Creating and Publishing an Android Library

by Ashraff Hathibelagal 2 Sep 2015 Difficulty: Intermediate Length: Medium Languages: English

Android Android SDK Android Studio IDEs Development Practices Open Source Mobile Development

Introduction

Our lives as Android developers would be a lot harder if not for all those third-party libraries out there that we love to include in our projects. In this tutorial, you will learn how to give back to the developer community by creating and publishing your own Android libraries, which people can effortlessly add and use in their projects.

1. Creating an Android Library

If your library is going to be composed of only Java classes, packaging it as a JAR and distributing it using a file host is perhaps the quickest and easiest way to share it. If you were to create it from the console, the following command would suffice:

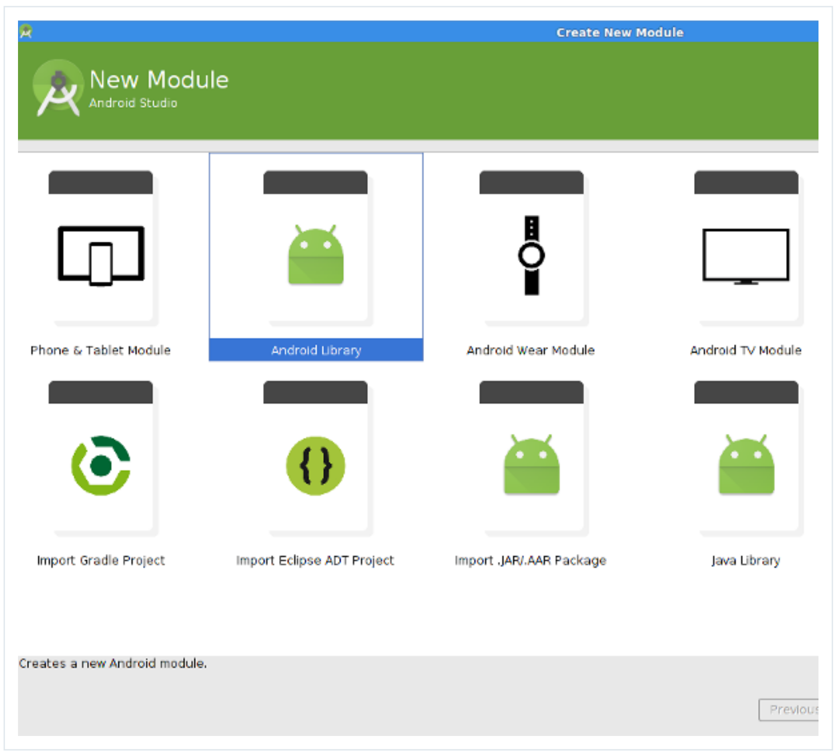
```
jar cvf mylibrary.jar Class1.class Class2.class ... ClassN.class
```

This tutorial however, shows you how to work with more complex libraries that contain not just Java classes, but also various types of XML files and resources. Such libraries are created as Android library modules and are usually packaged as **AAR** files.

Let's create a simple Android library that offers a custom `View` to developers who use it.

Step 1: Add a New Module

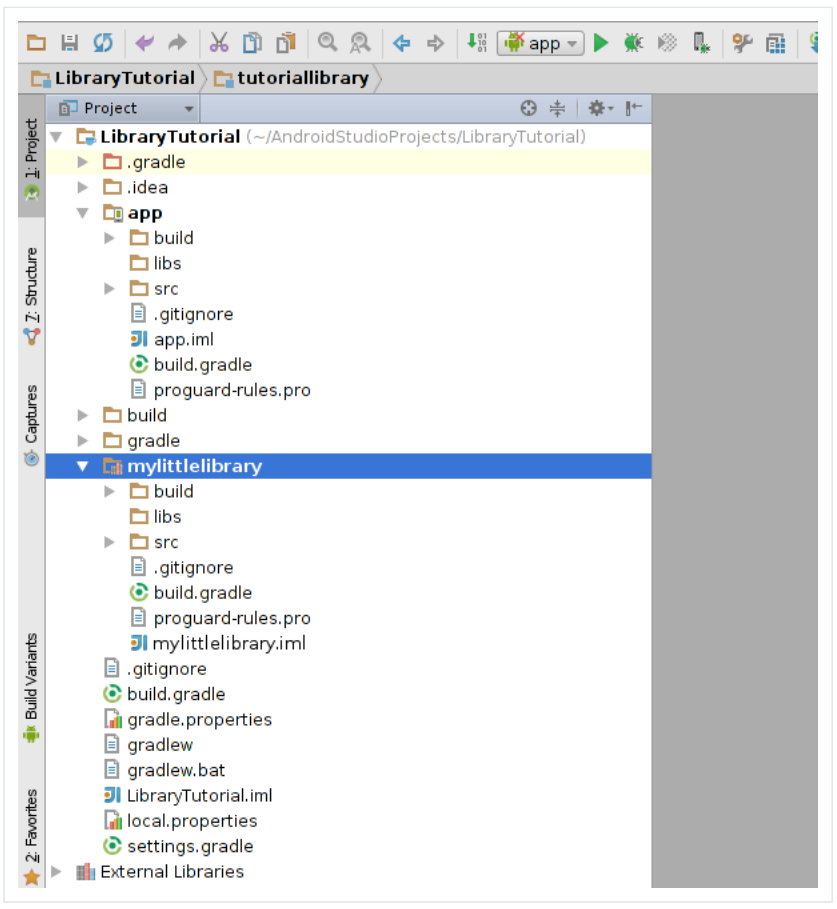
To begin, add a new Android module to your project by selecting **New > New Module** from the **File** menu. You will be shown the following screen, which offers lots of choices:



Select **Android Library** and press **Next**. In the form that follows, enter a name for your library and press **Next**. I'll be calling this library **mylittlelibrary**.

In the last screen, select **Add no Activity** and press **Finish**.

Your project will now have two modules, one for the app and one for the library. Here's what its structure looks like:



```
LibraryTutorial 〉 tutoriallibrary 〉
Project
▼ LibraryTutorial (~/AndroidStudioProjects/LibraryTutorial)
  ▶ .gradle
  ▶ .idea
  ▼ app
    ▶ build
      libs
    ▶ src
      .gitignore
      app.iml
      build.gradle
      proguard-rules.pro
  ▶ build
  ▶ gradle
  ▼ mylittlelibrary
    ▶ build
      libs
    ▶ src
      .gitignore
      build.gradle
      proguard-rules.pro
      mylittlelibrary.iml
    .gitignore
    build.gradle
    gradle.properties
    gradlew
    gradlew.bat
    LibraryTutorial.iml
    local.properties
    settings.gradle
▶ External Libraries
```

Step 2: Create a Layout

Create a new layout XML by right-clicking on the **res** folder of your library module and selecting **New > XML > Layout XML File**. Name it **my_view.xml**.

To keep this tutorial simple, we'll be creating a custom `View` that has two `TextView` widgets inside a `LinearLayout`. After adding some text to the `TextView` widgets, the layout XML file should look like this:

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/andrc
    android:layout_width="match_parent" android:layout_height="match_
    android:orientation="vertical"
    android:padding="16dp">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Hello!"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="This is my custom view from my little library."
</LinearLayout>
```

Step 3: Create a Java Class

Create a new Java class and name it **MyView.java**. Make sure to put this file in the **src** directory of the library module–not the app module.

To make this class behave as a `View`, make it a subclass of the `LinearLayout` class. Android Studio will prompt you to add a few constructors to the class. After adding them, the new class should look like this:

```java
public class MyView extends LinearLayout {

    public MyView(Context context) {
        super(context);
    }

    public MyView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

}
```

As you can see, we now have two constructors. To avoid adding initialization code to each constructor, call a method named **initialize** from each constructor. Add the following code to each constructor:

```java
initialize(context);
```

In the `initialize` method, call `inflate` to associate the layout we created in the previous step with the class.

```java
private void initialize(Context context){
    inflate(context, R.layout.my_view, this);
}
```

2. Using the Library Locally

Now that the library is ready, let's make use of it in the **app** module of the same project in order to make sure that there are no issues. To do so, add it as a `compile` dependency in the **build.gradle** file of the app module:

```
compile project(":mylittlelibrary")
```

Create a new Java class, **MainActivity**, inside the app module. Make it a subclass of the `Activity` class and override its `onCreate` method.

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

Inside the `onCreate` method, create an instance of the custom view using its constructor. Pass it to the `setContentView` method so that it fills all the screen space of the `Activity`:

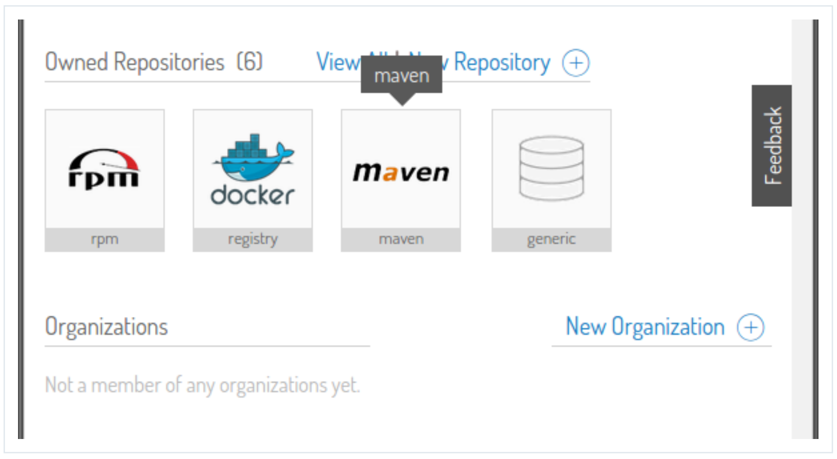
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    View v = new MyView(this);
    setContentView(v);

}
```

Your `Activity` is now ready. After adding it to the app manifest, build your project and deploy your app to an Android device. You should be able to see the custom view when the app starts.

