FREE eBook

LEARNING Xamarin.Android

Free unaffiliated eBook created from
Stack Overflow contributors.

#xamarin.an

droid

Table of Contents

About

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

Chapter 1: Getting started with Xamarin.Android

Remarks

Xamarin.Android allows you to create native Android applications using the same UI controls as you would in Java, except with the flexibility and elegance of a modern language (C#), the power of the .NET Base Class Library (BCL), and two first-class IDEs - Xamarin Studio and Visual Studio - at their fingertips.

For more information on installing Xamarin.Android on your Mac or Windows machine, refer to the Getting Started guides on the Xamarin developer center

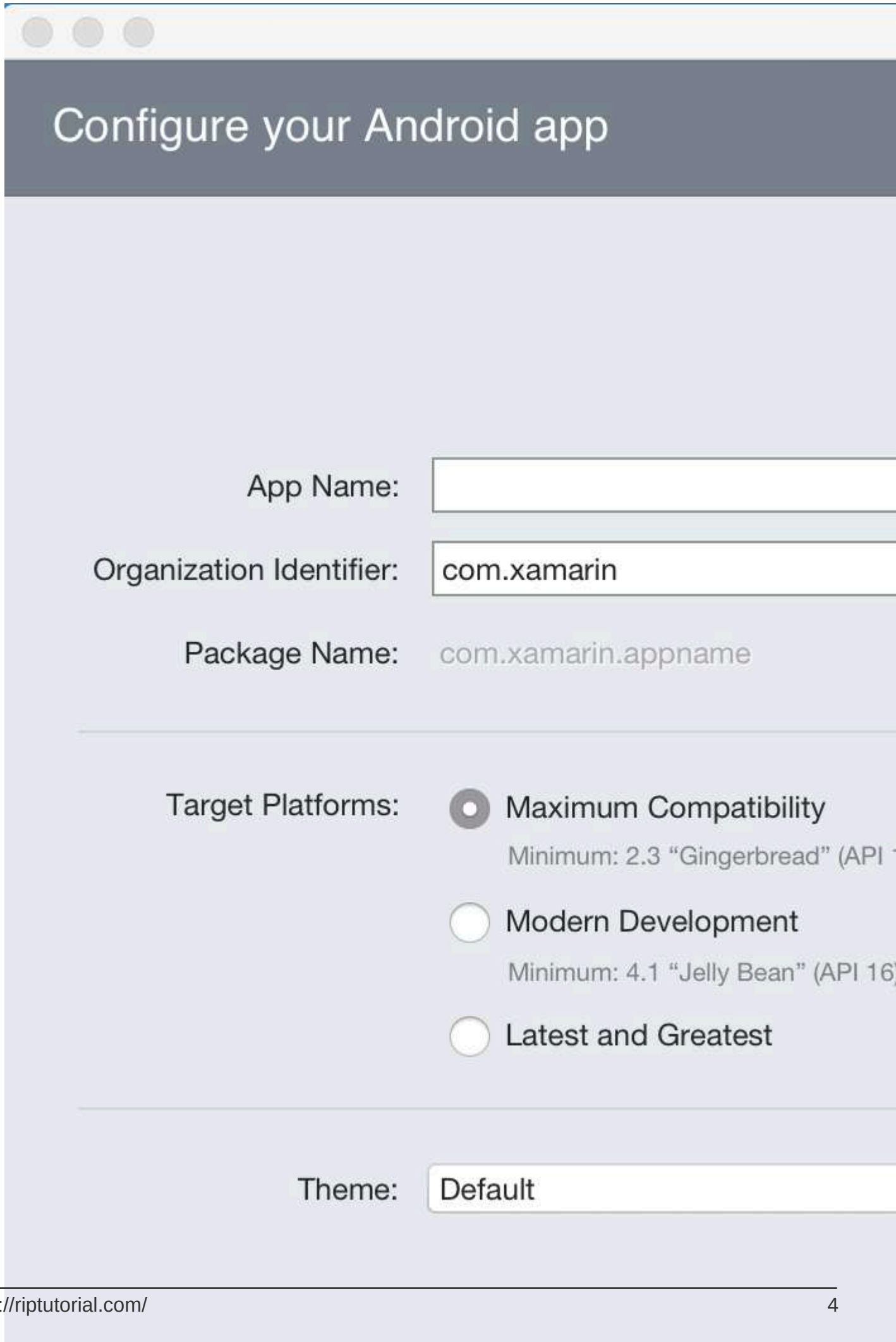
Versions

Version	Code Name	API Level	Release Date
1.0	None	1	2008-09-23
1.1	None	2	2009-02-09
1.5	Cupcake	3	2009-04-27
1.6	Donut	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	Gingerbread	9-10	2010-12-06
3.0-3.2.6	Honeycomb	11-13	2011-02-22
4.0-4.0.4	Ice Cream Sandwich	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	Lollipop	21-22	2014-11-12
6.0-6.0.1	Marshmallow	23	2015-10-05
7.0	Nougat	24	2016-08-22

Examples

Get started in Xamarin Studio

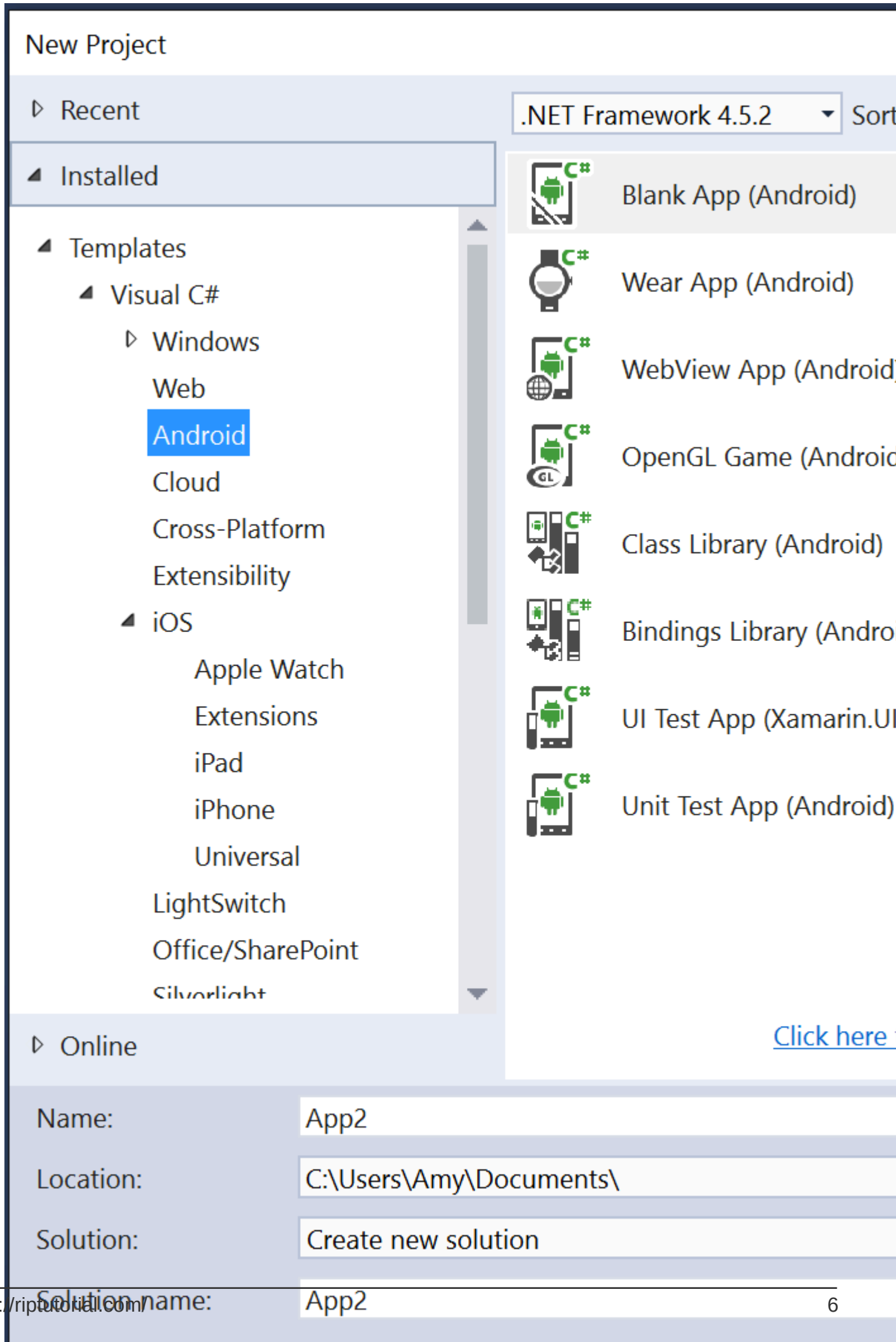
1. Browse to **File > New > Solution** to bring you up the new project dialog.
2. Select **Android App** and press **Next**.
3. Configure your app by setting your app name and organization ID. Select the Target Platform most suited for your needs, or leave it as the default. Press Next:


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. Set your Project name and Solution name, or leave as the default name. Click Create to create your project.
5. Set up your device for deployment, or configure an emulator
6. To run your application, select the **Debug** configuration, and press the Play button:

Get Started in Visual Studio

1. Browse to **File > New > Project** to bring you up the New Project dialog.
2. Navigate to **Visual C# > Android** and select Blank App:


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 (Androic
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. Give your app a **Name** and press **OK** to create your project.
4. Set up your device for deployment, or configure an emulator
5. To run your application, select the **Debug** configuration, and press the **Start** button:

Read Getting started with Xamarin.Android online: https://riptutorial.com/xamarin-android/topic/403/getting-started-with-xamarin-android

Chapter 2: App lifecycle - Xamarin.Andorid

Introduction

Xamarin.Android application lifecycle is the same as normal Android app. When talking about lifecycle we need to talk about: Application lifecycle, Activity lifecycle and Fragment lifecycle.

In the below I'll try to provide a good description and way of using them. I obtained this documentation from the official Android and Xamarin documentation and many helpful web resources provided in remarks section below.

Remarks

Interesting links to broad your knowledge about Android application lifecycle:

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

Application lifecycle

First of all you should know that you can extend Android.Application class so you can access two important methods related with app lifecycle:

- OnCreate - Called when the application is starting, before any other application objects have been created (like MainActivity).
- OnTerminate - This method is for use in emulated process environments. It will never be called on a production Android device, where processes are removed by simply killing them; No user code (including this callback) is executed when doing so. From the documentation: https://developer.android.com/reference/android/app/Application.html#onTerminate()

In Xamarin.Android application you can extend Application class in the way presented below. Add new class called "MyApplication.cs" to your project:

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

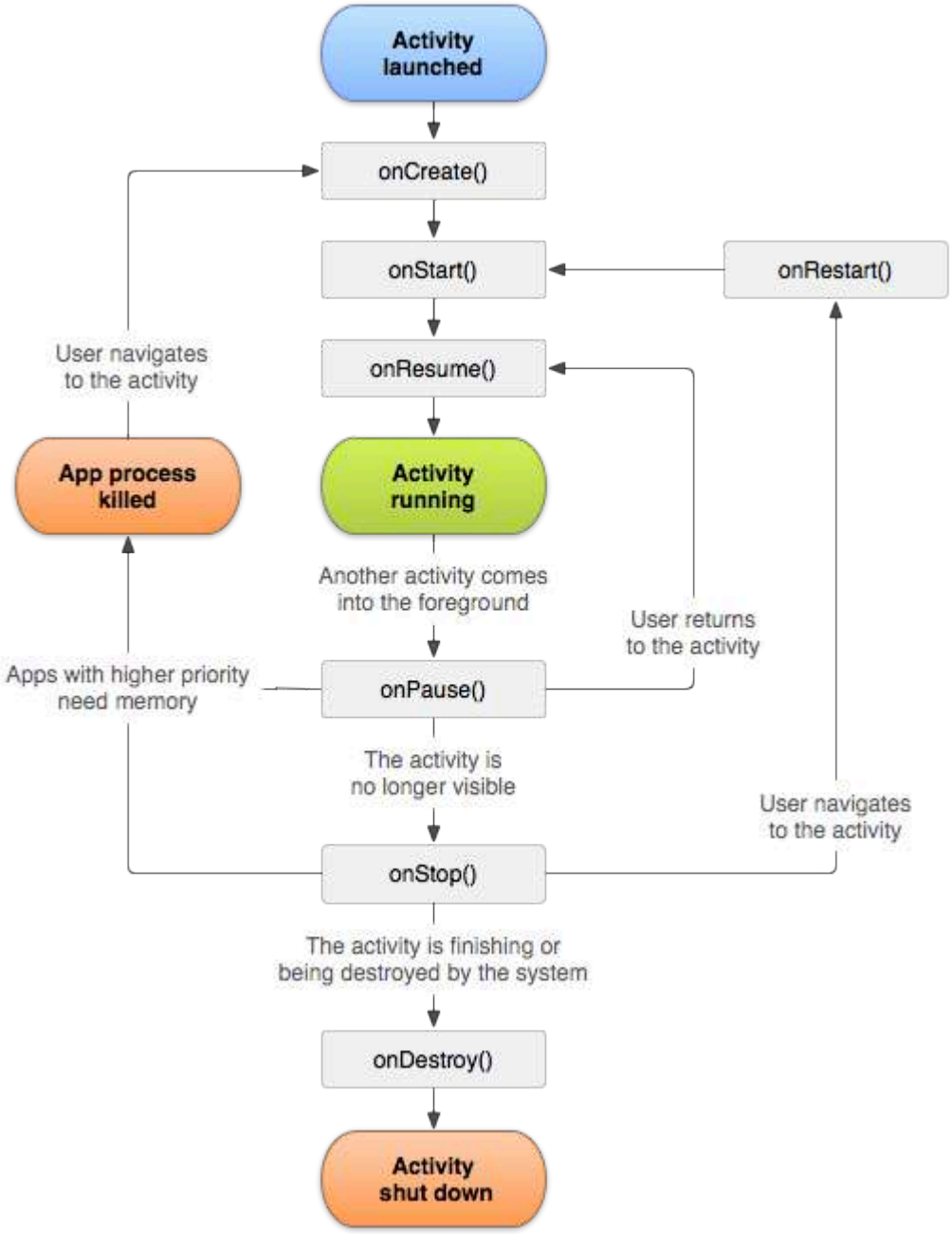
As you wrote above you can use OnCreate method. You can for instance initialize local database here or setup some additional configuration.

There is also more methods which can be overridden like: OnConfigurationChanged or OnLowMemory.

Activity lifecycle

Activity lifecycle is quite more complex. As you know Activity is single page in the Android app where user can perform interaction with it.

On the diagram below you can see how Android Activity lifecycle looks like:



As you can see there is specific flow of Activity lifecycle. In the mobile application you have of course methods in each Activity class that handle specific lifecycle fragment:

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

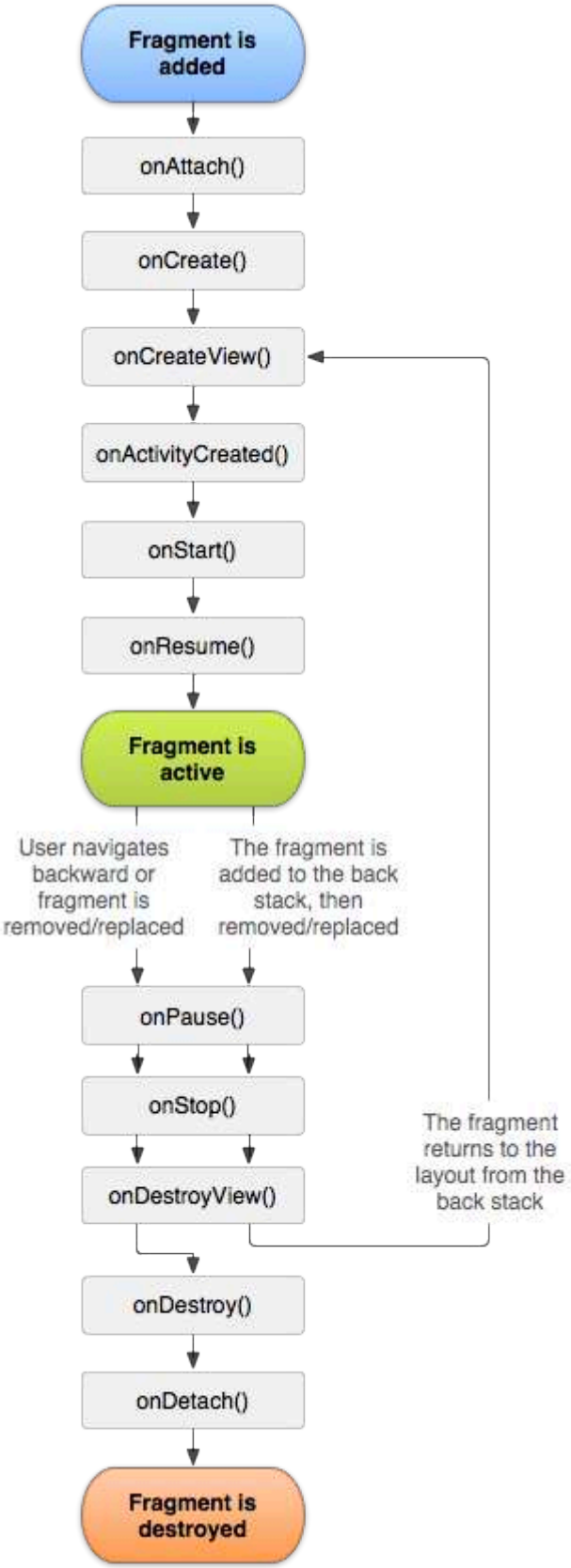
There is good description in the official Android documentation:

- The entire lifetime of an activity happens between the first call to onCreate(Bundle) through to a single final call to onDestroy(). An activity will do all setup of "global" state in onCreate(), and release all remaining resources in onDestroy(). For example, if it has a thread running in the background to download data from the network, it may create that thread in onCreate() and then stop the thread in onDestroy().
- The visible lifetime of an activity happens between a call to onStart() until a corresponding call to onStop(). During this time the user can see the activity on-screen, though it may not be in the foreground and interacting with the user. Between these two methods you can maintain resources that are needed to show the activity to the user. For example, you can register a BroadcastReceiver in onStart() to monitor for changes that impact your UI, and unregister it in onStop() when the user no longer sees what you are displaying. The onStart() and onStop() methods can be called multiple times, as the activity becomes visible and hidden to the user.
- The foreground lifetime of an activity happens between a call to onResume() until a corresponding call to onPause(). During this time the activity is in front of all other activities and interacting with the user. An activity can frequently go between the resumed and paused states -- for example when the device goes to sleep, when an activity result is delivered, when a new intent is delivered -- so the code in these methods should be fairly lightweight.

