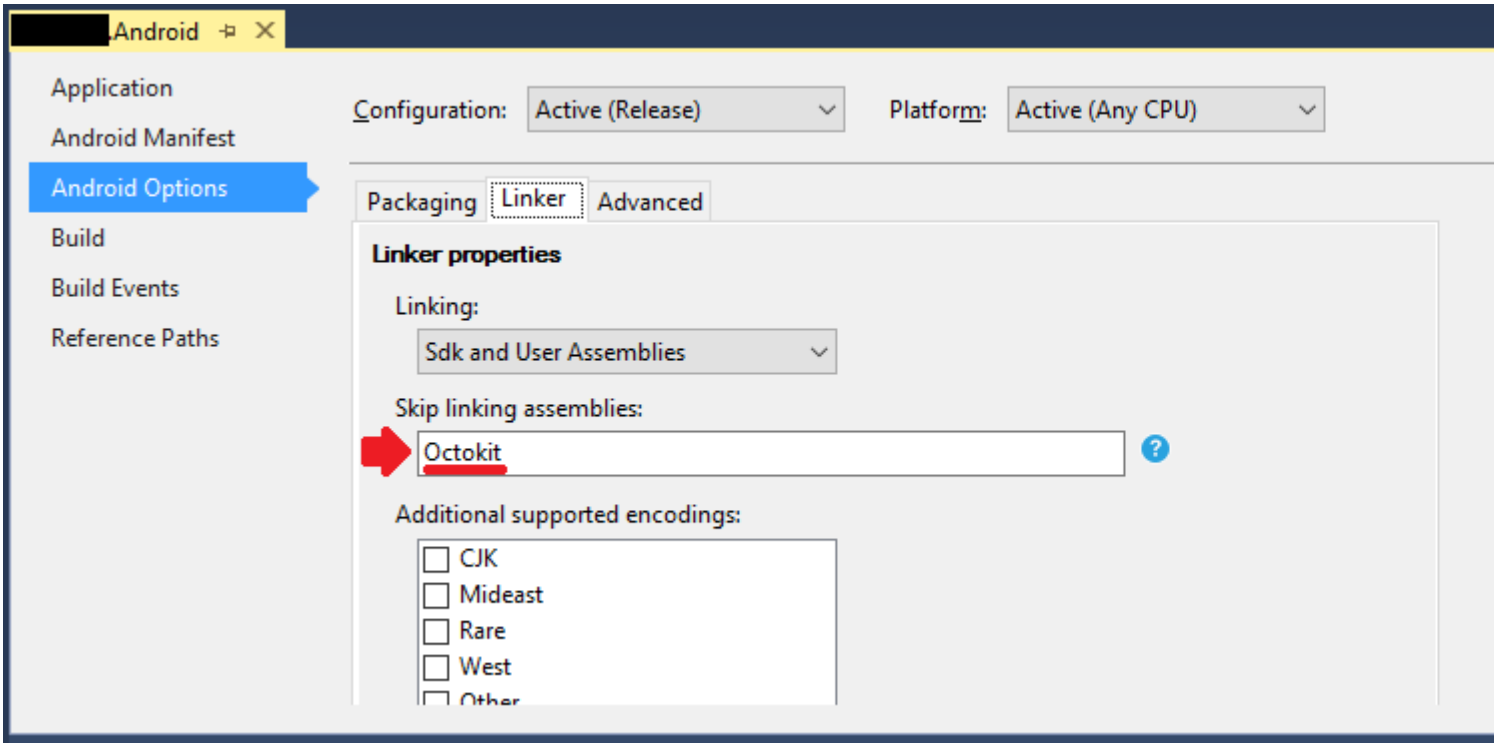
1. Estableciendo la opción de Enlace en Ninguno o Solo Ensambles SDK;
2. Saltar los conjuntos de enlace;
3. Usando el atributo Preservar.