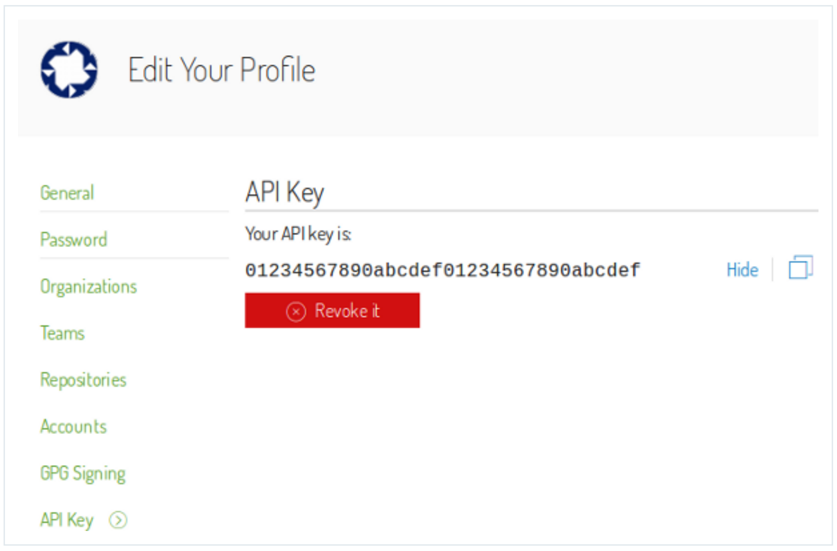
3. Publishing Your Library on Bintray

Bintray is a popular platform you can use to publish Android libraries. It is free and easy to use.

Start by creating an account on Bintray. After signing in to your account, you will see that you already own six repositories. You can either use one of them or create a new repository. For this tutorial, I will be using the repository called **maven**, which is a Maven repository.



Visit your profile page and click the **Edit** button. On the next page, click the **API Key** link to view your API key.



API Key

Your API key is:

01234567890abcdef01234567890abcdef

Make a note of the key, because you will be needing it to authenticate yourself when using the Bintray plugin.

Step 1: Add Necessary Plugins

To interact with Bintray in Android Studio, you should include the Bintray plugin in the `dependencies` of your project's **build.gradle** file.

```
classpath 'com.jfrog.bintray.gradle:gradle-bintray-plugin:1.2'
```

Because you will be uploading the library to a Maven repository, you should also add the Maven plugin as shown below.

```
classpath "com.github.dcendents:android-maven-gradle-plugin:1.3"
```

Step 2: Apply the Plugins

Open the **build.gradle** file of your library module and add the following code to apply the plugins we added in the previous step.

```
apply plugin: 'com.jfrog.bintray'
apply plugin: 'com.github.dcendents.android-maven'
```

Step 3: Specify POM Details

The Bintray plugin will look for a POM file when it uploads the library. Even though the Maven plugin generates it for you, you should specify the value of the `groupId` tag and the value of the `version` tag yourself. To do so, use the `group` and `version` variables in your gradle file.

```
group = 'com.github.hathibelagal.librarytutorial' // Change this to mc
version = '1.0.1' // Change this to match your version number
```

If you are familiar with Maven and you are wondering why we didn't specify the value of the `artifactId` tag, it is because the Maven plugin will, by default, use the name of your library as the `artifactId`.

Step 4: Generate a Sources JAR

To conform to the Maven standards, your library should also have a JAR file containing the library's source files. To generate the JAR file, create a new `Jar` task, **generateSourcesJar**, and specify the location of the source files using the `from` function.

```
task generateSourcesJar(type: Jar) {
    from android.sourceSets.main.java.srcDirs
    classifier 'sources'
}
```

Step 5: Generate a Javadoc JAR

It is also recommended that your library has a JAR file containing its Javadocs. Because you currently don't have any Javadocs, create a new `Javadoc` task, **generateJavadocs**, to generate them. Use the `source` variable to specify the location of the source files. You should also update the `classpath` variable so that the task can find classes that belong to the Android SDK. You can do this by adding the return value of the `android.getBootClasspath` method to it.

```
task generateJavadocs(type: Javadoc) {
    source = android.sourceSets.main.java.srcDirs
    classpath += project.files(android.getBootClasspath()
            .join(File.pathSeparator))
}
```