Fragment lifecycle

As you know you can have one activity but different fragments embedded in it. That is why fragment lifecycle is also important for developers.

On the diagram below you can see how Android fragment lifecycle looks like:



As described in the official Android documentation you should implement at least below three

methods:

- OnCreate - the system calls this when creating the fragment. Within your implementation, you should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.
- OnCreateView - The system calls this when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a View from this method that is the root of your fragment's layout. You can return null if the fragment does not provide a UI.
- OnPause - The system calls this method as the first indication that the user is leaving the fragment (though it does not always mean the fragment is being destroyed). This is usually where you should commit any changes that should be persisted beyond the current user session (because the user might not come back).

Here is sample implementation 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();
    }
}
```

Of course you can add additional methods here if you want to handle different states.

Full sample on GitHub

If you would like to get base project with methods described below you can download Xamarin.Android application template from my GitHub. You can find examples for:

- Application lifecycle methods
- Activity lifecycle methods
- Fragment lifecycle methods

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

Read App lifecycle - Xamarin.Andorid online: https://riptutorial.com/xamarin-android/topic/8842/app-lifecycle---xamarin-andorid

Chapter 3: Barcode scanning using ZXing library in Xamarin Applications

Introduction

Zxing library is well known for image processing. Zxing was based on java and .Net module is also available and it can be used in xamarin applications. click here to check official documentation. http://zxingnet.codeplex.com/

I recently used this libarry.

Step1: Add ZXing.Net.Mobile component into solution.

step2: In whichever activity we need to show barcode scanner, in that activity initialise MobileBarcodeScanner.

Step3: Write below code when tapped on any view to start scanning.

Examples

Sample Code

```
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;
}
```

Read Barcode scanning using ZXing library in Xamarin Applications online: https://riptutorial.com/xamarin-android/topic/9526/barcode-scanning-using-zxing-library-in-xamarin-applications

Chapter 4: Bindings

Examples

Removing Types

It is possible to instruct the Xamarin.Android Bindings Generator to ignore a Java type and not bind it. This is done by adding a `remove-node` XML element to the metadata.xml file:

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

Implementing Java interfaces

If a java library contains interfaces that should be implemented by the user (e.g. click listeners like `View.IOnClickListener` or callbacks), the implementing class has to inherit -- directly or indirectly -- from `Java.Lang.Object` or `Java.Lang.Throwable`. This is a common error, because the packaging steps just print a warning that is overlooked easily:

> Type 'MyListener ' implements Android.Runtime.IJavaObject but does not inherit from Java.Lang.Object. It is not supported.

Wrong

The usage of this implementation will result in unexpected behavior.

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

    public void Dispose()
    {
    }


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

Correct

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

```
}
```

Bindings libraries may rename methods and interfaces

Not everything in a bindings library will have the same name in C# as it does in Java.

In C#, interface names start with "I", but Java has no such convention. When you import a Java library, an interface named `SomeInterface` will become `ISomeInterface`.

Similarly, Java doesn't have properties like C# does. When a library is bound, Java getter and setter methods might be refactored as properties. For example, the following Java code

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

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

may be refactored as

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

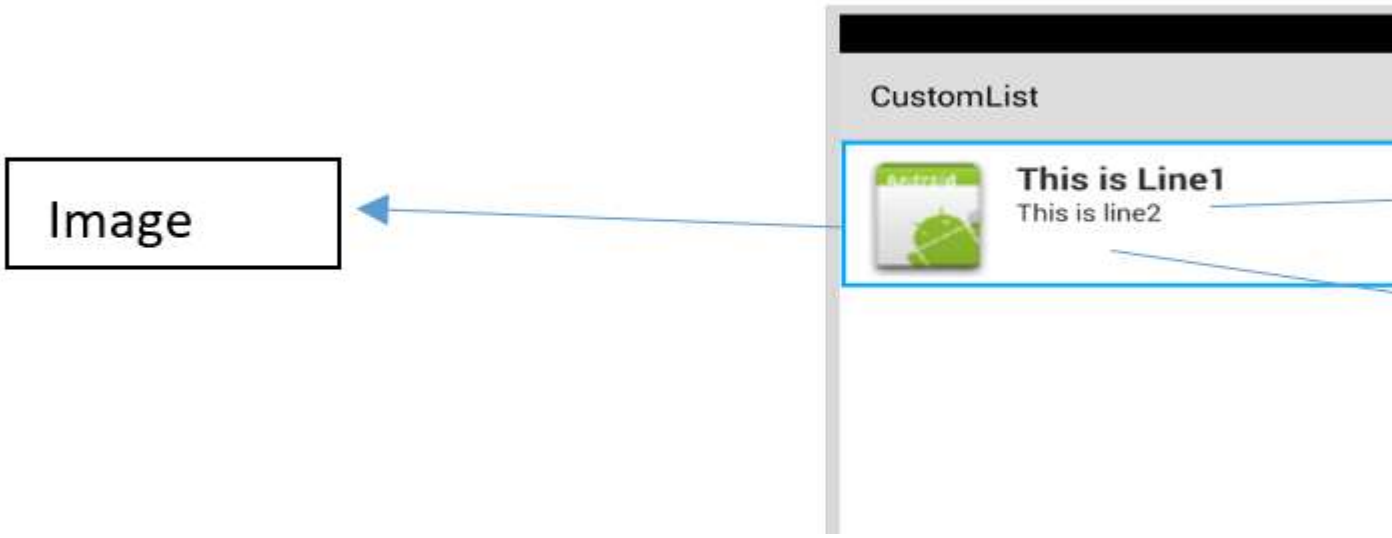
when it's bound.

Read Bindings online: https://riptutorial.com/xamarin-android/topic/771/bindings

Chapter 5: Custom ListView

Examples

Custom Listview comprises of rows that are designed as per the users needs.

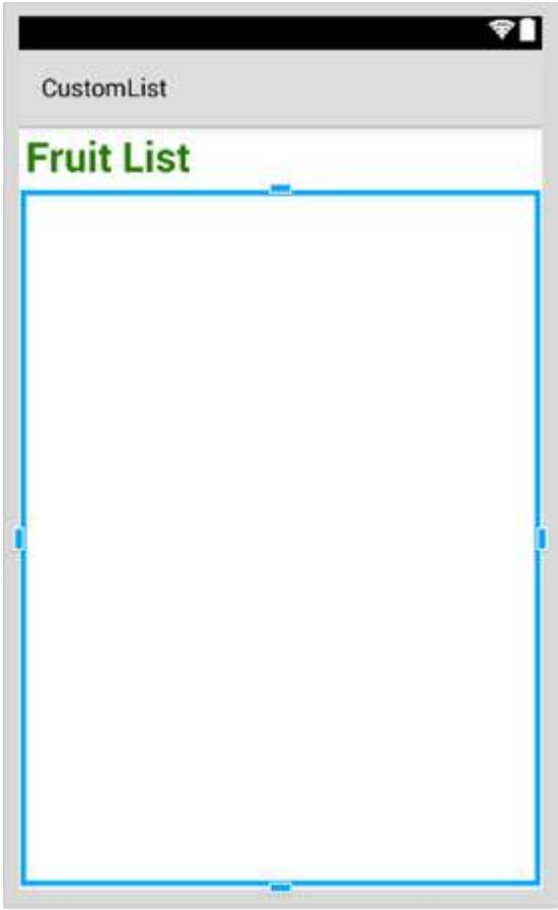


For the layout above your customrow.axml file is as shown below

```
    <?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>
```

Then you can design your main.axml, which contains a textview for the header and a listview.



Hope that is easy...

Next create your Data.cs class that will represent your row objects

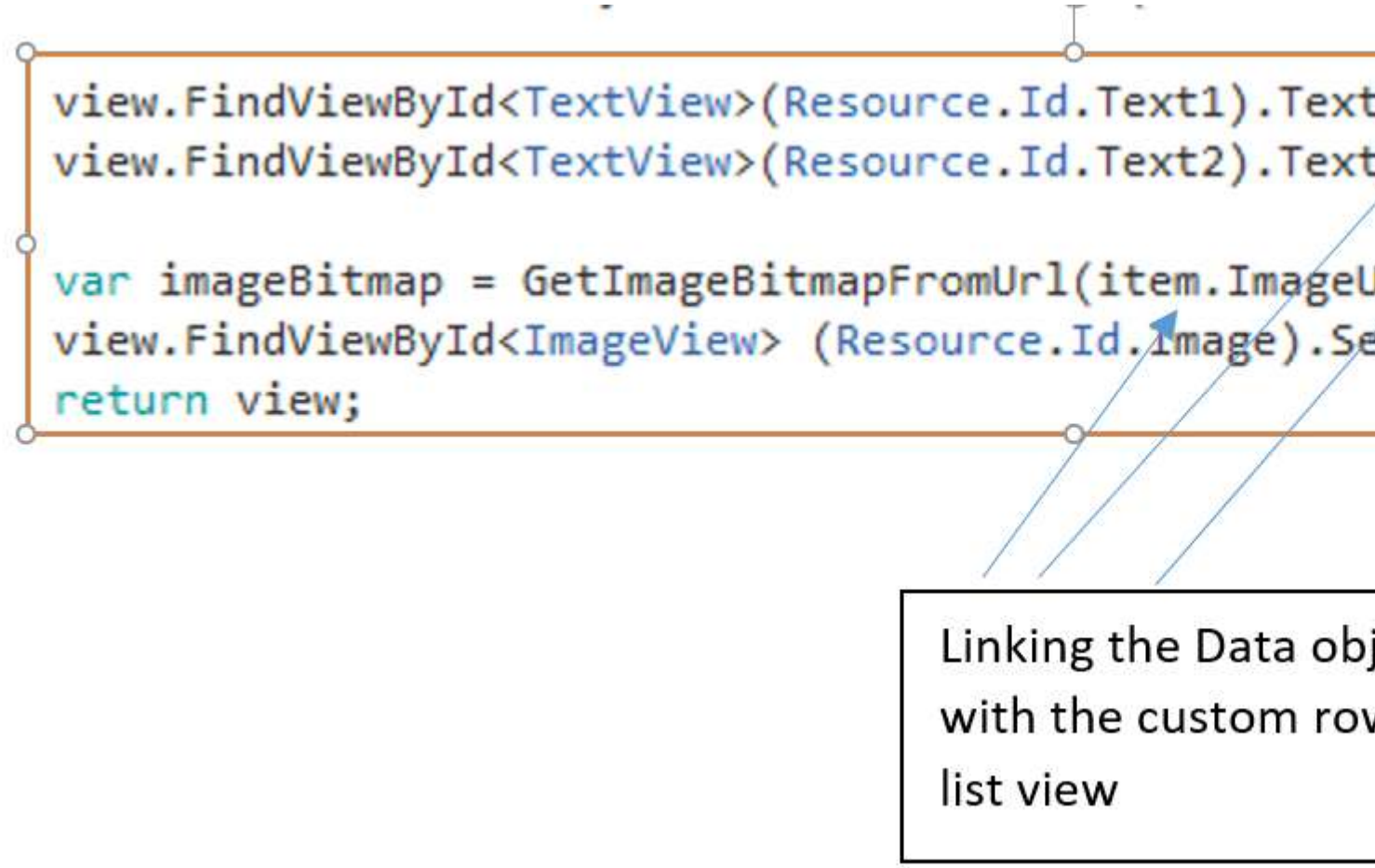
```
public class Data
{
    public string Heading;
    public string SubHeading;
    public string ImageURI;

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

Next you need the DataAdapter.cs class, Adapters link your data with the underlying view

```
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;
    }

}
```

The most important part is inside the GetView Function, this is where you link your object to your custom row.



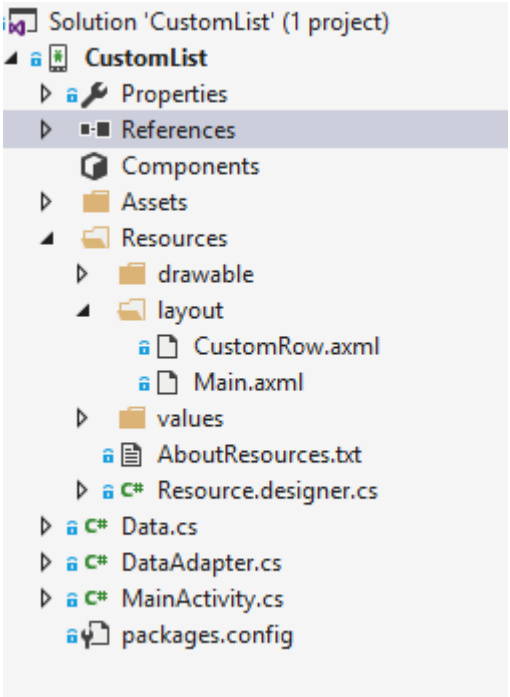
The GetImageBitmapFromUrl is not part of the dataadapter but I have put this over here for simplicity.

At last we come to the 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);

    }
```

Your final project structure is as show below.



If everything is fine you should see the output as shown

Read Custom ListView online: https://riptutorial.com/xamarin-android/topic/6406/custom-listview

Chapter 6: Dialogs

Remarks

Setting the `Context` of the dialog

When creating a `Dialog` from an `Activiy` we can use `this` as the context.

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

With `Fragments` we use the property `Context`.

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

Button types

`SetNeutralButton()` can be used for a simple notification and confirmation that the notification is read. `SetPositiveButton()` can be used for a confirmation for example: "Are you sure you want to delete this item?" `SetNegativeButton()` is for dismissing the dialog and cancelling it's action.

Disable cancel from backbutton

If we want to make sure that the user can't dismiss the dialog with the back button we can call `SetCanceable(false)`. This only works for the back button.

Rotation

If the screen is rotated whilst a dialog is visible it will be dismissed and the ok and cancel actions will not be called. You will need to handle this inside your activity and re-show the dialog after the activity has been reloaded.

To get around this use a `DialogFragment` instead.

Examples

Alert dialog

Creating an alert dialog

```
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();
```

Read Dialogs online: https://riptutorial.com/xamarin-android/topic/2510/dialogs

Chapter 7: Dialogs

Parameters

commonly used Public Method	Use
SetTitle(String)	Sets Title for the dialog
SetIcon(Drawable)	Set Icon for the alert dialog
SetMessage(string)	Set the message to display.
SetNegativeButton(String, EventHandler)	Set a listener to be invoked when the negative button of the dialog is pressed.
SetPositiveButton(String, EventHandler)	Set a listener to be invoked when the positive button of the dialog is pressed.
SetNeutralButton(String, EventHandler)	Set a listener to be invoked when the neutral button of the dialog is pressed.
SetOnCancelListener(IDialogInterfaceOnCancelListener)	Sets the callback that will be called if the dialog is canceled.
SetOnDismissListener(IDialogInterfaceOnDismissListener)	Sets the callback that will be called when the dialog is dismissed for any reason.
Show()	Creates a AlertDialog with the arguments supplied to this builder and Dialog.Show's the dialog.

Remarks

Requirements

Namespace: Android.App

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

Assembly Versions: 0.0.0.0

Public Constructors

AlertDialog.Builder(Context) :-

Constructor using a context for this builder and the AlertDialog it creates.

AlertDialog.Builder(Context, Int32) :-

Constructor using a context and theme for this builder and the AlertDialog it creates.

Using Material Design AlertDialog

In order to use the modern AlertDialog:

1. Install Support v7 AppCompat library from the NuGet packages
2. Replace AlertDialog with Android.Support.V7.App.AlertDialog or add the following statement at the top to make your dialog shine.

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

Examples

AlertDialog

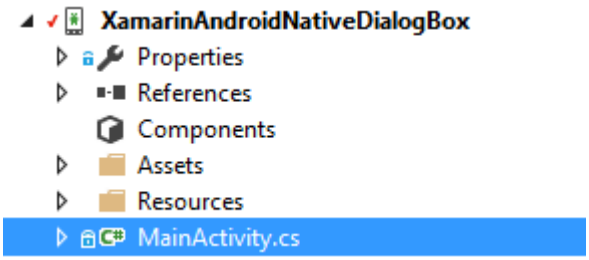
```
// 1. Instantiate an AlertDialog.Builder with its constructor
// the parameter this is the context (usually your activity)
AlertDialog.Builder builder = new AlertDialog.Builder(this);

// 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();
```

Simple Alert Dialog Example

We shall create a simple Alert Dialog in Xamarin.Android

Now considering you have gone through the getting started guide from the documentation.

You must be having the project structure like this:



Your Main Activity must be looking like this:

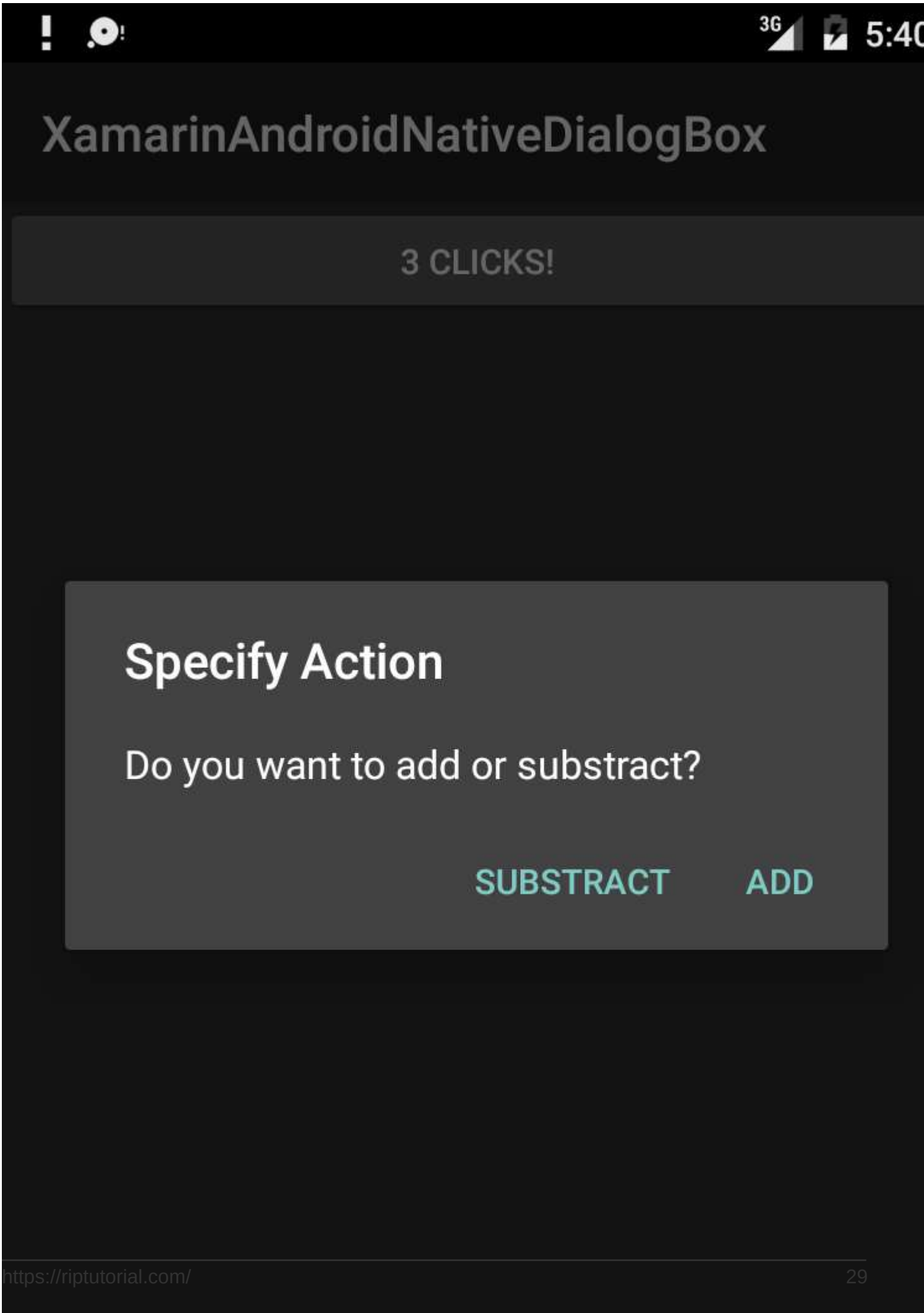
```
public class MainActivity : Activity
{
int count = 1;

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++); };
}
}
```

Now What we shall do is, instead of adding one to the counter on button click, we shall ask user if he wants to add or substract one in a simple Alert Dialog

And on Click of the Positive or the negative button we will take the action.

