**Creating Xamarin.Android Binding Library**

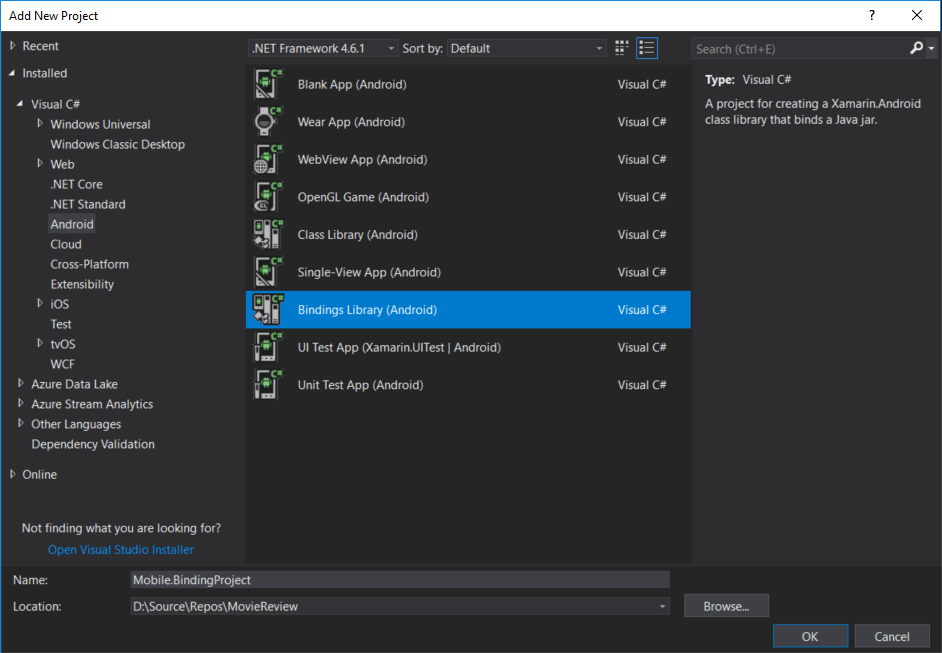
* **Posted:** January 22, 2018
* / **By:** Adam Pedley
* / **In:** [Native](https://xamarinhelp.com/category/native/)

To reference a Jar or Aar file in your Xamarin project, you can create a binding project to access the file. This is sometimes needed if you have code written in Java or an SDK that you need to access from Xamarin.

I will be doing a sample of how to bind the Google Play Services Wallet SDK. Please note that Xamarin have already done this for you, and you can just download from Nuget most of the Google Play Services Binding Libraries. This is just an example.

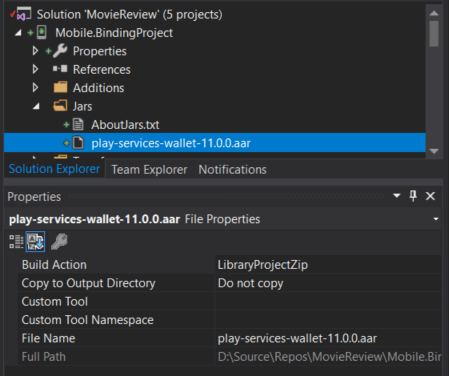
**Create Project**

First, create an Android Binding Library Project.



**Add Java File**

Next you add your Java file. If it is an ‘aar’ file, then set the build action to LibraryProjectZip. If it is a ‘jar’ file, set the build action to InputJar.



**Check References**

Most SDK’s will also reference other Java files. So you need to look for a **\*.pom** file with your java file. In here, it will contain what references are required. For example, the Wallet service shows.

<project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://maven.apache.org/POM/4.0.0" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.google.android.gms</groupId>

<artifactId>play-services-wallet</artifactId>

<version>11.0.0</version>

<packaging>aar</packaging>

<dependencies>

<dependency>

<groupId>com.google.android.gms</groupId>

<artifactId>**play-services-base**</artifactId>

<version>11.0.0</version>

<scope>compile</scope>

<type>aar</type>

</dependency>

<dependency>

<groupId>com.google.android.gms</groupId>

<artifactId>**play-services-basement**</artifactId>

<version>11.0.0</version>

<scope>compile</scope>

<type>aar</type>

</dependency>

<dependency>

<groupId>com.google.android.gms</groupId>

<artifactId>**play-services-identity**</artifactId>

<version>11.0.0</version>

<scope>compile</scope>

<type>aar</type>

</dependency>

<dependency>

<groupId>com.google.android.gms</groupId>

<artifactId>**play-services-maps**</artifactId>

<version>11.0.0</version>

<scope>compile</scope>

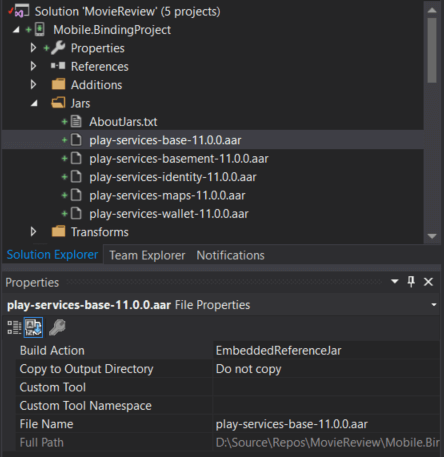
<type>aar</type>

</dependency>

</dependencies>

</project>

As you can see here, it contains 4 references. We now have to get the aar file for each of these references. Add them to your project, and set the build action to **EmbeddedReferenceJar**.