Ejemplo para 2. Saltar ensamblajes de enlace:

En el ejemplo a continuación, el uso de Xamarin.Linker provocó que un paquete NuGet (Octokit) funcionara bien para dejar de funcionar, porque ya no podía conectarse 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 ---
```

Para hacer que la biblioteca vuelva a funcionar, fue necesario agregar el nombre de referencia del paquete en el campo Omitir conjuntos de enlaces, ubicado en proyecto -> Propiedades -> Opciones de Android -> Enlace, como se muestra en la siguiente imagen:



Después de eso, la biblioteca comenzó a trabajar sin ningún problema.

Ejemplo para 3. Usando el atributo Preservar:

Xamarin.Linker percibe como código no utilizado principalmente código de clases modelo en el núcleo de su proyecto.

Para hacer que la clase se mantenga durante el proceso de vinculación, puede usar el atributo Conservar.

Primero, cree en el núcleo de su proyecto una clase llamada **PreserveAttribute.cs** , inserte el siguiente código y reemplace el espacio de nombres con el espacio de nombres de su proyecto:

PreserveAttribute.cs:

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

En cada clase de modelo del núcleo de su proyecto, inserte el atributo Preservar como se muestra en el siguiente ejemplo:

País.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;
        }
    }
}
```

Después de eso, el proceso de vinculación ya no eliminará el código preservado.

Entendiendo ProGuard

ProGuard es una herramienta en el proceso de construcción que elimina el código y las clases no utilizadas **de su código Java** . También confunde y optimiza el código.

Sin embargo, ProGuard a veces puede eliminar el código que percibe como no utilizado, cuando no lo está. Para evitar eso, el desarrollador debe depurar la aplicación (en Android Device Monitor y en Visual Studio Debug) y detectar qué clase se eliminó, para luego configurar el archivo de configuración ProGuard para mantener la clase.