```
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();
};
```

screenshot:


3G
5:40
XamarinAndroidNativeDialogBox
3 CLICKS!
Specify Action
Do you want to add or substract?
SUBSTRACT
ADD

Chapter 8: How to correct the orientation of a picture captured from Android device

Remarks

1. This app sample is available on my GitHub below:

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

2. Xamarin Mobile component documentation is available below:

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

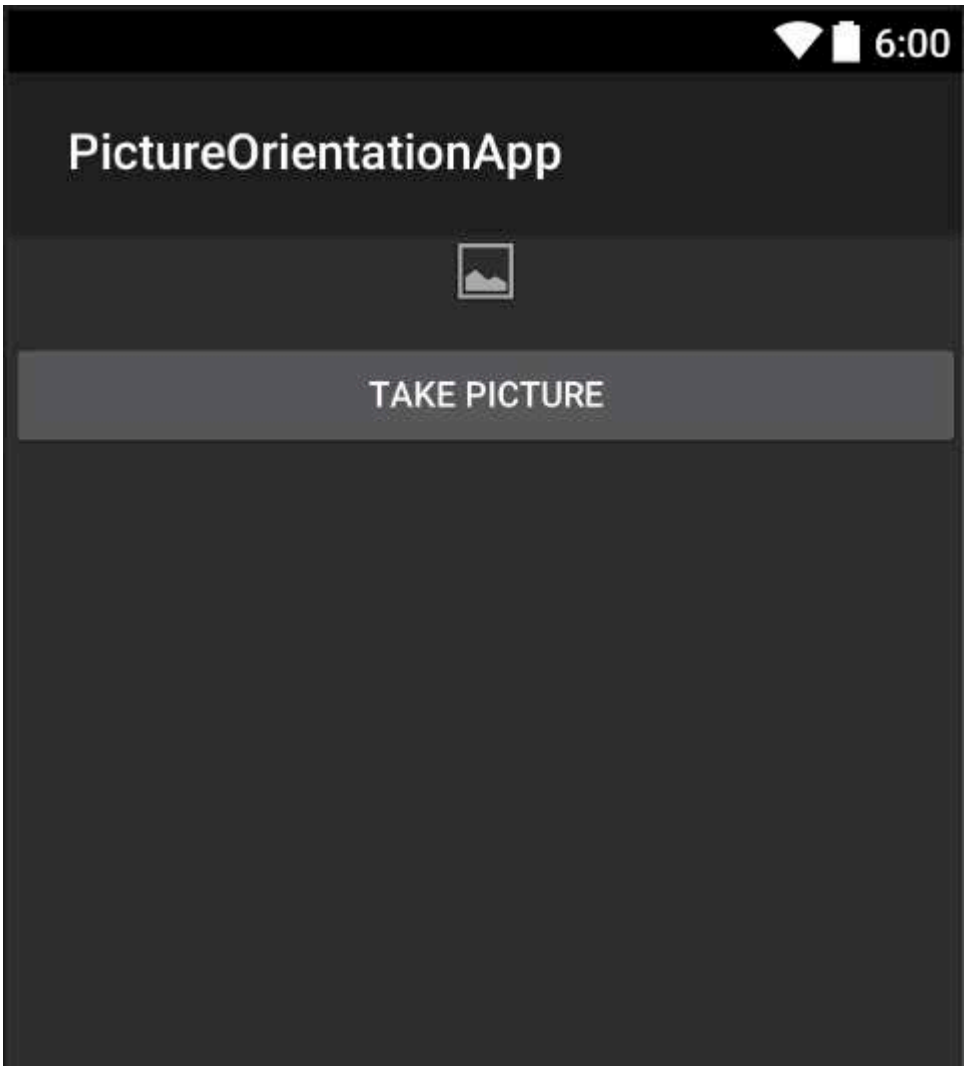
Examples

How to correct the orientation of a picture captured from Android device

This example shows how to take image and display it correctly on the Android device.

Firstly we have to create sample application with one button and one imageview. Once user clicks on the button camera is launched and after user selects picture it will be displayed with the proper orientation on the screen.

1. Add button named "TakePictureButton" and imageview named "TakenPictureImageView":

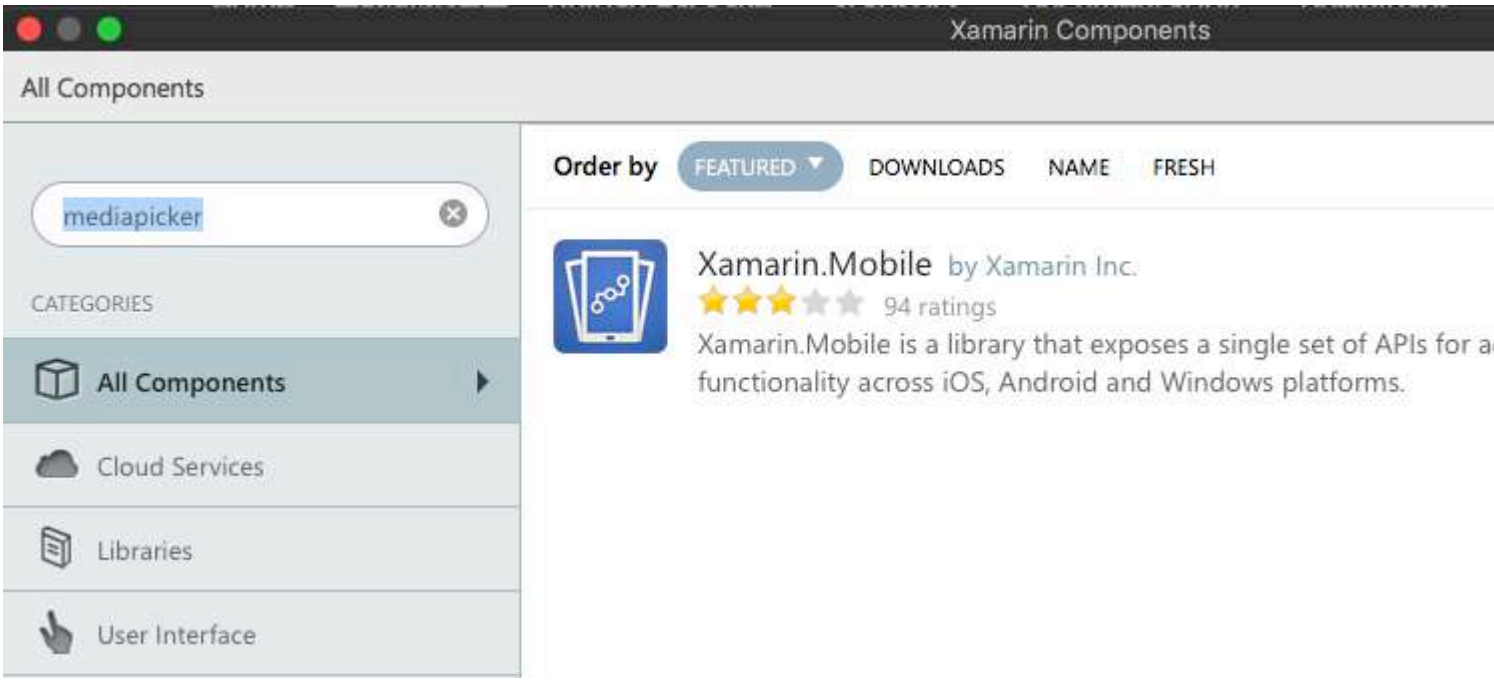


2. Now open activity code behind:

Here firstly get reference to your controls:

```
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. In our application we will use Xamarin Mobile component available in the Components Store:



4. Once you add it to the project we can move on. Add below code which is responsible for launching camera. This method should be invoked in the button click as you can see in the above code:

```
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. Once user takes picture we should display it in the proper orientation. To do it use below method. It is responsible for retrieveing exif information from the taken image (including orientation during the moment of taking picture) and than creating bitmap with the proper orientation:

```
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;
```

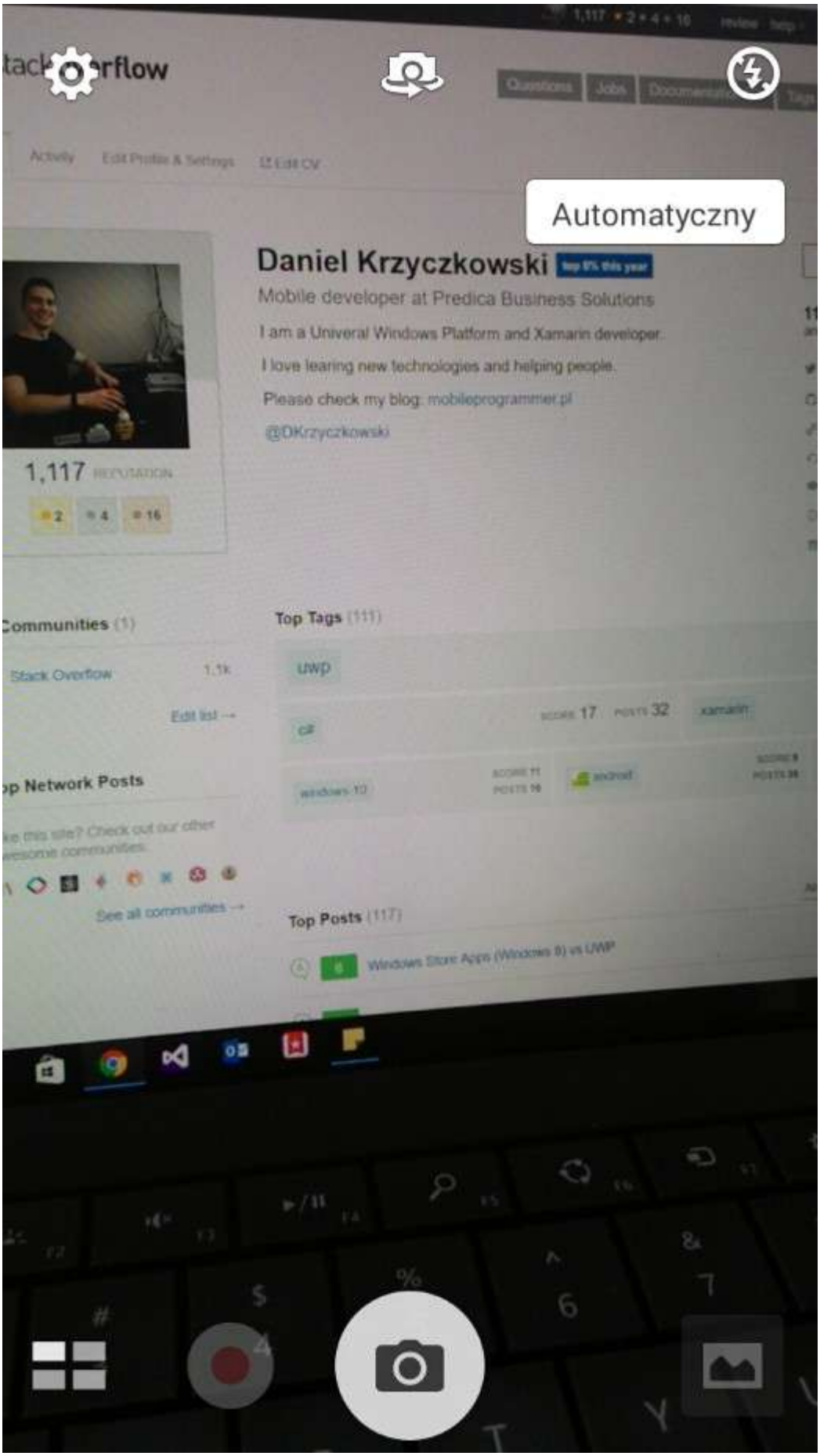
```
        height_tmp /= 2;
        scale++;
    }

    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;
```

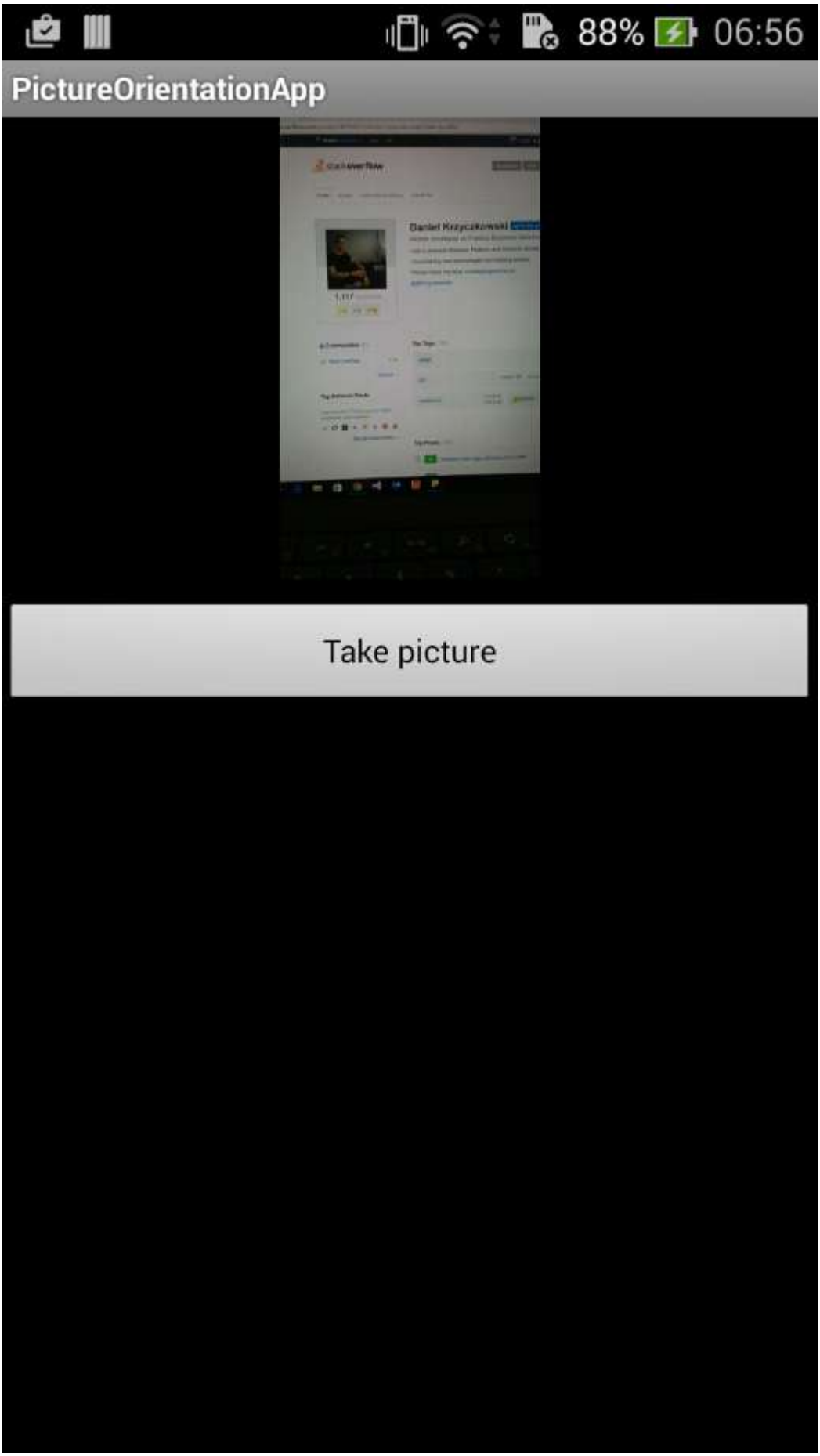
```
        }