Next, to generate a JAR from the Javadocs, create a `Jar` task, **generateJavadocsJar**, and pass the `destinationDir` property of `generateJavadocs` to its `from` function. Your new task should look like this:

```
task generateJavadocsJar(type: Jar) {
    from generateJavadocs.destinationDir
    classifier 'javadoc'
}
```

To make sure the `generateJavadocsJar` task only starts when the `generateJavadocs` task has completed, add the following code snippet, which uses the `dependsOn` method to order the tasks:

```
generateJavadocsJar.dependsOn generateJavadocs
```

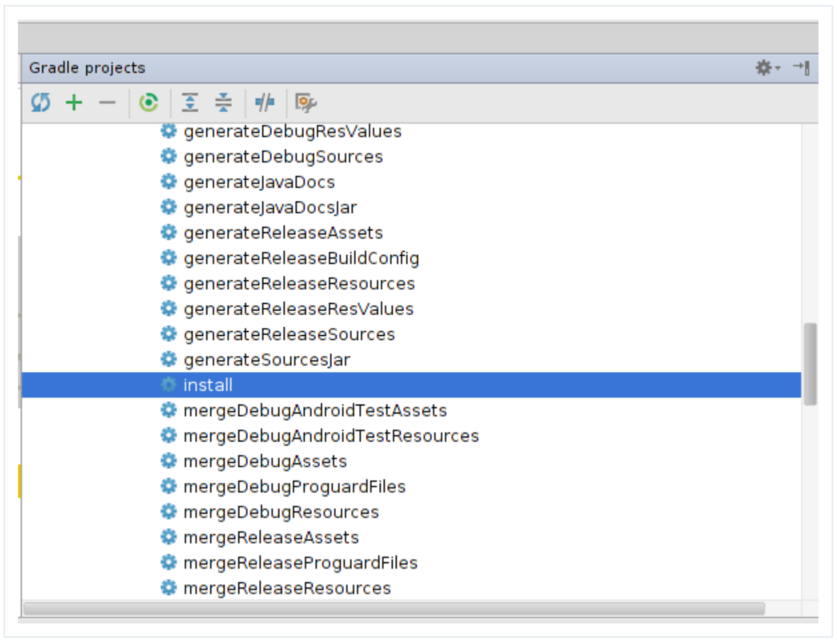
Step 6: Include the Generated JAR files

To include the source and Javadoc JAR files in the list of artifacts, which will be uploaded to the Maven repository, you should add the names of their tasks to a `configuration` called **archives**. To do so, use the following code snippet:

```
artifacts {
    archives generateJavaDocsJar
    archives generateSourcesJar
}
```

Step 7: Run Tasks

It is now time to run the tasks we created in the previous steps. Open the **Gradle Projects** window and search for a task named **install**.



generateDebugResValues
generateDebugSources
generateJavaDocs
generateJavaDocsJar
generateReleaseAssets
generateReleaseBuildConfig
generateReleaseResources
generateReleaseResValues
generateReleaseSources
generateSourcesJar
install
mergeDebugAndroidTestAssets
mergeDebugAndroidTestResources
mergeDebugAssets
mergeDebugProguardFiles
mergeDebugResources
mergeReleaseAssets
mergeReleaseProguardFiles
mergeReleaseResources

Double-click it to run the tasks associated with the library module. Once it's finished running, you will have everything you need to publish your library, a valid POM file, an AAR file, a sources JAR, and a Javadocs JAR.

Step 8: Configure the Bintray Plugin

To configure the plugin, you should use the `bintray` closure in your Gradle file. First, authenticate yourself using the `user` and `key` variables, corresponding to your Bintray username and API key respectively.

On Bintray, your library will reside inside a **Bintray package**. You should provide details about it using the intuitively named `repo`, `name`, `licenses`, and `vcsUrl` parameters of the `pkg` closure. If the package doesn't exist, it will be created automatically for you.

When you upload files to Bintray, they will be associated with a version of the Bintray package. Therefore, `pkg` must contain a `version` closure whose `name` property is set to a unique name. Optionally, you can also provide a description, release date, and Git tag using the `desc`, `released`, and `vcsTag` parameters.