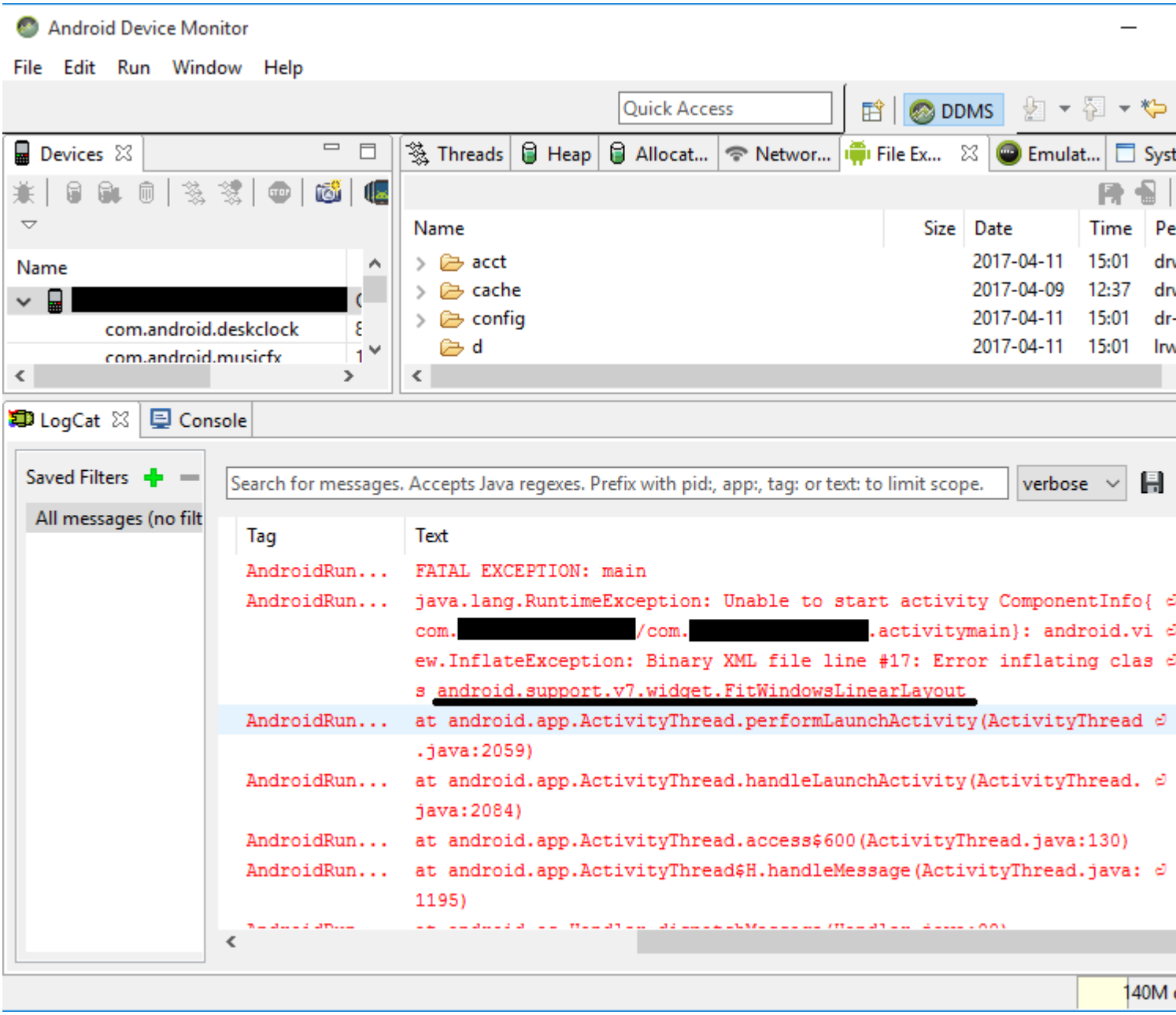
Ejemplo

En el siguiente ejemplo, ProGuard eliminó dos clases (Android.Support.V7.Widget.FitWindowsLinearLayout y Android.Support.Design.Widget.AppBarLayout) utilizadas en los archivos de diseño AXML, pero que se percibieron como no utilizadas en el código. La eliminación causó la excepción ClassNotFoundException en el código Java al representar el diseño de la actividad:

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 muestra un error al crear el diseño en SetContentView:



Para corregir este error, fue necesario agregar las siguientes líneas al archivo de configuración ProGuard del proyecto:

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

Después de eso, no se mostraron más errores al crear el diseño.

Advertencias ProGuard

ProGuard a veces muestra advertencias en la Lista de errores después de construir su proyecto. Aunque plantean la pregunta de si su aplicación está bien o no, no todas sus advertencias indican problemas, especialmente si su aplicación se construye con éxito.

Un ejemplo de esto es cuando se usa la biblioteca Picasso : cuando se usa ProGuard, esto puede mostrar advertencias como `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]) (...)` , pero la aplicación se compila y la biblioteca funciona sin problemas.

Lea Publicando tu Xamarin.Android APK en línea: https://riptutorial.com/es/xamarin-android/topic/9601/publicando-tu-xamarin-android-apk