        return resizedBitmap;
    }

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

6. Above method should be invoked in the OnActivityResult method invoked after user takes the picture:

```
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. Launch the application. Take photo an see the result:


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
Communities (1)
Top Tags (111)
Top Posts (117)



That's it. Now you will have all you taken picture displayed in correct orientation.

Read How to correct the orientation of a picture captured from Android device online: https://riptutorial.com/xamarin-android/topic/6683/how-to-correct-the-orientation-of-a-picture-captured-from-android-device

Chapter 9: Publishing your Xamarin.Android APK

Introduction

This topic shows information on how to prepare your Xamarin.Android app for release mode and how to optimize it.

Examples

Preparing your APK in the Visual Studio

You finished your app, tested on debug mode and it is working perfect. Now, you want to prepare it to publish in the Google Play Store.

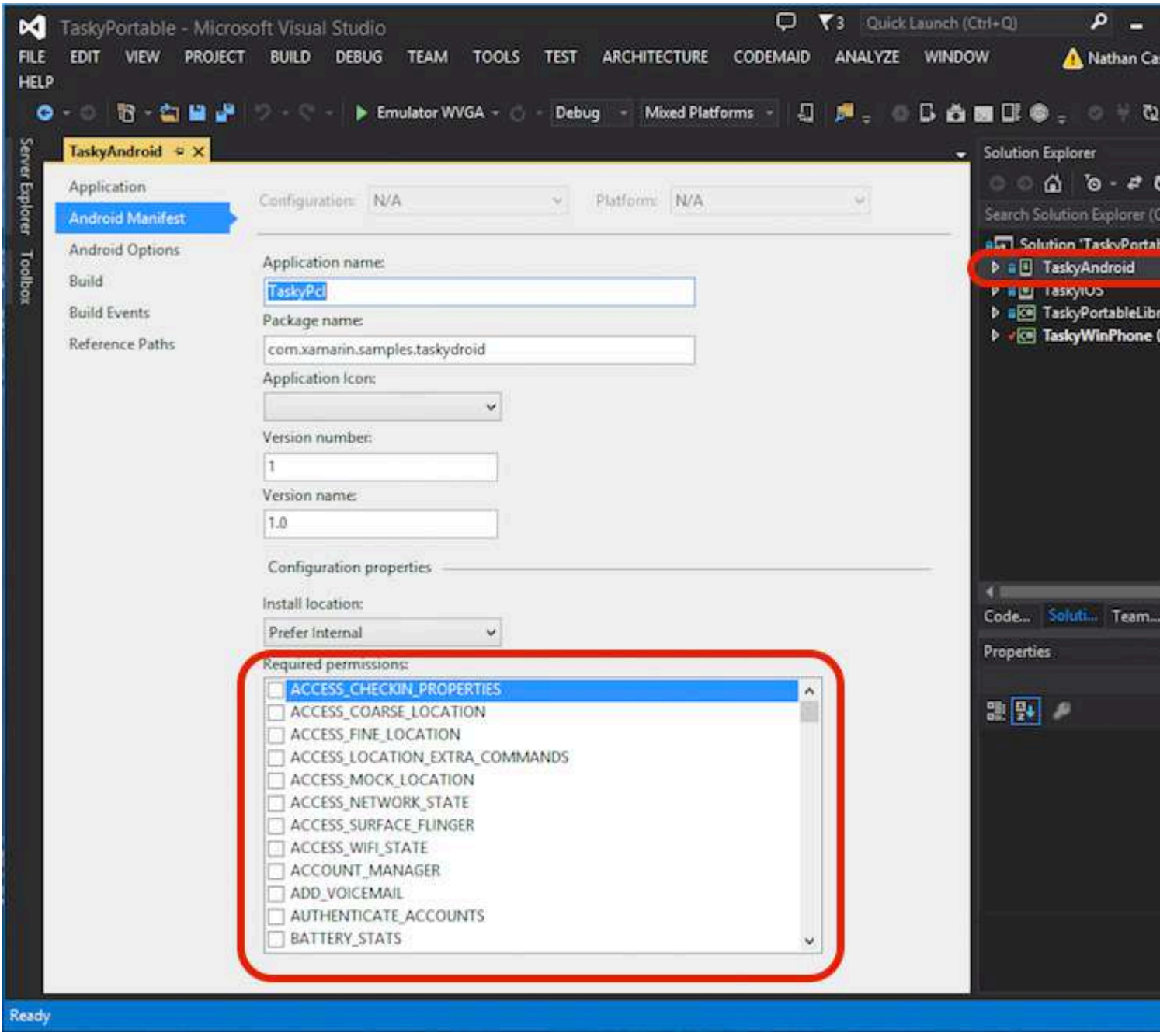
Xamarin documentation provides good informations in here:

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

---

Android Manifest

First, in Visual Studio, right-click your Xamarin.Android project in the Solution Explorer and select Properties. Then, go to the Android Manifest tab, to see this screen:



Unlike in Android Studio or Eclipse, you don't need the set the AndroidManifest.xml file by writing; Xamarin and Visual Studio do that for you. Activities, BroadcastReceivers and Services are inserted into Android Manifest by declaring specific attributes in their classes.

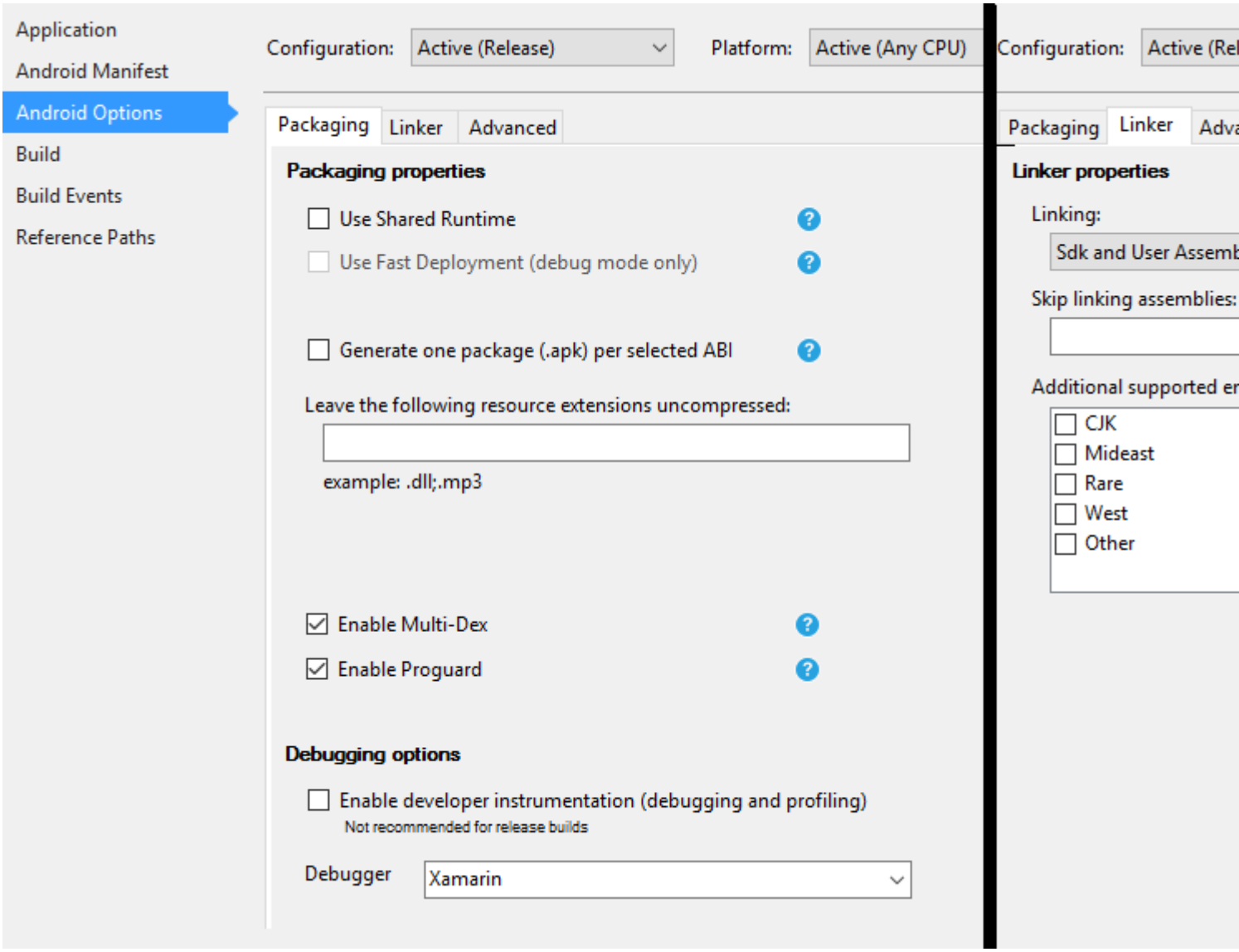
In this screen, the options are:

- **Application name**: This is the app name that will be visible for the user.
- **Package name**: This is the package name. It must be unique, meaning that it must not use the same package name of other apps in the Google Play Store.
- **Application Icon**: This is the icon that will be visible to the user, equivalent to the @drawable/ic_launcher used in Android Studio or Eclipse projects.
- **Version number**: The version number is used by Google Play for version control. When you want to publish an APK for an updated version of your app, you must add 1 to this number for each new upgrade.
- **Version name**: This is the version name that will be displayed to the user.

- **Install location**: This determines where your APK will be installed, in the device storage or SD Card.
- **Required permissions**: Here you determine which permissions are necessary for your app.

Android Options

In the screen below, you can configure the compiler options. Using the right options here can reduce a lot your APK size and also prevent errors.



- **Configuration**: **Active (Release)**.
- **Platform**: **Active (Any CPU)**. These are necessary to build your APK for the Google Play Store. If the Configuration is set to Debug, it will not be accepted by Google Play.
- **Use Shared Runtime**: **false**. If you set it to true, the APK will use Mono Runtime to execute. The Mono Runtime is installed automatically when debugging through USB, but not in the Release APK. If Mono Runtime is not installed in the device and this option is set to true in the Release APK, the app will crash.
- **Generate one package (.apk) per selected ABI**: **false**. Create your APK for as many platforms as possible, for compatibility reasons.
- **Enable Multi-Dex**: **true**, but you can set it to false if your app is not very complex (that is,

has less than 65536 methods, see here).
- **Enable Proguard**: **true**. This enables the Proguard tool that obfuscates Java code in your app. Note that it does not apply to .NET code; if you want to obfuscate .NET code, you must use Dotfuscator. More information on Proguard for Xamarin.Android can be found here.
- **Enable developer instrumentation (debugging and profiling)**: **false** for Release APK.
- **Linking**: **SDK and User Assemblies**. This will make the Xamarin Linker to remove all unused classes from SDK and your code, reducing the APK size.

Important

Xamarin.Linker may sometimes remove classes that are not seemed to be used by your code, especially if they are in the project's Core (PCL library). To avoid that, you can either set the Linking to "Sdk Assemblies Only" or use the Preserve attribute in your classes, example:

PreserveAttribute.cs

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

In a class:

```
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;
        }
    }
}
```

- **Supported architectures**: **Select all**, for compatibility reasons.

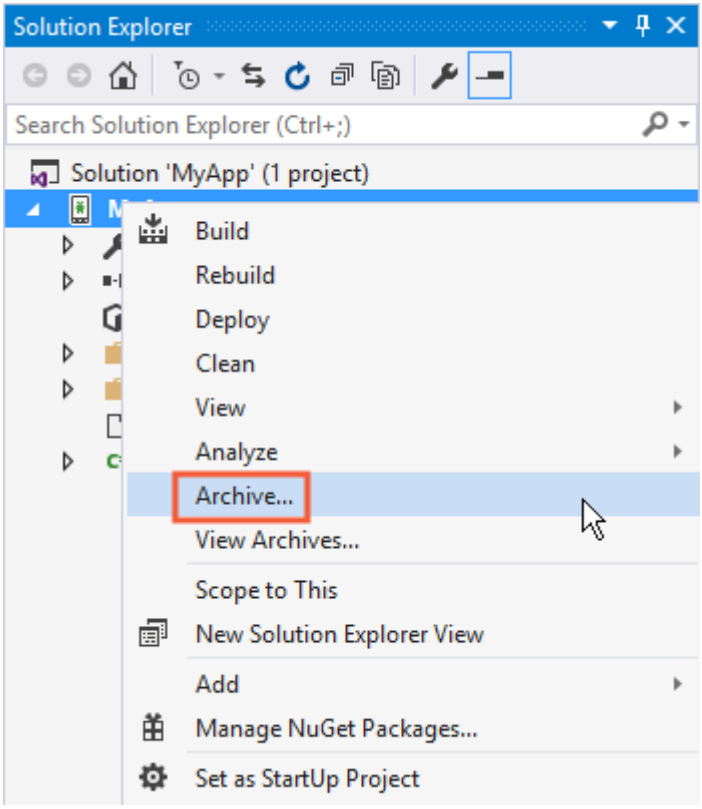
After configuring everything, Rebuild the Project to make sure that it builds successfully.

Creating the APK for Release mode

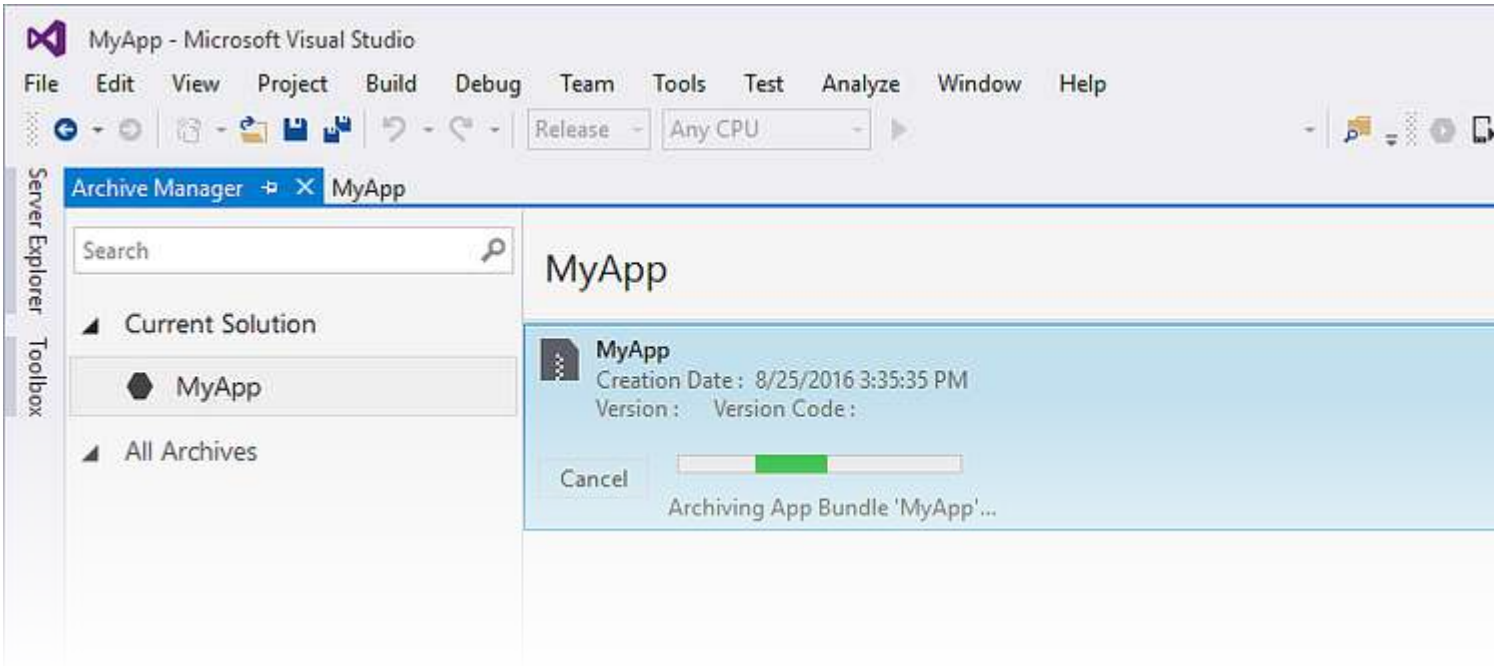
You finished configuring your Android project for Release. The tutorial below shows how to generate the APK in Visual Studio. A full tutorial from Xamarin documentation can be found here:

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

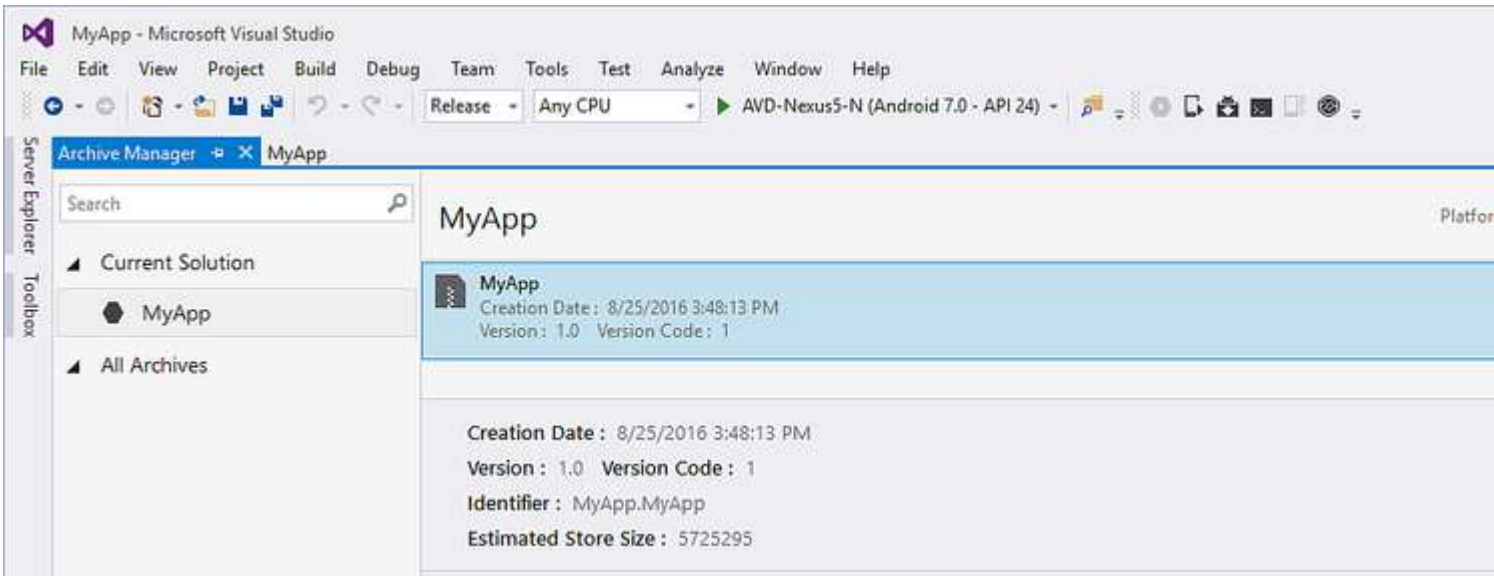
To create the APK file, right-click the Xamarin.Android project in the Solution Explorer and select Archive...



This will open the Archive manager and begin archiving the project, preparing to create the APK file.

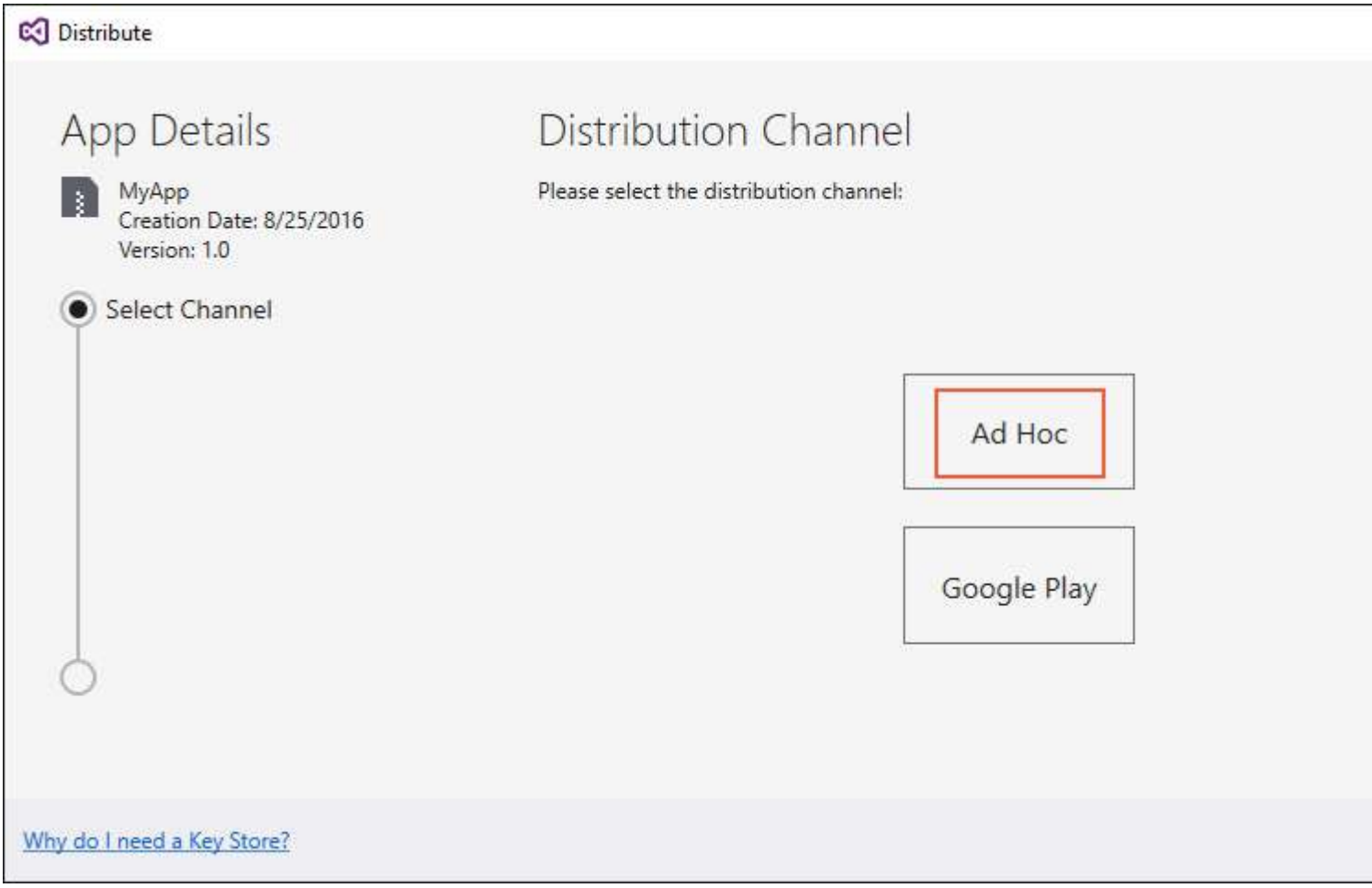


When it finishes archiving the project, click in Distribute... to proceed.

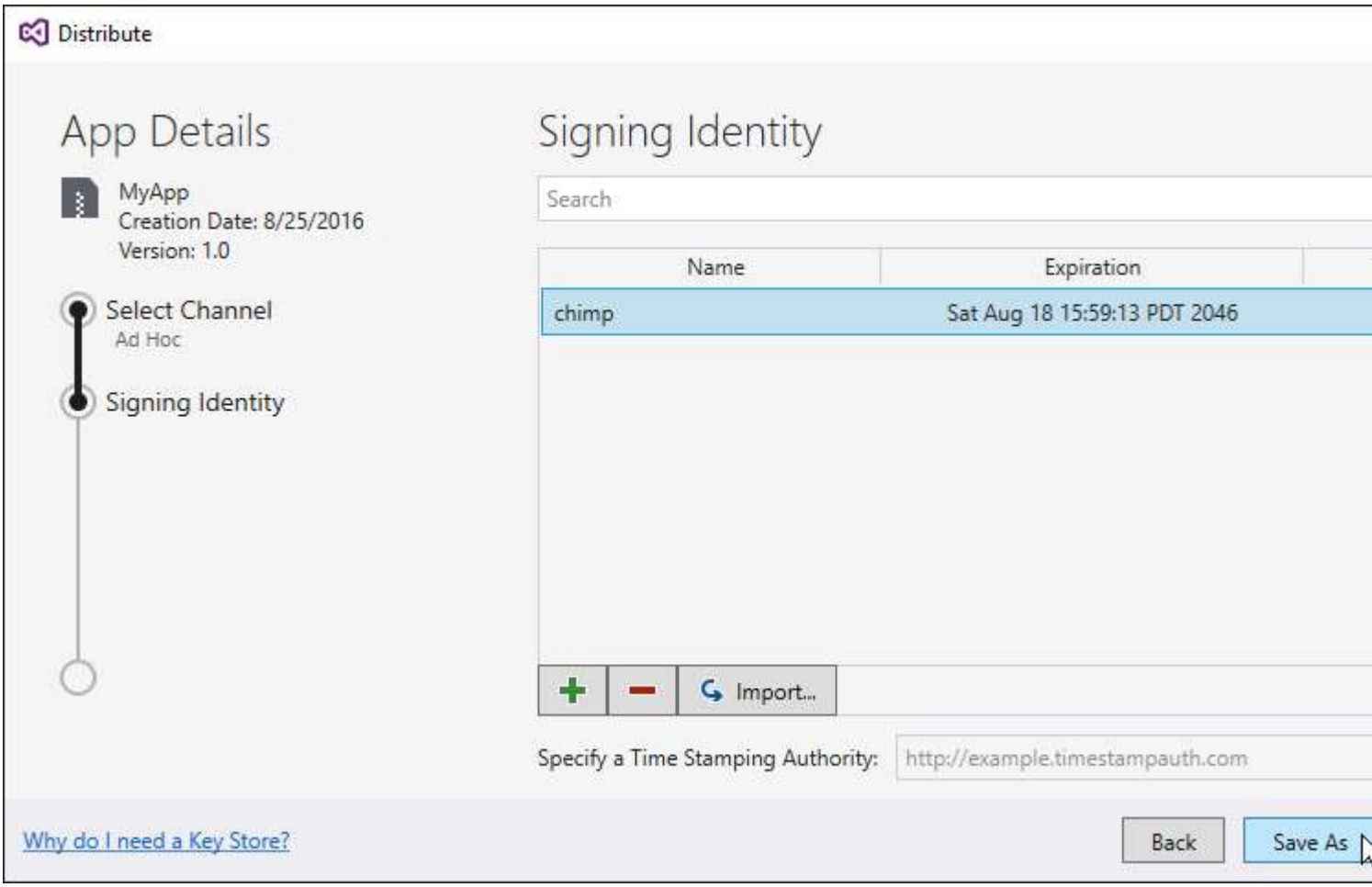


The Distribute screen will present you two options: Ad-hoc and Google Play. The first will create an APK and save it in your computer. The second will directly publish the app in Google Play.

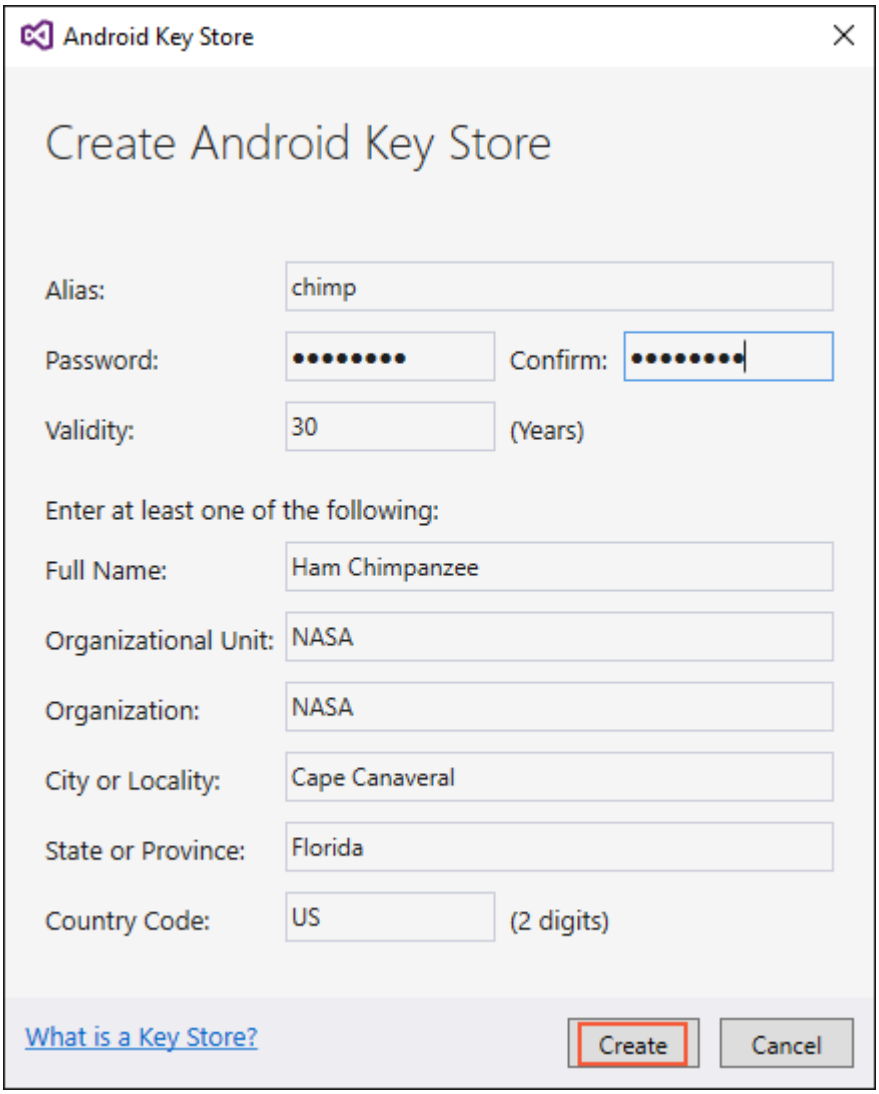
Choosing the first is recommended, so you can test the APK in other devices if you want.



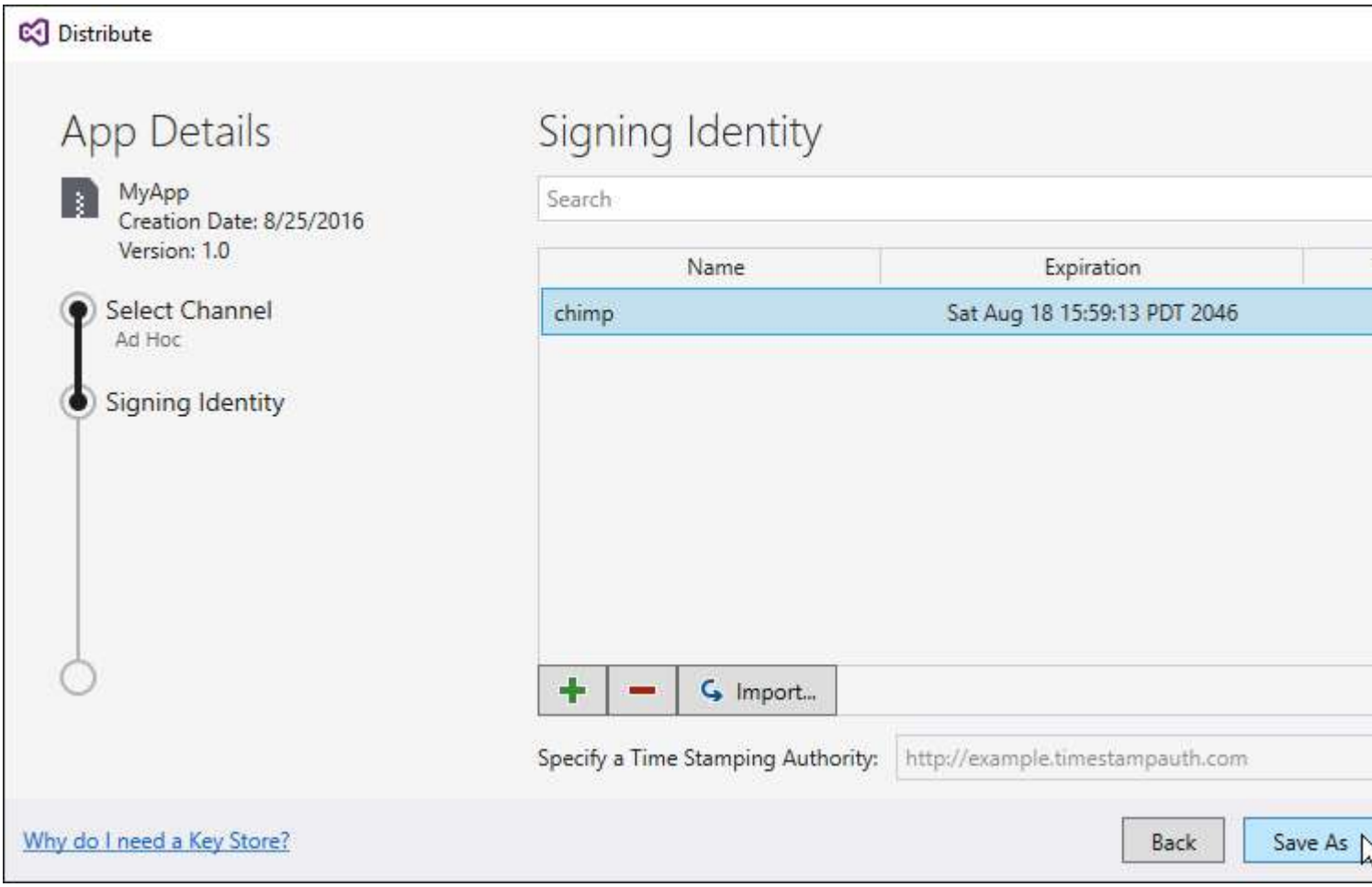
In the following screen, an Android Key Store is needed to sign the APK. If you already have one, you can use it by clicking in Import...; if you don't, you can create a new Android Key Store by clicking in +.

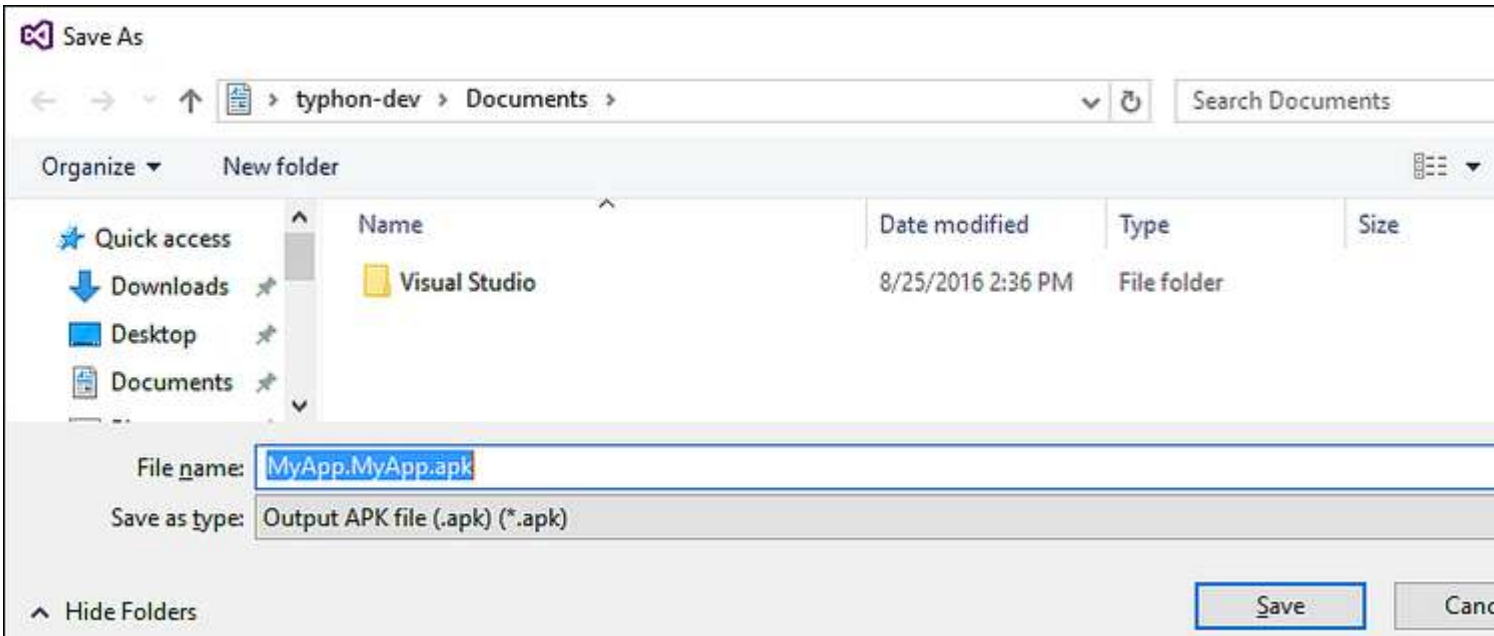


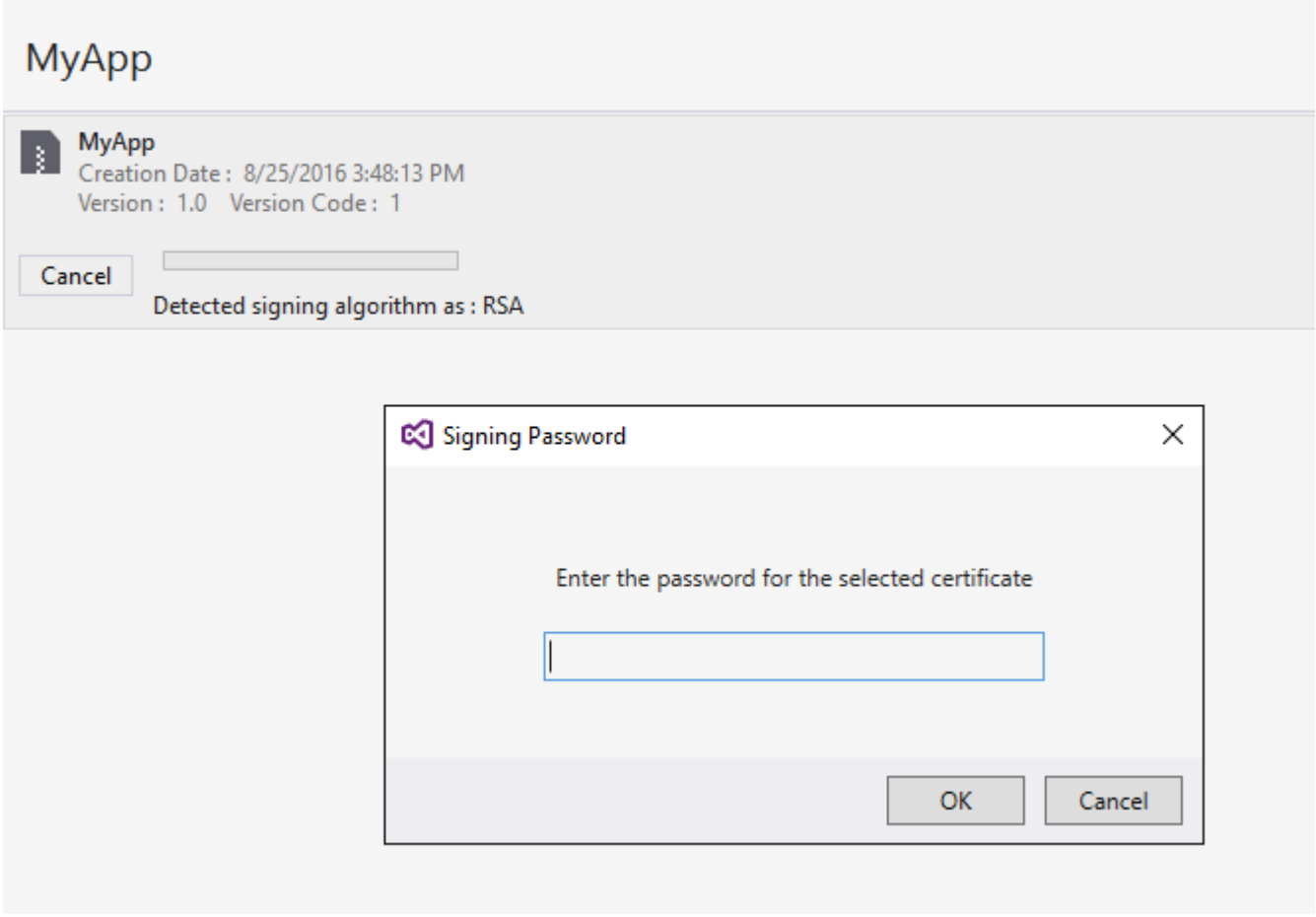
Creating a new Android Key Store screen:



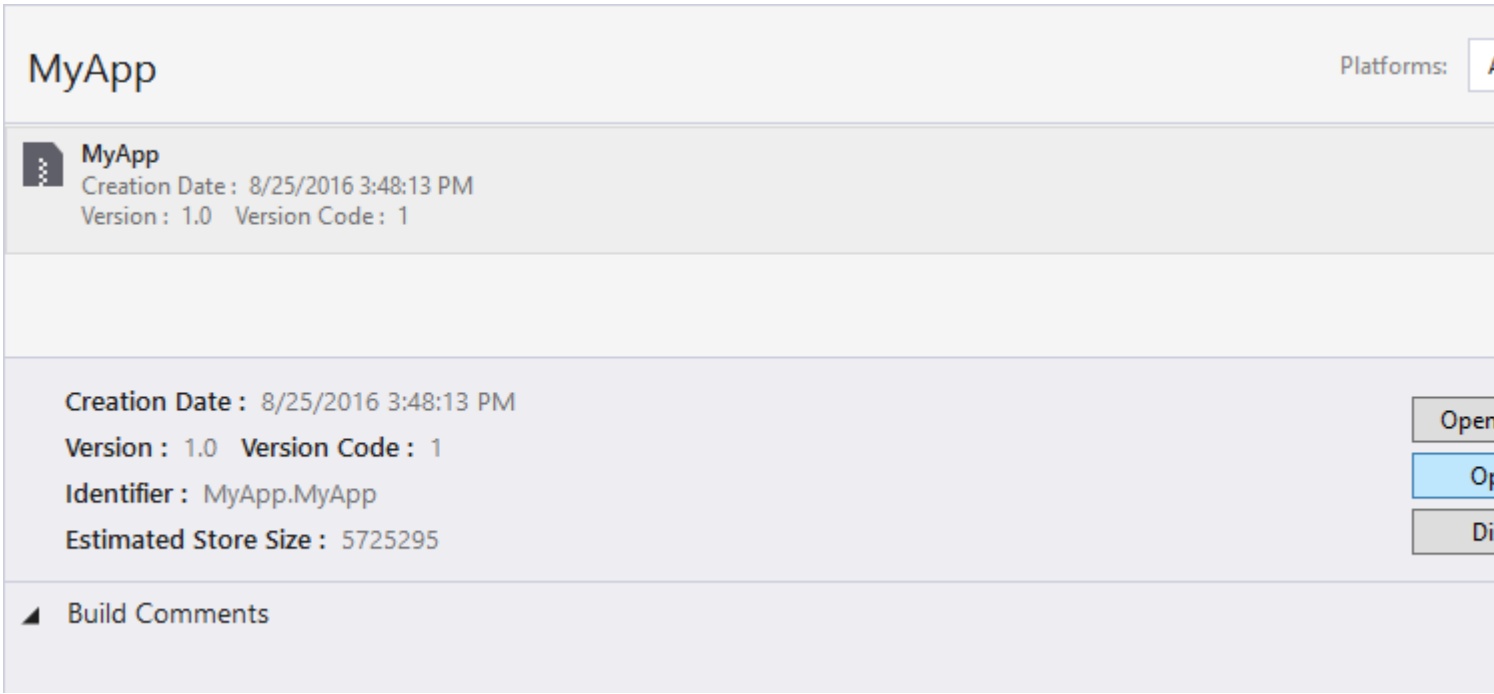
To create the APK, click in Save As. You may be prompted to type the Key Store password.

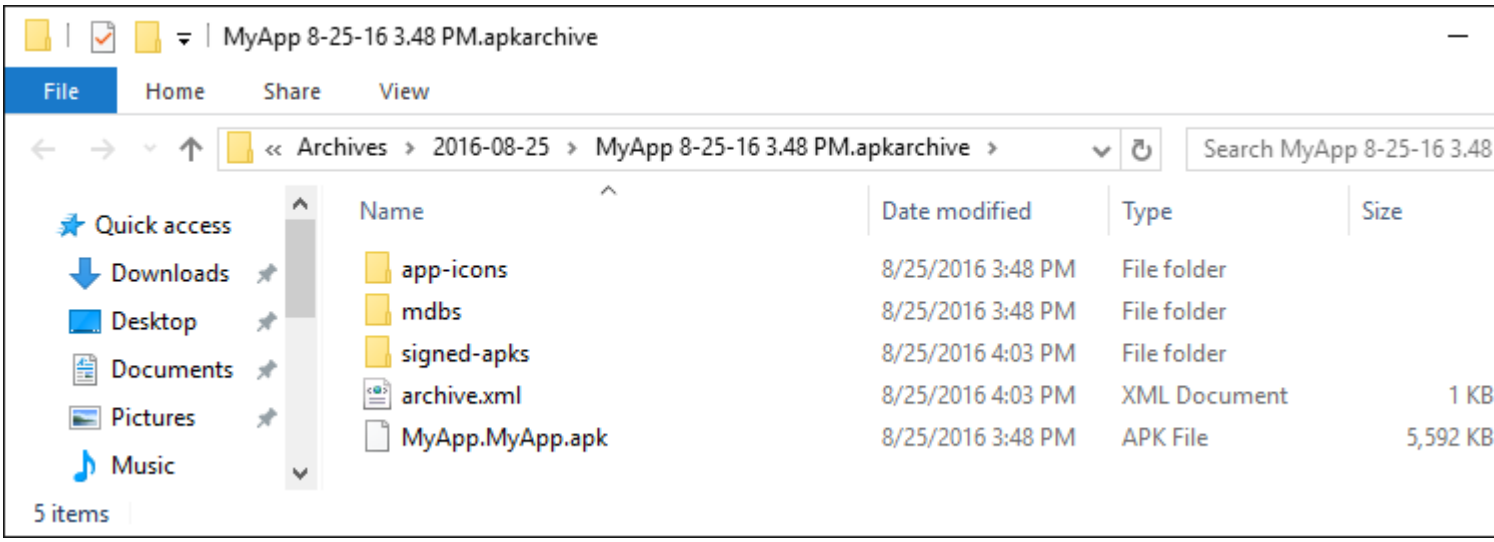

Distribute
App Details
MyApp
Creation Date: 8/25/2016
Version: 1.0
Select Channel
Ad Hoc
Signing Identity
Signing Identity
Search
Name
Expiration
chimp
Sat Aug 18 15:59:13 PDT 2046
Import...
Specify a Time Stamping Authority:
http://example.timestampauth.com
Why do I need a Key Store?
Back
Save As


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



When it completes, you can click in Open Folder on the Archives screen to see your generated APK file.





Enabling MultiDex in your Xamarin.Android APK

MultiDex is a library in the Android APK that allows the app to have more than 65,536 methods.