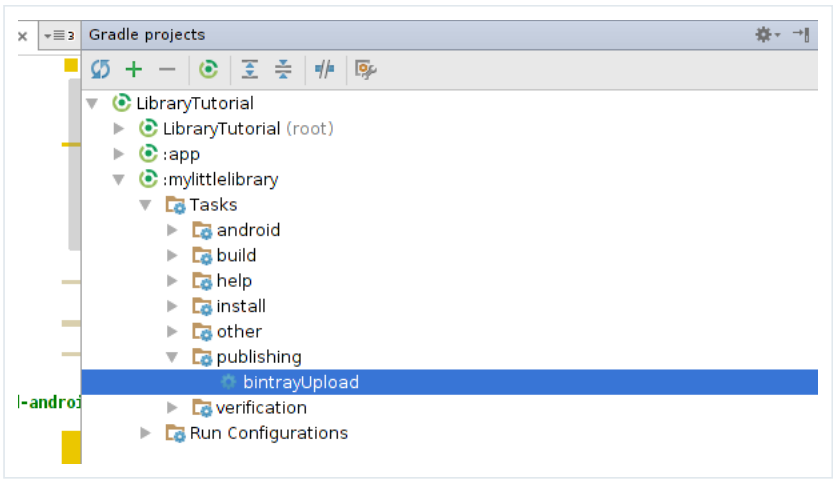
Finally, to specify the files that should be uploaded, set the value of the `configuration` parameter to **archives**.

This is a sample configuration:

```
bintray {
    user = 'test-user'
    key = '01234567890abcdef01234567890abcdef'
    pkg {
        repo = 'maven'
        name = 'com.github.hathibelagal.mylittlelibrary'

        version {
            name = '1.0.1-tuts'
            desc = 'My test upload'
            released  = new Date()
            vcsTag = '1.0.1'
        }

        licenses = ['Apache-2.0']
        vcsUrl = 'https://github.com/hathibelagal/LibraryTutorial.git
        websiteUrl = 'https://github.com/hathibelagal/LibraryTutorial
    }
    configurations = ['archives']
}
```

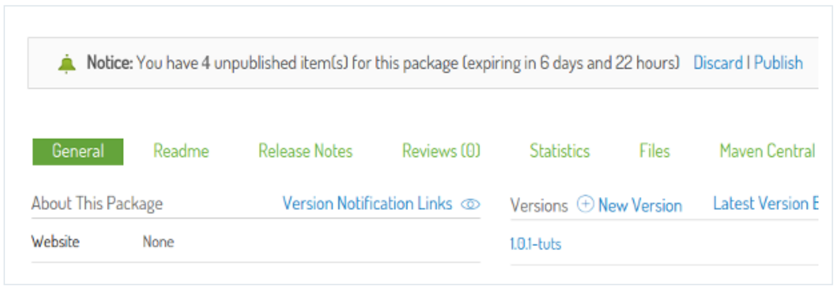
Step 9: Upload Files Using the Bintray Plugin

Open the **Gradle Projects** window again and search for the **bintrayUpload** task. Double-click it to begin uploading the files.



LibraryTutorial
LibraryTutorial (root)
:app
:mylittlelibrary
Tasks
android
build
help
install
other
publishing
bintrayUpload
verification
Run Configurations

Once the task completes, open a browser to visit your Bintray package's details page. You will see a notification that says that you have four unpublished files. To publish these files, click the **Publish** link.



4. Using the Library From Bintray

Your library is now available as a Bintray package. Once you share the URL of your Maven repository, along with the group ID, artifact ID, and version number, any developer can access your library. For example, to use the library we created, developers would have to include the following code snippet:

```
repositories {
    maven {
        url 'https://dl.bintray.com/eruzza/maven'
    }
}

dependencies {
    compile 'com.github.hathibelagal.librarytutorial:mylittlelibrary:1
}
```

Note that the developer has to explicitly include your repository in the list of `repositories` before adding the library as a `compile` dependency.

KBC

Experience the Bank of You

Find out more ›

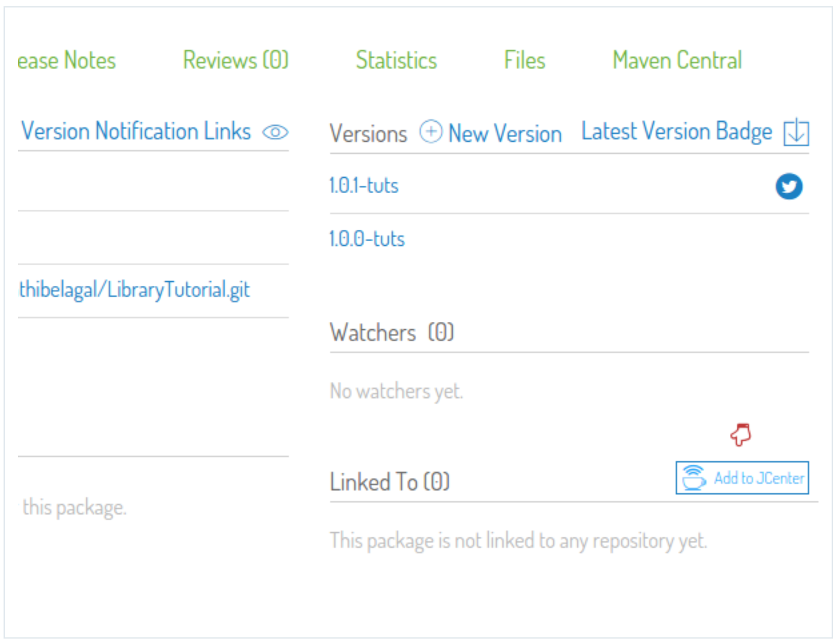
KBC Bank Ireland plc is regulated by the Central Bank of Ireland.