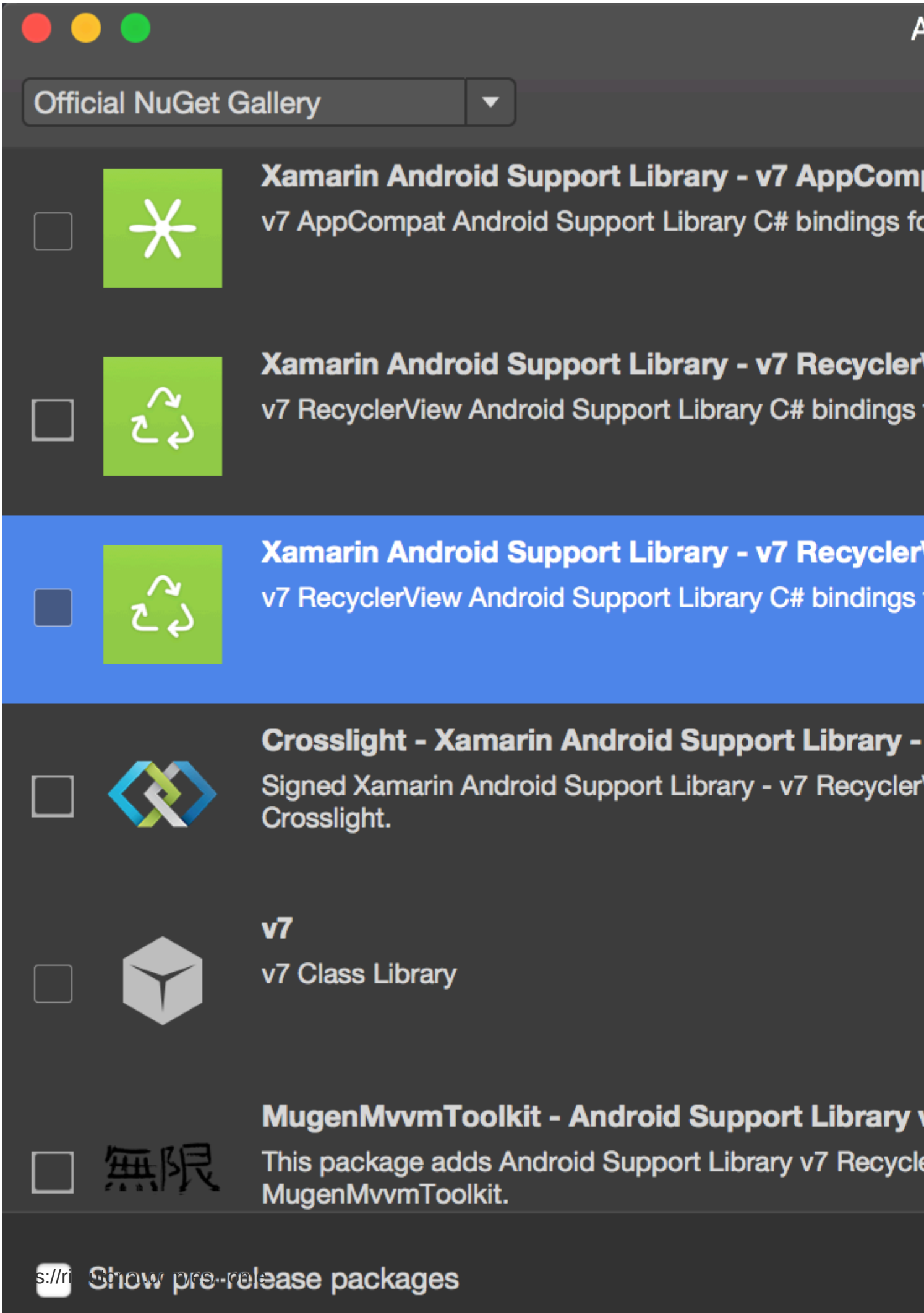
Capítulo 10: RecyclerView

Examples

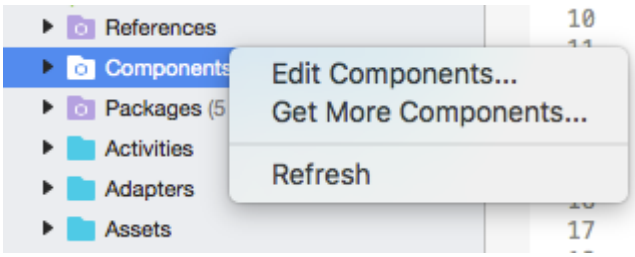
Conceptos básicos de RecyclerView

Este es un ejemplo del uso de `Android Support Library V7 RecyclerView` . Las bibliotecas de soporte generalmente se recomiendan porque proporcionan versiones de nuevas características compatibles con versiones anteriores, proporcionan elementos útiles de IU que no están incluidos en el marco y proporcionan una gama de utilidades en las que pueden basarse las aplicaciones.