The Android APKs have Dalvik Executable files (.dex) that contain the generated bytecodes compiled from your Java code. Each .dex file can contain up to 65,536 methods (2^16).

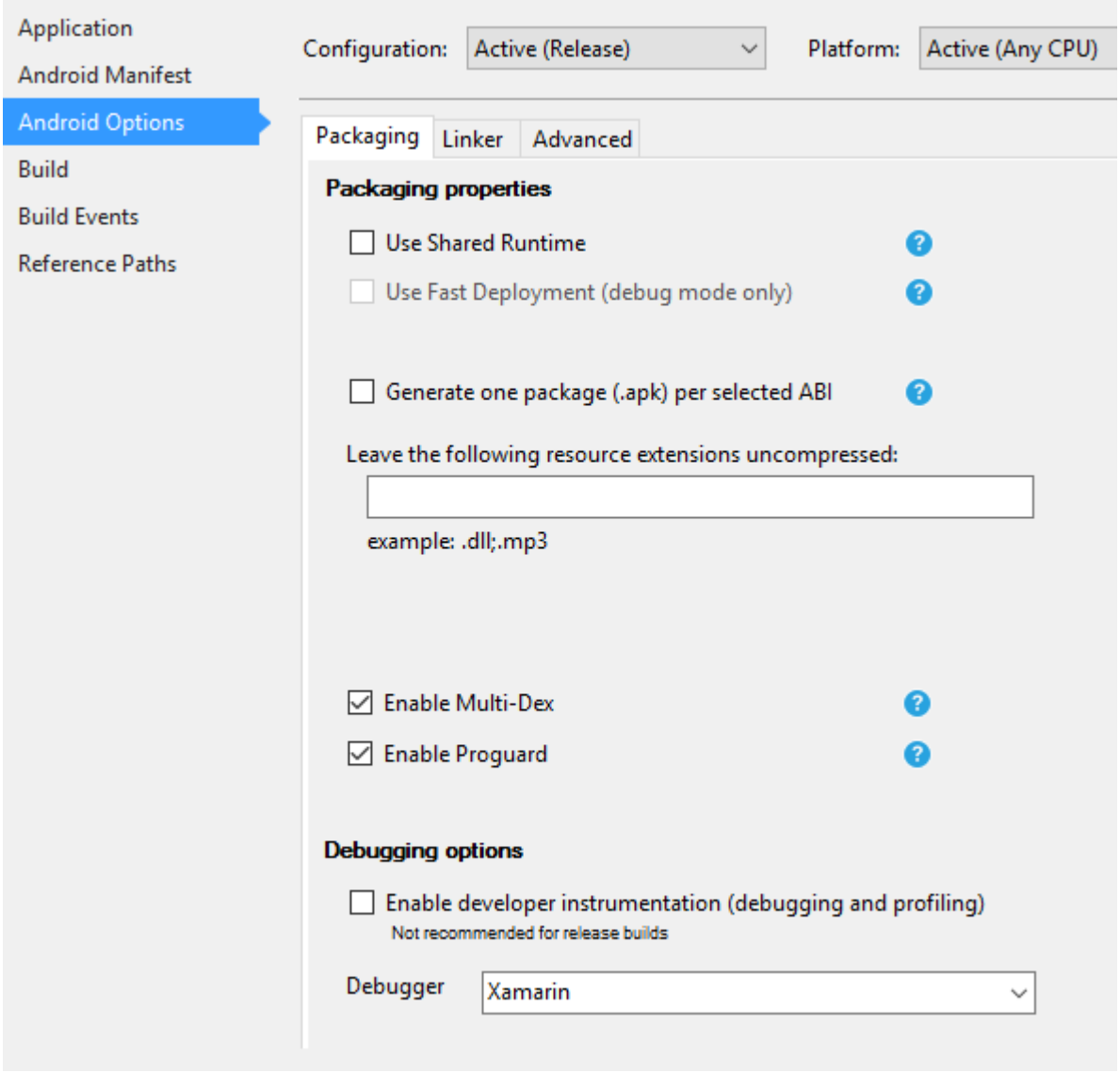
Android OS versions before Android 5.0 Lollipop (API 21) use the Dalvik runtime, which only supports one .dex file per APK, limiting to 65,536 methods per APK. Starting from Android 5.0, the Android OS use ART runtime, which can support more than one .dex file per APK, avoiding the limit.

To surpass the 65k methods limit in Android versions below API 21, the developers must use the MultiDex support library. The MultiDex creates extra classes.dex files (classes2.dex, classes3.dex, ...) referencing them in the classes.dex file. When the app starts loading, it uses an MultiDexApplication class to load the extra .dex files.

If your Android app aims for a minimum SDK version above or equal to API 21 (Android 5.0 Lollipop) it is not necessary to use the MultiDex library, because the OS handles natively the extra .dex files. However, if for compatibility reasons the developer wants to support older Android OS, then he/she should use the MultiDex library.

How to use MultiDex in your Xamarin.Android app

First, to enable MultiDex in your Xamarin.Android app, go to your project Properties -> Android Options -> Packaging -> Enable Multi-Dex, as in the print screen below:



Then, you must create a MultiDexApplication class in your app. In the project's root, create a new class (in the Solution Explorer, right-click in the project, Add.. -> Class, or Shift+Alt+C). In the new class file, copy the following code, replacing the namespace Sample with the name of your Xamarin.Android project namespace.

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

Code source here.

If you are developing in Visual Studio for Windows, there is also a bug in the Android SDK build tools that you need to fix in order to properly create the classes.dex files when building your project.

Go to your Android SDK folder, open the build-tools folder and there will be folders with the numbers of the Android SDK compilers, such as:

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

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

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

Inside each of those folders, there is a file called **mainClassesDex.bat**, a batch script used to create the classes.dex files. Open each mainClassesDex.bat file with a text editor (Notepad or Notepad++) and in its script, find and replace the block:

```
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
```

With the block:

```
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
```

Source here.

Save each mainClassesDex.bat in the text editor after changes.

After the steps above, you should be able to successfully build your Xamarin.Android app with MultiDex.

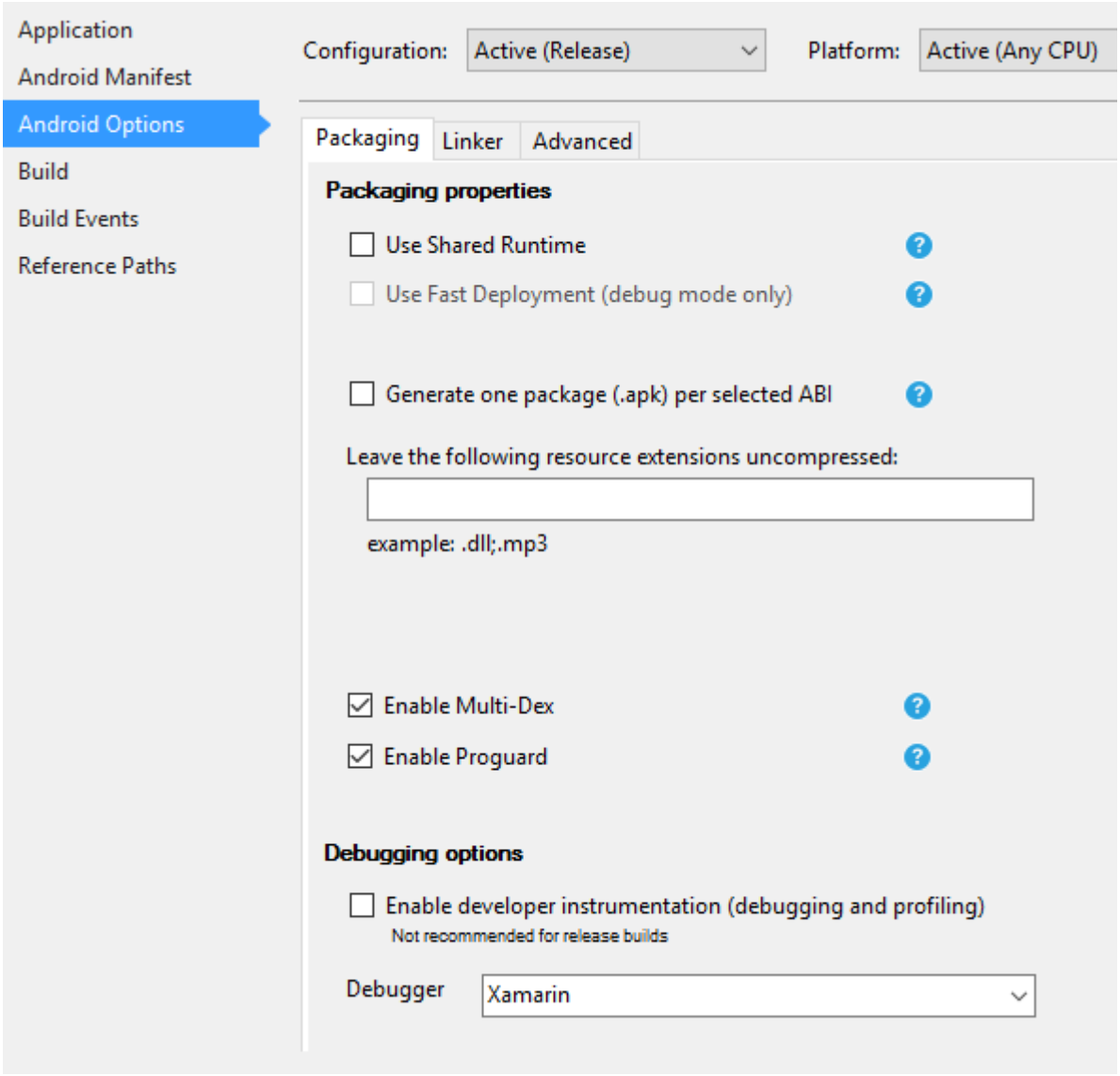
Enabling ProGuard in your Xamarin.Android APK

ProGuard is a tool used in the building process to optimize and obfuscate the Java code of your APK, and also remove unused classes. The resulting APK when using ProGuard will have a smaller size and will be harder to reverse-engineer (decompilation).

ProGuard can be used too in Xamarin.Android apps, and also will reduce the APK file size and obfuscate the Java code. Be aware, however, that the ProGuard obfuscation applies only to Java code. To obfuscate .NET code, the developer should use Dotfuscator or similar tools.

How to use ProGuard in your Xamarin.Android app

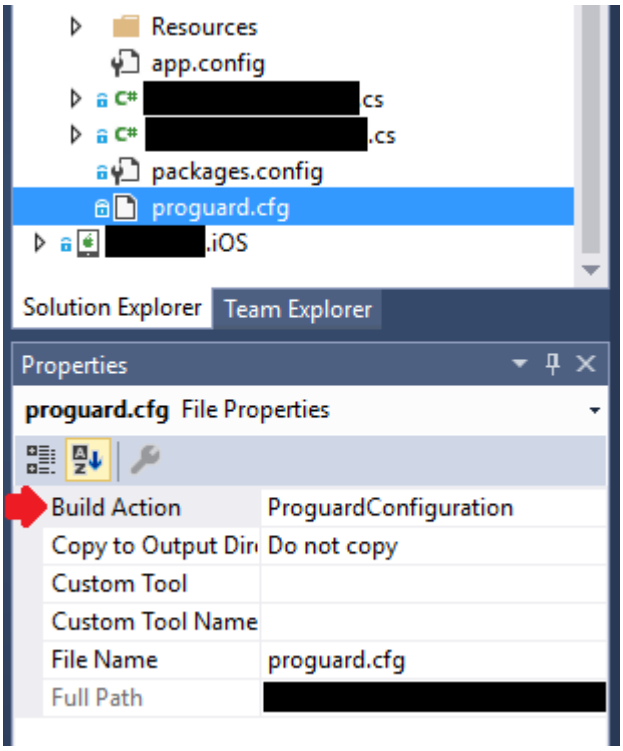
First, to enable ProGuard in your Xamarin.Android app, go to your project Properties -> Android Options -> Packaging -> Enable ProGuard, as in the print screen below:



This enables ProGuard when building your app.

Xamarin.Android, by default, sets its own configurations for ProGuard, that can be found inside the folders `obj/Debug/proguard` or `obj/Release/proguard`, in the files `proguard_project_primary.cfg`, `proguard_project_references.cfg` and `proguard_xamarin.cfg`. The three files are combined as configurations for ProGuard and they are automatically created by Xamarin when building.

If the developer wishes to further customize the ProGuard options, he/she can create a file in the project's root named `proguard.cfg` (other names are valid too, as long as the extension is .cfg) and setting its Build Action to ProguardConfiguration, as in the picture below:



In the file, custom ProGuard options can be inserted, such as `-dontwarn`, `-keep class` and others.

Important

As by now (April/2017), the Android SDK that is usually downloaded has an old version of ProGuard, which can cause errors when building the app using Java 1.8. When building, the Error List shows the following message:

```
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]
```

Source here.

To fix this problem, you must download the most recent version of ProGuard (here) and copy the contents of the .zip file to `android-sdk\tools\proguard\`. That will update the ProGuard and building process should run without problems.

After that, you should be able to successfully build your Xamarin.Android app with ProGuard.

"Mysterious" bugs related to ProGuard and Linker

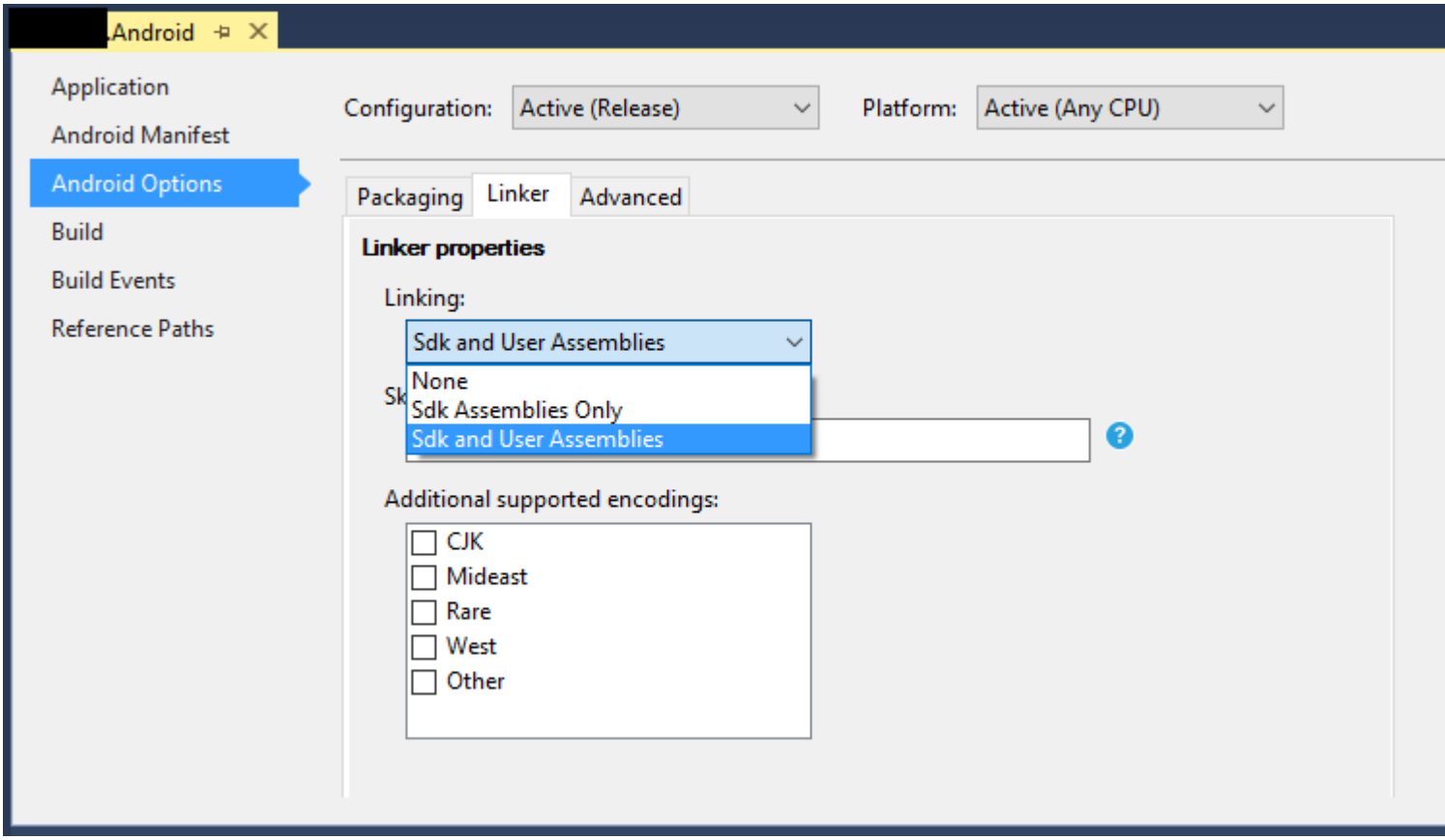
You made a great app and tested it in Debug, with good results. Everything was working fine!

But then, you decided to prepare your app for release. You set up MultiDex, ProGuard and Linker, and then, it stopped working.

This tutorial aims to help you to find out common problems related to ProGuard and Linker that can cause mysterious bugs.

Understanding Xamarin.Linker

Xamarin.Linker is a tool in the building process that removes unused code and classes **from your .NET code (not Java code)**. In your project's Properties -> Android Options -> Linker, there will be an selection box Linking with the options:



None: No code is removed.

Sdk Assemblies Only: This option makes the Xamarin.Linker to check for unused code only in the Xamarin libraries. **This option is safe.**

Sdk and User Assemblies: This option makes the Xamarin.Linker to check for unused code in the Xamarin libraries and in the project code (including PCLs, Xamarin components and NuGet packages). **This option is not always safe!**

When using Sdk and User Assemblies option, Xamarin.Linker may think that parts of the code are unused when actually they are very much used! That may cause some libraries to stop working properly and cause bugs in your app.

To make the Xamarin.Linker not remove code, there are 3 options:

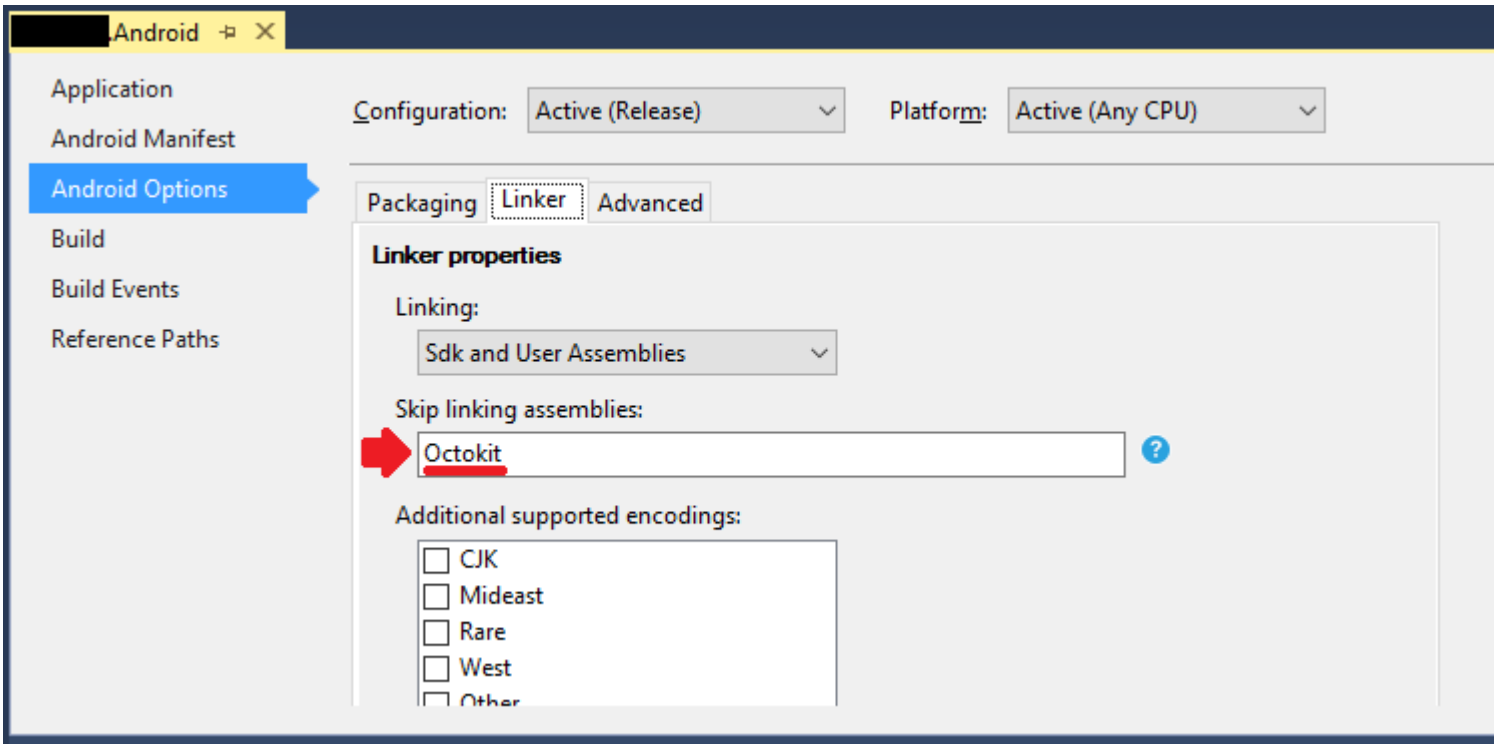
1. Setting the Linking option to None or Sdk Assemblies Only;
2. Skip linking assemblies;
3. Using the Preserve attribute.

Example for 2. Skip linking assemblies:

In the example below, using Xamarin.Linker caused a NuGet Package (Octokit) that works fine to stop working, because it could not connect to the internet anymore:

```
[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 ---
```

To make the library start working again, it was necessary to add the package reference name in the Skip linking assemblies field, located in project -> Properties -> Android Options -> Linker, as in the picture below:



After that, the library started to work without any issues.

Example for 3. Using the Preserve attribute:

Xamarin.Linker perceives as unused code mostly code from model classes in your project's core.

To make the class preserved during the linking process, you can use the Preserve attribute.

First, create in your project core's a class named **PreserveAttribute.cs**, insert the following code and replace the namespace with your project's namespace:

PreserveAttribute.cs:

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

In each model class of your project's core, insert the Preserve attribute as in the example below:

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;
        }
    }
}
```

After that, the linking process will not remove the preserved code anymore.

Understanding ProGuard

ProGuard is a tool in the building process that removes unused code and classes **from your Java code**. It also obfuscates and optimizes the code.

However, ProGuard sometimes may remove code that it perceives as unused, when it is not. To avoid that, the developer must debug the app (in Android Device Monitor and in the Visual Studio Debug) and detect which class was removed, for then to configure the ProGuard configuration file to keep the class.

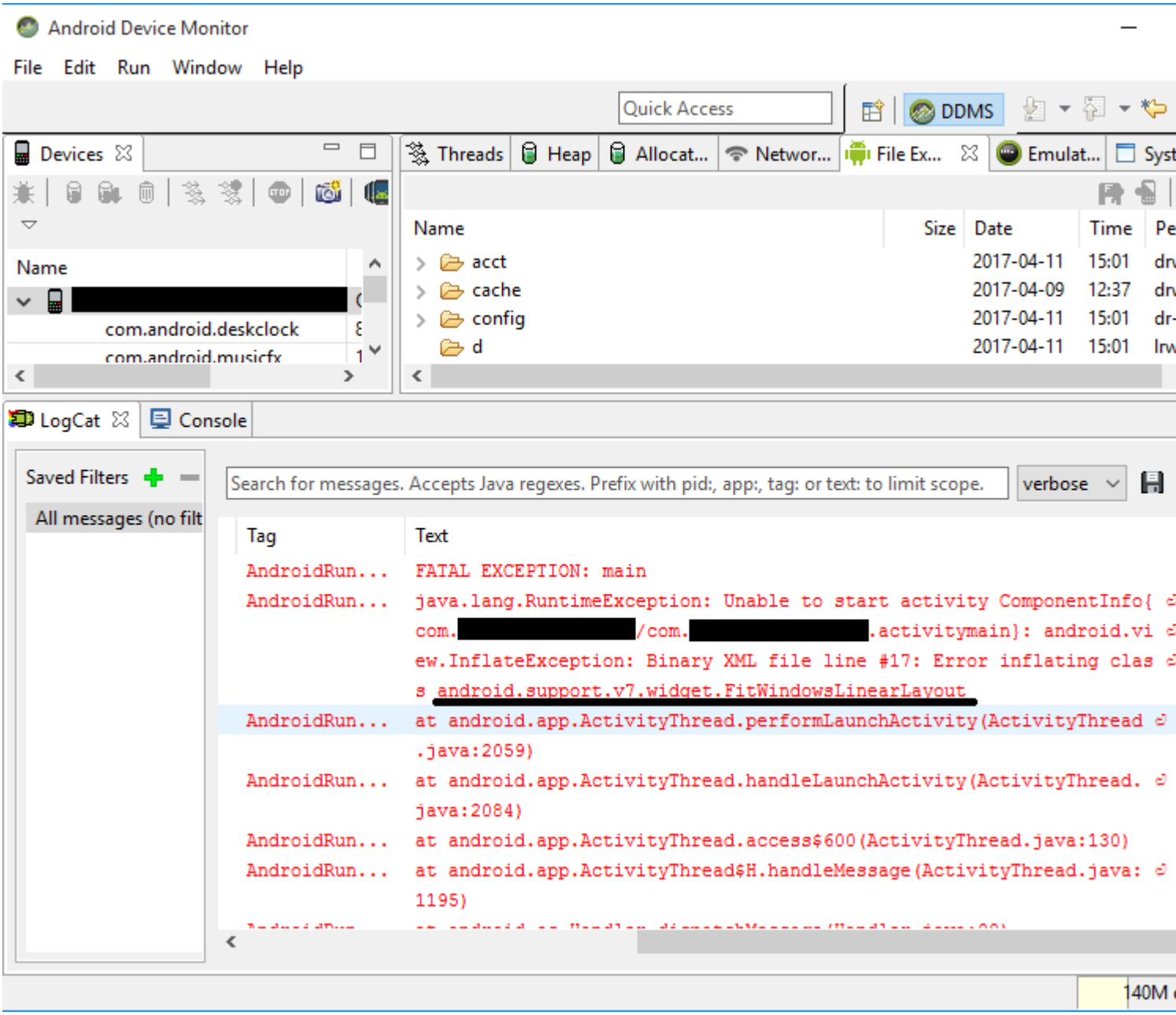
Example

In the example below, ProGuard removed two classes (Android.Support.V7.Widget.FitWindowsLinearLayout and Android.Support.Design.Widget.AppBarLayout) used in AXML layout files, but that were perceived as unused in the code. The removal caused ClassNotFoundException in the Java code when rendering the activity layout:

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 showing error when creating the layout in SetContentView:



To fix this error, it was necessary to add the following lines to the ProGuard configuration file of the project:

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

After that, no more errors were shown when creating the layout.

ProGuard Warnings

ProGuard sometimes show warnings in the Error List after building your project. Although they raise a question of whether your app is OK or not, not all of their warnings indicate troubles, especially if your app successfully builds.

One example for that is when using the Picasso library: when using ProGuard, this may show warnings such as `okio.Okio: can't find referenced class (...)` or `can't write resource [META-`

`INF/MANIFEST.MF] (Duplicate zip entry [okhttp.jar:META-INF/MANIFEST.MF]) (...)`, but the app builds and the library works without problems.