Advertisement

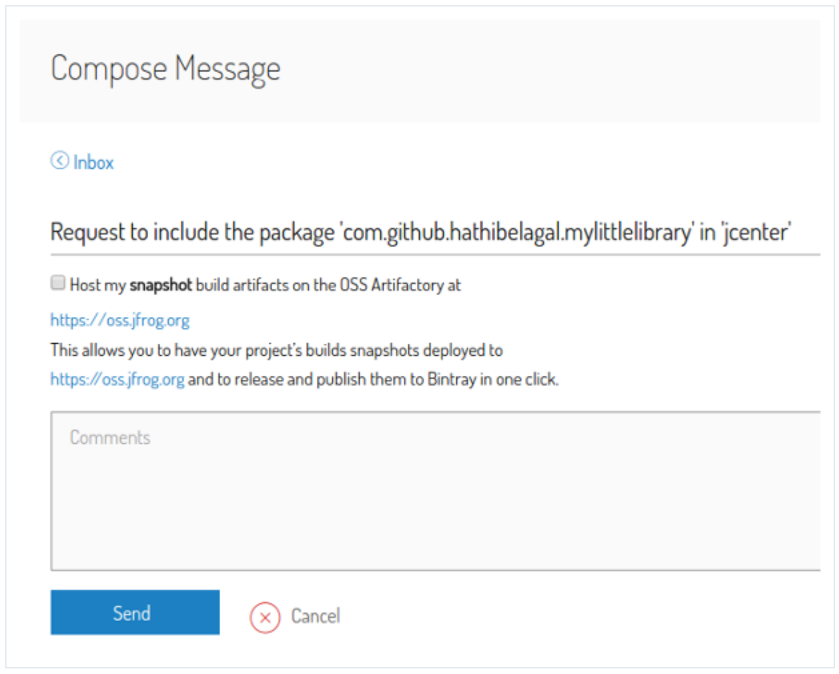
5. Adding the Library to JCenter

By default, Android Studio searches for libraries in a repository called **JCenter**. If you include your library in the JCenter repository, developers won't have to add anything to their `repositories` list.

To add your library to JCenter, open a browser and visit your Bintray package's details page. Click the button labeled **Add to JCenter**.



You will then be taken to a page that lets you compose a message. You can use the **Comments** field to optionally mention any details about the library.

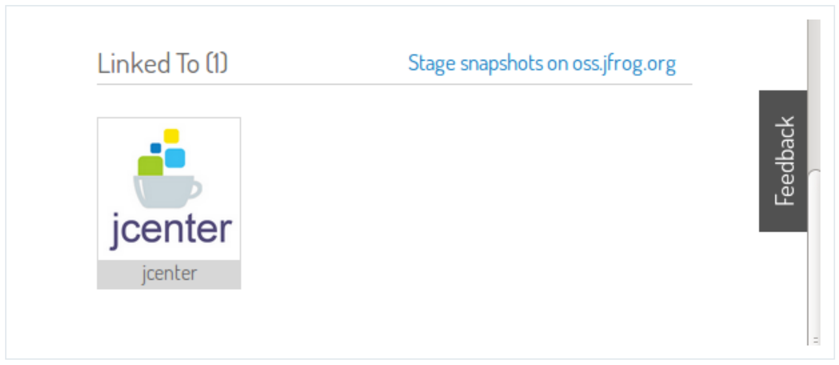

Compose Message

Inbox

Request to include the package 'com.github.hathibelagal.mylittlelibrary' in 'jcenter'

Host my **snapshot** build artifacts on the OSS Artifactory at https://oss.jfrog.org
This allows you to have your project's builds snapshots deployed to https://oss.jfrog.org and to release and publish them to Bintray in one click.

Click the **Send** button to begin Bintray's review process. Within a day or two, the folks at Bintray will link your library to the JCenter repository and you will be able to see the link to JCenter on your package's details page.



Any developer can now use your library without changing the list of `repositories`.