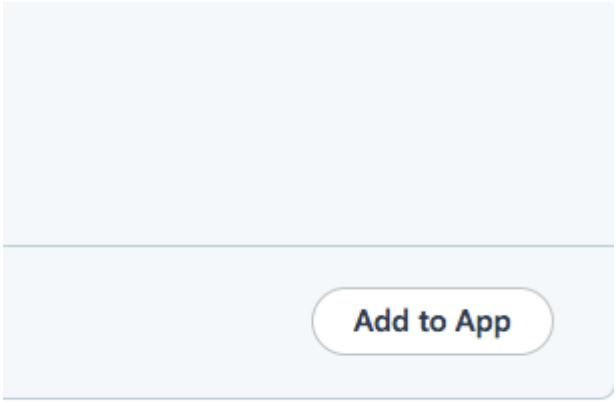
Para obtener el `RecyclerView` , instalaremos los paquetes necesarios de Nuget. En primer lugar, vamos a buscar `v7 recyclerview` . Desplácese hacia abajo hasta que veamos `Xamarin Android Support Library - v7 RecyclerView` . Selecciónelo y haga clic en **Agregar paquete** .


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

está disponible como un componente de Xamarin. Para agregar el componente, haga clic con el botón derecho en `Components` dentro del proyecto de Android en el Explorador de soluciones y haga clic en `Get More Components` .



Dentro de la ventana del Almacén de componentes que aparece, busque RecyclerView. En la lista de búsqueda, seleccione `Android Support Library V7 RecyclerView` . Luego haga clic en `Add to App` . El componente se agrega al proyecto.



El siguiente paso es agregar RecyclerView a una página. Dentro del `axml` (diseño), podemos agregar `RecyclerView` como se muestra a continuación.

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

RecyclerView requiere que se configuren al menos dos clases auxiliares para la implementación estándar básica, a saber: `Adapter` y `ViewHolder` . `Adapter` infla los diseños de elementos y vincula los datos a las vistas que se muestran dentro de RecyclerView. ViewHolder busca y almacena referencias de vista. El titular de la vista también ayuda a detectar clics de vista de elemento.

Aquí hay un ejemplo básico de clase de adaptador

```
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;
        }
    }
}
```

En el método `OnCreateViewHolder` primero inflamos una Vista y creamos una instancia de la clase ViewHolder. Esta instancia tiene que ser devuelta. Este método es invocado por el adaptador cuando requiere una nueva instancia de ViewHolder. Este método no será invocado para cada celda individual. Una vez que RecyclerView tenga suficientes celdas para llenar la Vista, reutilizará las celdas antiguas que se desplazan fuera de la Vista para obtener más celdas.

El `OnBindViewHolder` invoca la devolución de llamada de `OnBindViewHolder` para mostrar los datos en la posición especificada. Este método debe actualizar el contenido del itemView para reflejar el item en la posición dada.

Como la celda solo contiene un solo `TextView` , podemos tener un ViewHolder simple como se muestra a continuación.

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


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

El siguiente paso es conectar las cosas en la `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 clase LayoutManager es responsable de medir y posicionar las vistas de elementos dentro de RecyclerView, así como de determinar la política de cuándo reciclar las vistas de elementos que ya no son visibles para el usuario. Antes de `RecyclerView` , tuvimos que usar `ListView` para organizar las celdas como en una lista de desplazamiento vertical y `GridView` para mostrar los elementos en una cuadrícula desplazable de dos dimensiones. Pero ahora podemos lograr ambos con RecyclerView configurando un LayoutManger diferente. `LinearLayoutManager` organiza las celdas como en un ListView y `GridLayoutManager` organiza las celdas en forma de cuadrícula.

RecyclerView con eventos Click

Este ejemplo muestra cómo configurar Click EventHandlers en un Xamarin.Android RecyclerView.

En Android Java , la forma de configurar un oyente para un clic es utilizar un onClickListener para la vista en la que se hará clic, como este:

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

Sin embargo, en Xamarin.Android, la forma de configurar un oyente para un evento Click es mediante la **adición de** un EventHandler, de las siguientes maneras:

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
}
```

Tenga en cuenta que **el EventHandler se agrega, no se establece.** Si el Click EventHandler se agrega dentro de un método GetView desde un adaptador GridView / ListView, o un método OnBindViewHolder desde un RecyclerView.Adapter, cada vez que se crea la vista del elemento, se agregará un nuevo EventHandler. Después de desplazarse varias veces, se agregarán varios EventHandlers y, cuando se haga clic en la vista, se dispararán todos.

Para evitar este problema, los EventHandlers deben darse de baja y suscribirse posteriormente en los métodos GetView o OnBindViewHolder. Además, deben usar el número **3.** para configurar EventHandler, de lo contrario no será posible cancelar la suscripción de EventHandlers.

A continuación se muestra un ejemplo de un RecyclerView.Adapter con eventos 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);
    }

}
```