Read Publishing your Xamarin.Android APK online: https://riptutorial.com/xamarin-android/topic/9601/publishing-your-xamarin-android-apk

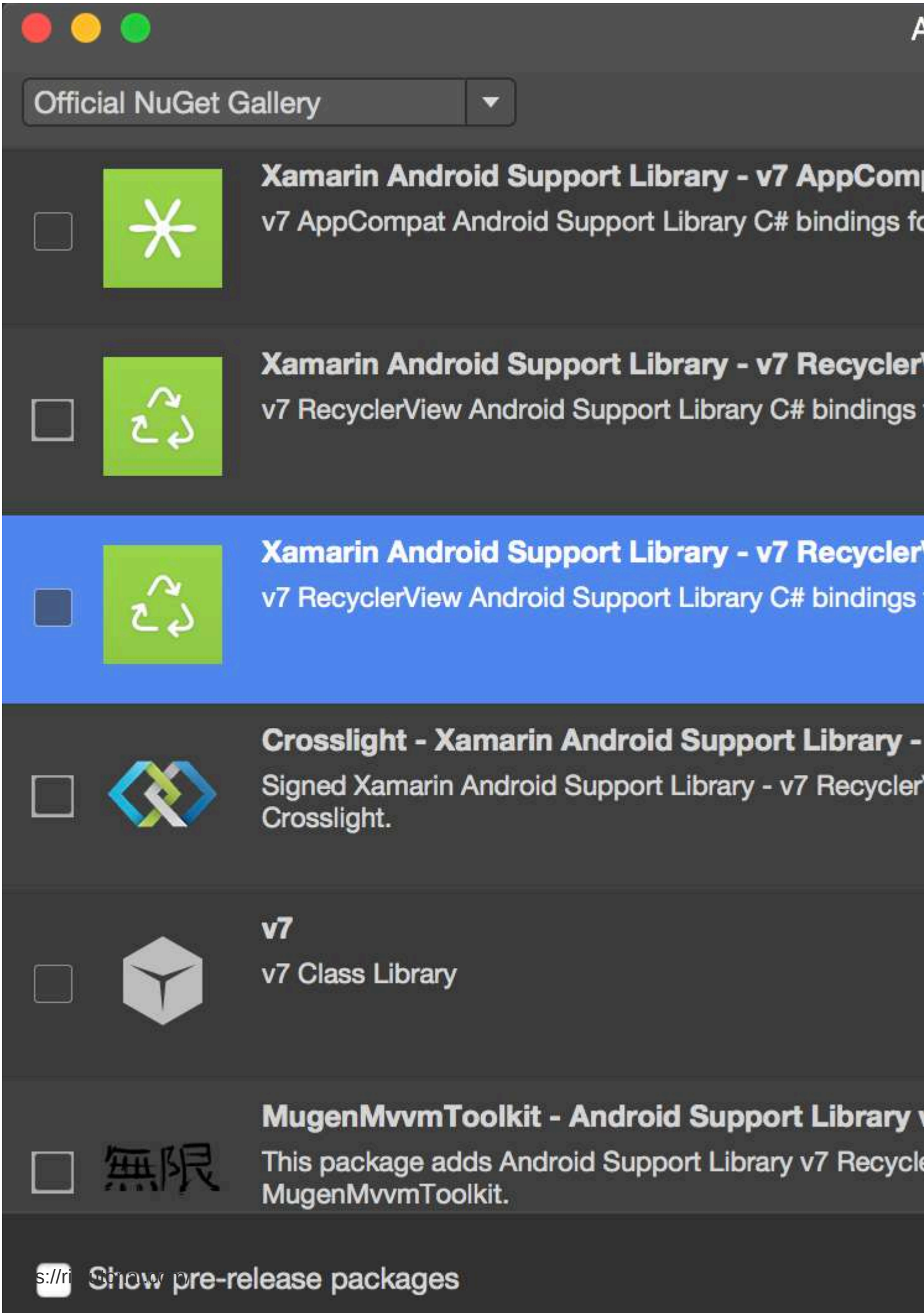
Chapter 10: RecyclerView

Examples

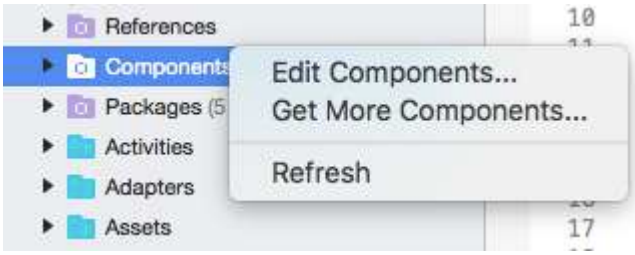
RecyclerView Basics

This is an example of using `Android Support Library V7 RecyclerView`. Support libraries are generally recommended because they provide backward-compatible versions of new features, provide useful UI elements that are not included in the framework, and provide a range of utilities that apps can draw on.

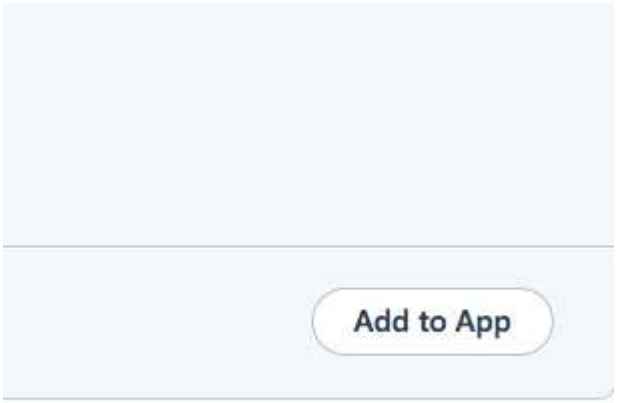
To get the `RecyclerView`, we will install the necessary Nuget packages. First, we will search for `v7 recyclerview`. Scroll down until we see `Xamarin Android Support Library – v7 RecyclerView`. Select it and click **Add Package**.


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.
無限
Show pre-release packages

is available as a Xamarin component. In order to add the component, right-click on `Components` within the Android project in Solution explorer and click on `Get More Components`.



Within the Component Store window that appears, search for RecyclerView. In the search list, select `Android Support Library V7 RecyclerView`. Then click on `Add to App`. The component gets added to the project.



Next step is to add the RecyclerView to a page. Within the `axml` (layout) file, we can add `RecyclerView` as below.

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

RecyclerView requires at least two helper classes to be set-up for basic standard implementation viz: `Adapter` and `ViewHolder`. `Adapter` inflates item layouts and binds data to views that are displayed within a RecyclerView. ViewHolder looks up and stores view references. The view holder also helps with detecting item-view clicks.

Here is a basic example of 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;
        }
    }
}
```

In the `OnCreateViewHolder` method we first inflate a View and create an instance of the ViewHolder class. This instance has to be returned. This method is invoked by the Adapter when it requires a new instance of ViewHolder. This method won't be invoked for every single cell. Once RecyclerView has enough cells to fill the View, it will re-use the old cells that is scrolled out of the View for further cells.

The `OnBindViewHolder` callback is invoked by Adapter to display the data at the specified position. This method should update the contents of the itemView to reflect the item at the given position.

Since the cell contains just a single `TextView`, we can have a simple ViewHolder as below.

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


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

Next step is to wire-up things 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;
}
```

LayoutManager class is responsible for measuring and positioning item views within a RecyclerView as well as determining the policy for when to recycle item views that are no longer visible to the user. Before the `RecyclerView`, we had to use `ListView` to arrange cells in a s in a vertically scrolling list and `GridView`to display items in a two-dimensional, scrollable grid. But now we can achieve both with RecyclerView by setting a different LayoutManger. `LinearLayoutManager` arranges cells as in a ListView and `GridLayoutManager` arranges cells Grid fashion.

RecyclerView with Click events

This example shows how to set Click EventHandlers in a Xamarin.Android RecyclerView.

In Android Java, the way to set up a listener for a Click is using a onClickListener for the view that will be clicked, like this:

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

However, in Xamarin.Android, the way to set up a listener for a Click event is by **adding** a EventHandler, in the following ways:

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
}
```

Note that **the EventHandler is added, not set.** If the Click EventHandler is added inside a GetView method from a GridView/ListView adapter, or a OnBindViewHolder method from a RecyclerView.Adapter, every time that the item view is created a new EventHandler will be added. After scrolling several times, multiple EventHandlers will be added, and when the view gets clicked, all of them will be fired.

To avoid this trouble, the EventHandlers must be unsubscribed and subscribed subsequently in the GetView or OnBindViewHolder methods. Also, they must use the number **3.** way to set the EventHandler, otherwise it will not be possible to unsubscribe the EventHandlers.

An example of an RecyclerView.Adapter with Click events is shown below:

```
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);
    }

}
```

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

Chapter 11: Toasts

Examples

Basic Toast Message

First, instantiate a Toast object with one of the `MakeText()` methods. This method takes three parameters: the application `Context`, the text message, and the duration for the toast. It returns a properly initialized Toast object. You can display the toast notification with `Show()`, as shown in the following example:

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

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

This example demonstrates everything you need for most toast notifications. You should rarely need anything else. You may, however, want to position the toast differently or even use your own layout instead of a simple text message. The following sections describe how you can do these things.

You can also chain your methods, call as a one-liner and avoid holding on to the Toast object, like this:

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

For more information refer to the more complete Android documentation on the topic.

Colored Toast Messages

Sometimes we want to give extra information to our user with colors (for example red means something wrong has happened) We can change toast message background color using setting a color filter to the view which our toast give us (here I use a 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();
```

And also we can change the text color if background is light or dark:

```
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
```

Change Toast Position

We can change our toast using SetGravity method. This method takes three parameters: first is gravity of toast on screen and two others set toast offset from the starting position (which is set by the first parameter):

```
//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();
```

Read Toasts online: https://riptutorial.com/xamarin-android/topic/3550/toasts

Chapter 12: Xamarin.Android - Bluetooth communication

Introduction

In **Xamarin.Android** the **BluetoothSocket.InputStream** and **BluetoothSocket.OutputStream** properties are by design automatically converted to **System.IO.Stream**. In case of so called interactive communication protocol, when server responds only when client talks to it, System.IO.Stream is not good because it has no method or property to get the number of available response bytes before reading the response.

Parameters

Parameter	Details
socket	An instance of BluetoothSocket object. Socket must be opened before call this method.
cmd	Command as a byte array to send to BT device.
_mx	Since this method uses a hardware resource, it is better to call it from a separate worker thread. This parameter is an instance of System.Threading.Mutex object and is used to synchronize the thread with other threads optionally calling this method.
timeOut	Wait time in milliseconds between Write and Read operations.

Examples

Send and receive data from and to bluetooth device using socket

The below example uses Android.Runtime.InputStreamInvoker and Android.Runtime.OutputStreamInvoker types obtain Java.IO.InputStream and Java.IO.OutputStream. Once we have a **Java.IO.InputStream** instance, we can use its **.Available()** method to get the number of available response bytes which we can use in **.Read()** method:

```
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;
}
```

Read Xamarin.Android - Bluetooth communication online: https://riptutorial.com/xamarin-android/topic/10844/xamarin-android---bluetooth-communication

Chapter 13: Xamarin.Android - How to create a toolbar

Remarks

Dear Team,

I think that its good to mention about official Android documentation where toolbar control is explained in details:

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

There is also interested content about Android.Support.v7 library used in the sample:

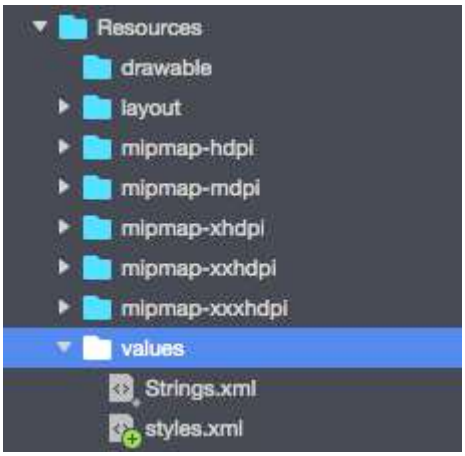
https://developer.android.com/training/appbar/index.html

Examples

Add toolbar to the Xamarin.Android application

Firstly you have to add Xamarin.Android.Support.V7.AppCompat library for NuGet:
https://www.nuget.org/packages/Xamarin.Android.Support.v7.AppCompat/

In the "values" folder under "Resources" add new xml file called "styles.xml":

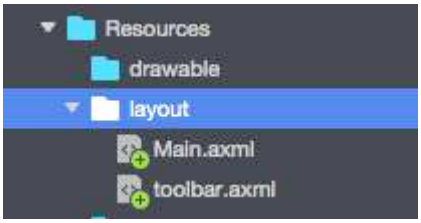


"styles.xml" file should contain below code:

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

Next step is to add "toolbar.axml" file that contains toolbar control definition to the "layout" folder:



Add below code to define toolbar:

```
<?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" />
```

Now please open "Main.axml" file and add below code just below closing tag for the first layout. Your code should look like below:

```
<?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>
```

Now you have to add information about theme that your app uses. Open "AndroidManifest" file and add theme information to the "application" tag:

```
<application android:theme="@style/MyTheme" android:allowBackup="true"
android:icon="@mipmap/icon" android:label="@string/app_name">
```

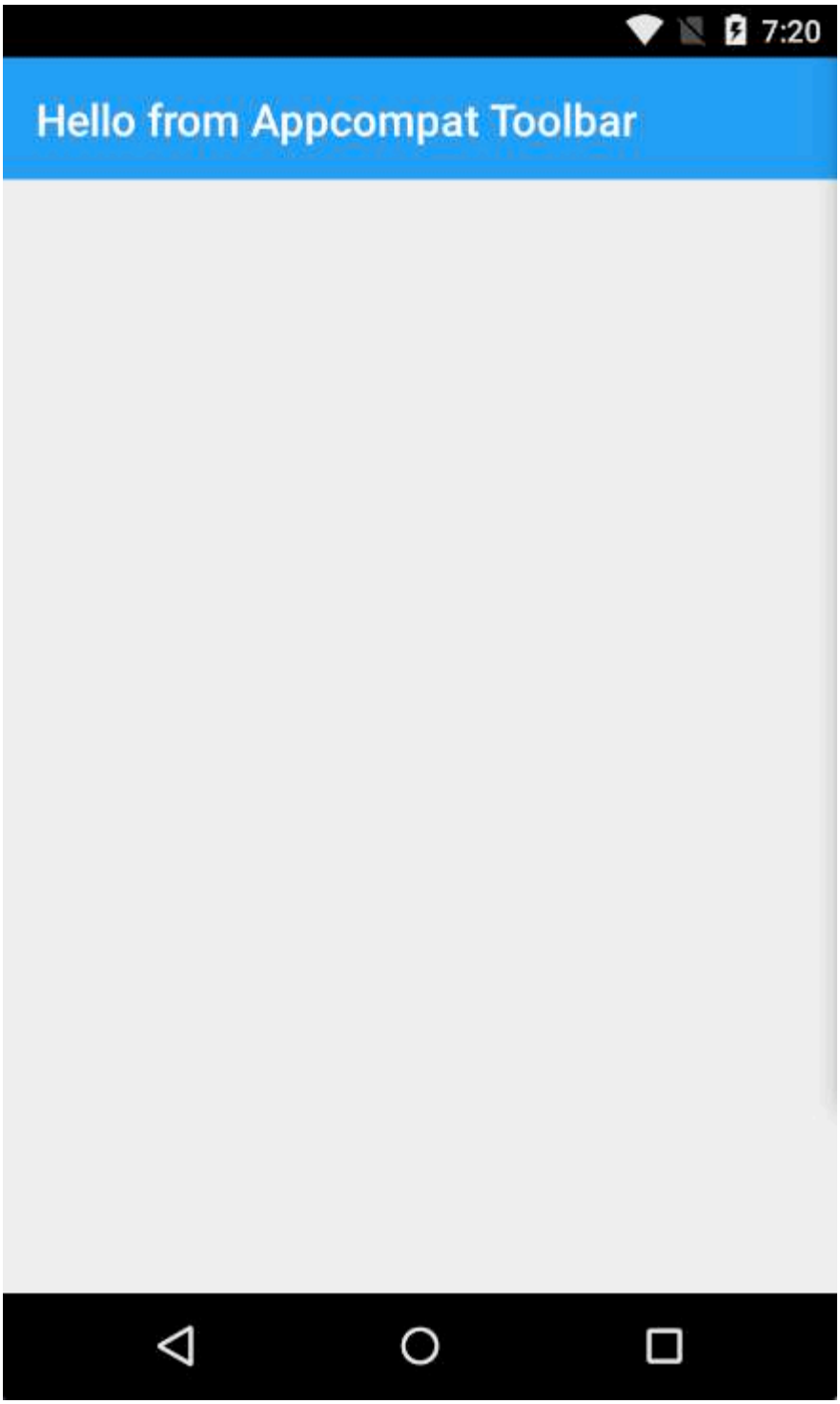
Last step is to connect the toolbar in Activity file. Open "MainActivity.cs" file. You have to change derivation from "Activity" to "AppCompatActivity". Now get reference to the toolbar and set it as default toolbar for the activity in the "OnCreate" method. You can also define title:

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

Whole method should look like below:

```
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";
    }
```

Rebuild project and launch it to see result:



Read Xamarin.Android - How to create a toolbar online: https://riptutorial.com/xamarin-android/topic/4755/xamarin-android---how-to-create-a-toolbar

Credits

S. No	Chapters	Contributors
1	Getting started with Xamarin.Android	Amy Burns, Community, Jon Douglas, Kevin Montrose, Ryan Weaver
2	App lifecycle - Xamarin.Andorid	CDrosos, Daniel Krzyczkowski, Steven Mark Ford
3	Barcode scanning using ZXing library in Xamarin Applications	GvSharma
4	Bindings	EJoshuaS, Jon Douglas, jonp, Matthew, Prashant C, Sven-Michael Stübe
5	Custom ListView	user3814750
6	Dialogs	JimBobBennett, Pilatus
7	How to correct the orientation of a picture captured from Android device	Daniel Krzyczkowski
8	Publishing your Xamarin.Android APK	Alexandre
9	RecyclerView	Alexandre, Matthew, Ryan Alford, Sreeraj, Zverev Eugene
10	Toasts	GONeale, Matthew, Piet, user2912553
11	Xamarin.Android - Bluetooth communication	Ladislav
12	Xamarin.Android - How to create a toolbar	Daniel Krzyczkowski, tylerjgarland