Conclusion

In this tutorial, you learned how to create a simple Android library module and publish it to both your own Maven repository and to the JCenter repository. Along the way, you also learned how to create and execute different types of gradle tasks.

To learn more about Bintray, visit Bintray's user manual.

Ashraff Hathibelagal

Hathibelagal is an independent Android app developer and blogger who loves tinkering with new frameworks, SDKs and devices.

Email Address

Update me weekly

Envato Tuts+ tutorials are translated into other languages by our community members—you can be involved too!

Translate this post

Powered by native

24 Comments Tuts+ Hub 1 Login

Recommend 6 Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

Tomer Bu • 2 years ago

Thanks for the Excellent tutorial. Best one out there!
you have a type in here:

```
artifacts {
archives generateJavaDocsJar
archives generateSourcesJar
}
```

should change generateJavaDocsJar to: generateJavadocsJar (lower case letter d)

also, I head errors with building the javadocs since it's very strict:
adding
failOnError false to the task like so:

```
task generateJavadocs(type: Javadoc) {
failOnError false
source = android.sourceSets.main.java.srcDirs
classpath += project.files(android.getBootClasspath().join(File.pathSeparator))
}
```

did it for me :-)

3 • Reply • Share ›

Param Bug → Tomer Bu • 2 years ago

Thanks for the case sensitive warning, I was stuck there for 2 days.

But right now I'm facing this error while performing the installation.

```
Error:Execution failed for task ':libraryutilities:install'.
> Could not publish configuration 'archives'
> Failed to install artifact https://github.com/Aritra17...
C:\Users\Aritra\.m2\repository\https:\github\com\Aritra1704\UtilsLibrary\librar
1.0.1.aar (The filename, directory name, or volume label syntax is incorrect)
```

I'm using windows and i really don't have a clue what I'm doing wrong.

Please help.

• Reply • Share ›

Swati Garg • 2 years ago

Hi.
I am getting this error on running the tasks :

```
Error:(37, 0) Could not find property 'generateJavaDocsJar' on
org.gradle.api.internal.artifacts.dsl.DefaultArtifactHandler_Decorated@2688bec9.
```

Someone, please help in resolving this error.

1 • Reply • Share ›

Bayu Wijaya Permana Putra → Swati Garg • a month ago

I think that is should be generateJavadocsJar not generateJavaDocsJar

• Reply • Share ›

Param Bug → Swati Garg • 2 years ago

Hi,

Did you find any solution for this?

• Reply • Share ›

Swati Garg → Param Bug • 2 years ago

Not yet.

• Reply • Share ›

Madhur Gupta → Swati Garg • 9 months ago

Hi,
If you get the solution, then please share.

• Reply • Share ›

Reuben Tan → Madhur Gupta • 7 months ago

it should be 'generateJavadocsJar' as the above Jar files stated a lower case d but the artifacts has a typo

3 ^ | ∨ • Reply • Share ›

Mahmud Basunia • 5 months ago

This is an excellent tutorial to start with! deployed my first library as directed!!! thanks.

^ | ∨ • Reply • Share ›

ashrafunissa • 8 months ago

how to make our own sdks

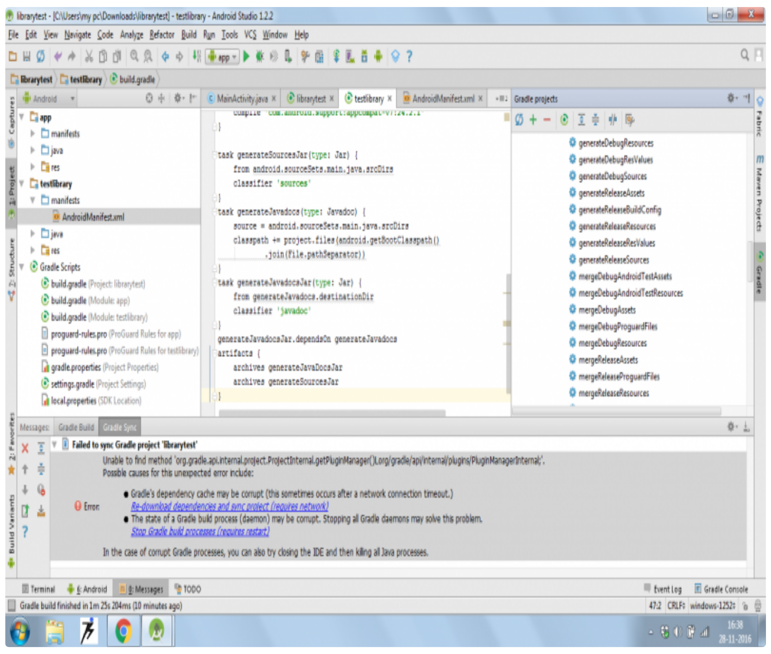
^ | ∨ • Reply • Share ›

Marco Ramos • a year ago

I can't see "Add To jcenter" on binary.com. Please help me.