Lea RecyclerView en línea: https://riptutorial.com/es/xamarin-android/topic/3452/recyclerview

Capítulo 11: Tostadas

Examples

Mensaje básico de tostadas

Primero, cree una instancia de un objeto Toast con uno de los métodos `MakeText()` . Este método toma tres parámetros: el `Context` la aplicación, el mensaje de texto y la duración del brindis. Devuelve un objeto Toast correctamente inicializado. Puede mostrar la notificación de brindis con `Show()` , como se muestra en el siguiente ejemplo:

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

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

Este ejemplo muestra todo lo que necesita para la mayoría de las notificaciones de brindis. Rara vez debería necesitar algo más. Sin embargo, puede querer colocar el brindis de manera diferente o incluso usar su propio diseño en lugar de un simple mensaje de texto. Las siguientes secciones describen cómo puedes hacer estas cosas.

También puede encadenar sus métodos, llamar como una sola línea y evitar aferrarse al objeto Toast, como esto:

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

Para obtener más información, consulte la documentación más completa de Android sobre el tema.

Mensajes de color tostado

A veces, queremos dar información adicional a nuestro usuario con colores (por ejemplo, rojo significa que algo anda mal). Podemos cambiar el color de fondo del mensaje de tostada al configurar un filtro de color en la vista que nos brinda nuestra tostada (aquí 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();
```

Y también podemos cambiar el color del texto si el fondo es claro u oscuro:

```
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
```

Cambiar la posición de la tostada

Podemos cambiar nuestra tostada utilizando el método SetGravity. Este método toma tres parámetros: primero es la gravedad de la tostada en la pantalla y otros dos establecen la desviación de la tostada desde la posición inicial (que se establece con el primer parámetro):

```
//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();
```

Lea Tostadas en línea: https://riptutorial.com/es/xamarin-android/topic/3550/tostadas

Capítulo 12: Xamarin.Android - Cómo crear una barra de herramientas

Observaciones

Querido equipo,

Creo que es bueno mencionar sobre la documentación oficial de Android donde el control de la barra de herramientas se explica en detalle:

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

También hay contenido interesado en la biblioteca de Android.Support.v7 utilizada en el ejemplo:

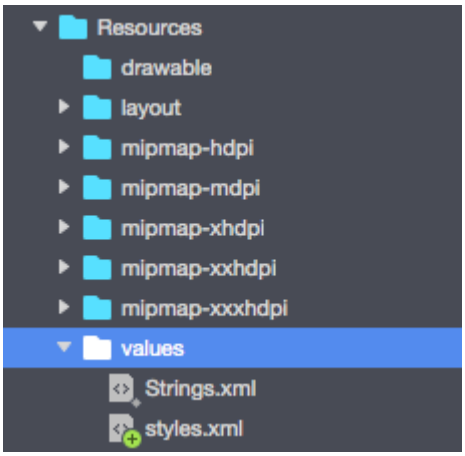
https://developer.android.com/training/appbar/index.html

Examples

Añadir barra de herramientas a la aplicación Xamarin.Android

En primer lugar, debe agregar la biblioteca Xamarin.Android.Support.V7.AppCompat para NuGet: https://www.nuget.org/packages/Xamarin.Android.Support.v7.AppCompat/

En la carpeta de "valores" en "Recursos", agregue un nuevo archivo xml llamado "styles.xml":

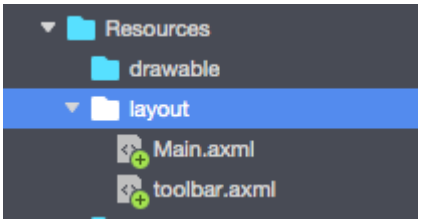


El archivo "styles.xml" debe contener el siguiente código:

```
<?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. -->
```

El siguiente paso es agregar el archivo "toolbar.axml" que contiene la definición de control de la barra de herramientas a la carpeta "diseño":



Agregue el siguiente código para definir la barra de herramientas:

```
<?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" />
```

Ahora, abra el archivo "Main.axml" y agregue el siguiente código justo debajo de la etiqueta de cierre para el primer diseño. Su código debe verse como a continuación:

```
<?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>
```

Ahora tienes que agregar información sobre el tema que usa tu aplicación. Abra el archivo "AndroidManifest" y agregue información del tema a la etiqueta de "aplicación":

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

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

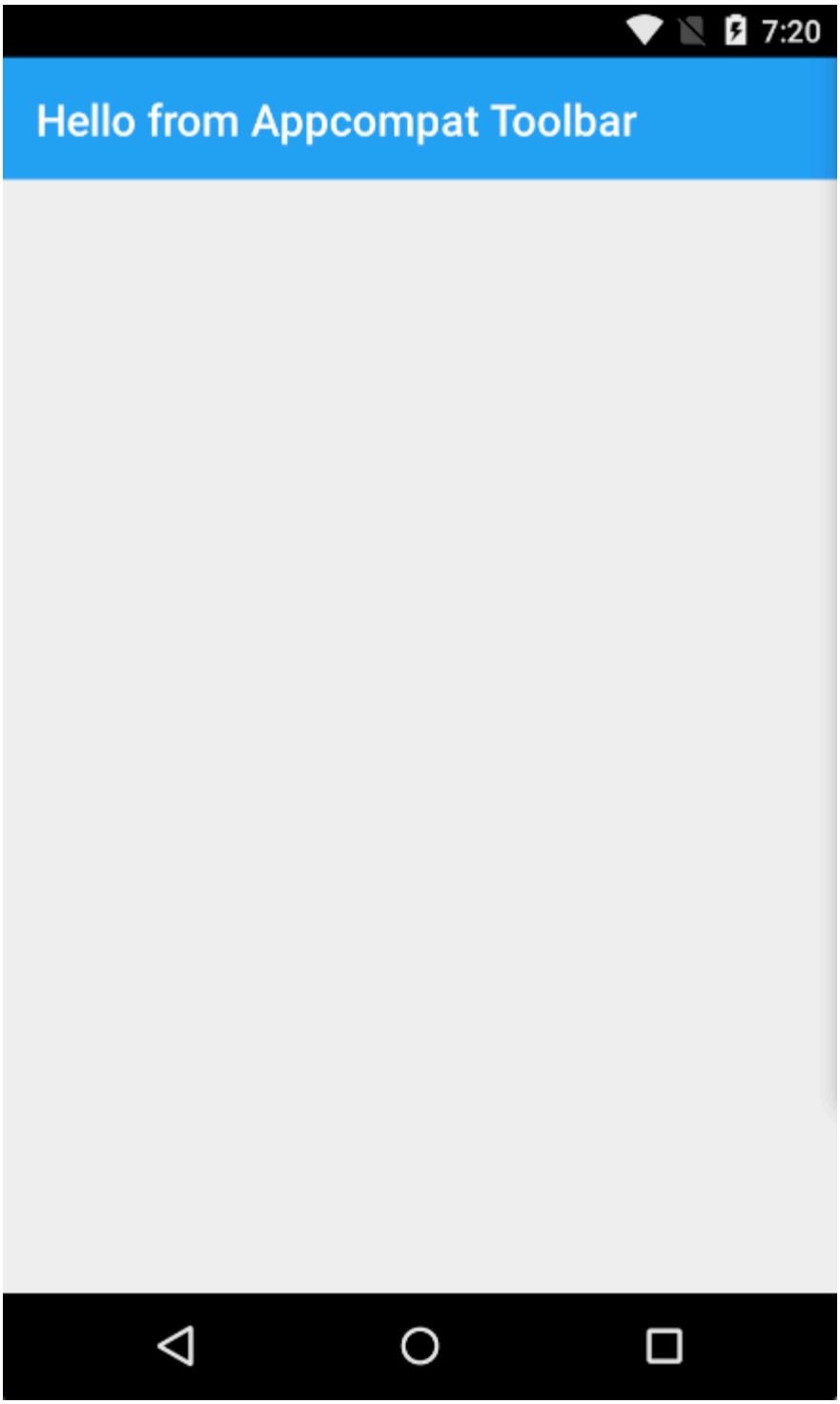
El último paso es conectar la barra de herramientas en el archivo de actividad. Abra el archivo "MainActivity.cs". Tiene que cambiar la derivación de "Actividad" a "AppCompatActivity". Ahora obtenga la referencia a la barra de herramientas y configúrela como barra de herramientas predeterminada para la actividad en el método "OnCreate". También puedes definir el título:

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

Todo el método debe verse como a continuación:

```
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";
    }
```

Reconstruye el proyecto y ejecútalo para ver el resultado:



Lea Xamarin.Android - Cómo crear una barra de herramientas en línea: https://riptutorial.com/es/xamarin-android/topic/4755/xamarin-android---como-crear-una-barra-de-herramientas

Capítulo 13: Xamarin.Android - Comunicacion Bluetooth

Introducción

En **Xamarin.Android,** las propiedades **BluetoothSocket.InputStream** y **BluetoothSocket.OutputStream** se convierten automáticamente por diseño a **System.IO.Stream** . En el caso del llamado protocolo de comunicación interactivo, cuando el servidor responde solo cuando el cliente habla con él, System.IO.Stream no es bueno porque no tiene ningún método o propiedad para obtener el número de bytes de respuesta disponibles antes de leer la respuesta.

Parámetros

Parámetro	Detalles
enchufe	Una instancia del objeto BluetoothSocket. El zócalo debe estar abierto antes de llamar a este método.
cmd	Comando como una matriz de bytes para enviar al dispositivo BT.
_mx	Dado que este método utiliza un recurso de hardware, es mejor llamarlo desde un subproceso de trabajo separado. Este parámetro es una instancia del objeto System.Threading.Mutex y se usa para sincronizar el hilo con otros hilos que, opcionalmente, llaman a este método.
se acabó el tiempo	Tiempo de espera en milisegundos entre las operaciones de escritura y lectura.

Examples

Envíe y reciba datos desde y hacia un dispositivo bluetooth usando un socket

El ejemplo siguiente utiliza Android.Runtime.InputStreamInvoker y Android.Runtime.OutputStreamInvoker tipos obtienen java.io.InputStream y java.io.OutputStream . Una vez que tengamos una instancia de **Java.IO.InputStream** , podemos usar su método **.Available ()** para obtener el número de bytes de respuesta disponibles que podemos usar en el método **.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;
}
```

Lea Xamarin.Android - Comunicacion Bluetooth en línea: https://riptutorial.com/es/xamarin-android/topic/10844/xamarin-android---comunicacion-bluetooth

Creditos

S. No	Capítulos	Contributors
1	Empezando con Xamarin.Android	Amy Burns, Community, Jon Douglas, Kevin Montrose, Ryan Weaver
2	Ciclo de vida de la aplicación - Xamarin.Andorid	CDrosos, Daniel Krzyczkowski, Steven Mark Ford
3	Cómo corregir la orientación de una imagen capturada desde un dispositivo Android	Daniel Krzyczkowski
4	Diálogos	JimBobBennett, Pilatus
5	Escaneado de códigos de barras utilizando la biblioteca ZXing en aplicaciones Xamarin	GvSharma
6	Fijaciones	EJoshuaS, Jon Douglas, jonp, Matthew, Prashant C, Sven-Michael Stübe
7	ListView personalizado	user3814750
8	Publicando tu Xamarin.Android APK	Alexandre
9	RecyclerView	Alexandre, Matthew, Ryan Alford, Sreeraj, Zverev Eugene
10	Tostadas	GONeale, Matthew, Piet, user2912553
11	Xamarin.Android - Cómo crear una barra de herramientas	Daniel Krzyczkowski, tylerjgarland
12	Xamarin.Android -	Ladislav

	Comunicacion Bluetooth