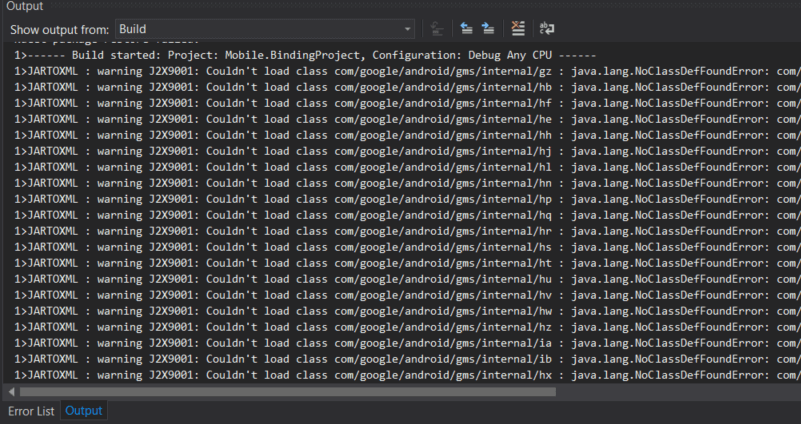


Please note, that you will also have to look at the \*.pom file for each reference, and also add any of those references, and so forth. You don’t have to add any of the Android Support Libraries however, as they are added during the build.

**Build Output**

Now we get to build. There may be errors that appear, which we will cover shortly, or it may build successfully. But, a successful build, doesn’t mean it worked successfully either. To ensure it did work, you have to go to the Build Output window and see what is happening.

If you forgot to add certain references, you are very likely to get a build output, looking like this.



This type of error message is basically saying that something isn’t referenced, but it doesn’t give you much of hint as to what it is.

1>JARTOXML : warning J2X9001: Couldn't load class com/google/android/gms/internal/hf : java.lang.NoClassDefFoundError: com/google/android/gms/internal/zzed

Double check all references are added, and they have their Build Action set to EmbeddedReferenceJar.

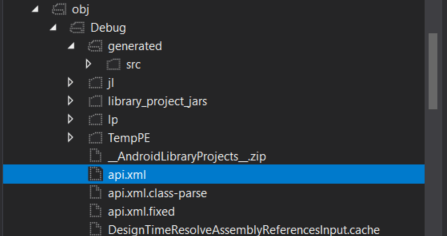
**Special Note:** I also discovered you can get this error if the Class Parser isn’t set to class-parse. You can set this in the properties of the Binding project, or you can add this to your csproj, just under the line *<AndroidCodegenTarget>XamarinAndroid</AndroidCodegenTarget>.*

<AndroidClassParser>class-parse</AndroidClassParser>

I had a strange issue, where I set it in properties, but it didn’t save it to the csproj. So take the extra step and look inside the csproj to make sure the line is there.

**Check API**

While there may be some warnings or errors, you can check out the api.xml file that is located in obj > Debug > api.xml.



This file contains the APIs it is trying to make public, and it can be used to debug any errors that are preventing you from building.

**Transformations**

Looking at the API, gives you great context, to be able to make transformations. Sometimes a field name might be the same as the class or interface it is contained in. While legal for Java, it is illegal for C#. With this you can go to the Transformations folder and open the Metadata.xml file. It has some examples of removing nodes. You can also add and rename nodes if desired.

<remove-node path="/api/package[@name='com.google.android.gms.wallet']/interface[@name='Wallet']/field[@name='Wallet']" />

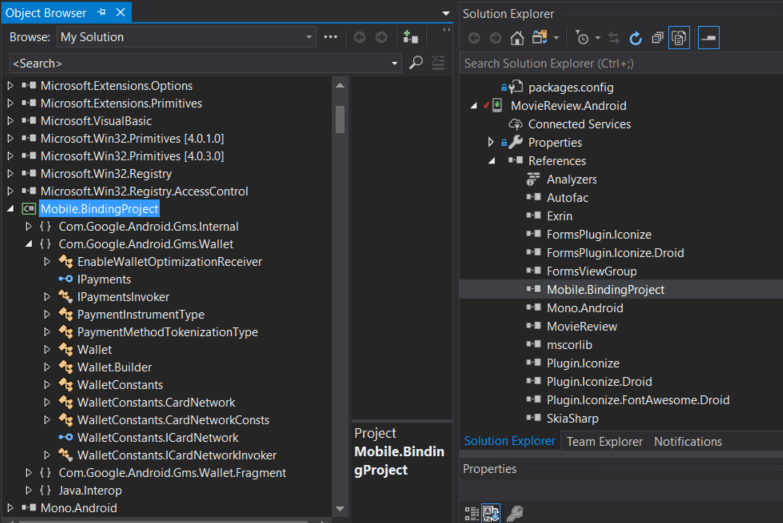
It’s easy to modify. There are different nodes you can choose, classes, interfaces, fields, methods etc. Just do /node[@name=’TheName’]/ etc. Just follow the structure of the xml file to reach the node, you will see they match 1 to 1.

**Using Binding**

Once you have a successfully building binding project, you can reference it just like any other project, to your Xamarin.Android project. You will notice that lowercase names have been switched to PascalCasing.

Com.Google.Android.Gms.Wallet

If you want to make sure the binding was successful. When you reference a project, you can double click the reference to open up the Object Browser, and view the classes and methods available.



**More Information**

If you want a more detailed look at some parts of the binding process, Jon Douglas has a guide [Approaching a Xamarin.Android Bindings Case](https://gist.github.com/JonDouglas/dda6d8ace7d071b0e8cb) that contains more information, right from the developer who helped build the binding engine.