^ | ∨ • Reply • Share ›

Subham Gupta • a year ago



```
task generateSourcesJar(type: Jar) {
    from android.sourceSets.main.java.srcDirs
    classifier 'sources'
}
task generateJavadocs(type: Javadoc) {
    source = android.sourceSets.main.java.srcDirs
    classpath += project.files(android.getBootClasspath()
            .join(File.pathSeparator))
}
task generateJavadocsJar(type: Jar) {
    from generateJavadocs.destinationDir
    classifier 'javadoc'
}
generateJavadocsJar.dependsOn generateJavadocs
artifacts {
    archives generateJavaDocsJar
    archives generateSourcesJar
}
```

```
Failed to sync Gradle project 'librarytest'
Unable to find method 'org.gradle.api.internal.project.ProjectInternal.getPluginManager()Lorg/gradle/api/internal/plugins/PluginManagerInternal;'.
Possible causes for this unexpected error include:
Error:  • Gradle's dependency cache may be corrupt (this sometimes occurs after a network connection timeout.)
          Re-download dependencies and sync project (requires network)
        • The state of a Gradle build process (daemon) may be corrupt. Stopping all Gradle daemons may solve this problem.
          Stop Gradle build processes (requires restart)
In the case of corrupt Gradle processes, you can also try closing the IDE and then killing all Java processes.

Gradle build finished in 1m 25s 204ms (10 minutes ago)
```

librarytest - [C:\Users\my pc\Downloads\librarytest] - testlibrary - Android Studio 1.2.2
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
app
manifests
java
res
testlibrary
AndroidManifest.xml
Gradle Scripts
build.gradle (Project: librarytest)
build.gradle (Module: app)
build.gradle (Module: testlibrary)
proguard-rules.pro (ProGuard Rules for app)
proguard-rules.pro (ProGuard Rules for testlibrary)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)
MainActivity.java
librarytest
testlibrary
AndroidManifest.xml
Gradle projects
generateDebugResources
generateDebugResValues
generateDebugSources
generateReleaseAssets
generateReleaseBuildConfig
generateReleaseResources
generateReleaseResValues
generateReleaseSources
mergeDebugAndroidTestAssets
mergeDebugAndroidTestResources
mergeDebugAssets
mergeDebugProguardFiles
mergeDebugResources
mergeReleaseAssets
mergeReleaseProguardFiles
mergeReleaseResources
Messages: Gradle Build Gradle Sync
Terminal
6: Android
0: Messages
TODO
Event Log
Gradle Console
47:2 CRLF: windows-1252
16:38
28-11-2016

Hi

My Gradle Project window is not showing install option. Please suggest what to do

^ | ∨ • Reply • Share ›

Clipart ➦ Subham Gupta • 7 months ago

Graceful work you have here.

1 ^ | ∨ • Reply • Share ›

Samuel Agbede • a year ago

Thanks for the tutorial. How do I please use my library locally but in a completely different project?

^ | ∨ • Reply • Share ›

Srt • a year ago

Amazing tutorial!!.. How can I update my repository once I published it? Like, how can I publish a new version of my existing library from Android Studio?

^ | ∨ • Reply • Share ›

Mir Suhail • a year ago

Hi, there is one more simple way of publishing a library via jitpack.io...can you please expalin the advantage of using above mentioned method?

^ | ∨ • Reply • Share ›

SmartGoBuy • 2 years ago

how do i generate the source jar

^ | ∨ • Reply • Share ›

Bayu Wijaya Permana Putra ➦ SmartGoBuy • a month ago

In this tutorial will generate 4 files directly including source jar when you double click 'install'

^ | ∨ • Reply • Share ›

Ameya Godse • 2 years ago

In which build.gradle do we mention the 'group' and 'version' tags. How exactly is it to be done ? And how to create a jar Task ?

^ | ∨ • Reply • Share ›

Himani Tank ➦ Ameya Godse • a month ago

did you get answer?

^ | ∨ • Reply • Share ›

Aiman Baharum ➦ Ameya Godse • 2 years ago

This tutorial isn't well-directed

^ | ∨ • Reply • Share ›

Siddharth Verma ➦ Ameya Godse • 2 years ago

Did you get an answer to this?

^ | ∨ • Reply • Share ›

Shivanand Darur • 2 years ago

Ultimate tutorial. Thanks for the detailed explaination.

^ | ∨ • Reply • Share ›

Kaushik Bharadwaj • 2 years ago

Wow this a really amazing tutorial !