googlesamples / **android-ndk**

hello-libs: Error with Gradle plugin 3.0.0 when re-building libs #450

New issue

Closed **joe-skb7** opened this issue on 13 Nov 2017 · 13 comments

joe-skb7 commented on 13 Nov 2017 • edited Contributor

I'm trying to make libs to be rebuilt on each build. For that I did next:

1. Removed pre-built libraries (`distribution/` directory)
2. Enabled `gen-libs` module building (commented out corresponding lines in `app/build.gradle` and `settings.gradle` files)

With current state of android-ndk project (commit `4df5a27`), build works fine.

But after updating the project to use Gradle plugin 3.0.0 (in Android Studio 3.0), I'm seeing next error when running build:

```
:app:externalNativeBuildDebug
Build hello-libs arm64-v8a
ninja: error: '.../hello-libs/distribution/gmath/lib/arm64-v8a/libgmath.a', needed by
'.../hello-libs/app/build/intermediates/cmake/debug/obj/arm64-v8a/libhello-libs.so',
missing and no known rule to make it
:app:externalNativeBuildDebug FAILED

FAILURE: Build failed with an exception.

* What went wrong:
Execution failed for task ':app:externalNativeBuildDebug'.
> Build command failed.
  Error while executing process ~/Android/Sdk/cmake/3.6.4111459/bin/cmake with arguments
{--build .../hello-libs/app/.externalNativeBuild/cmake/debug/arm64-v8a --target hello-
libs}
  ninja: error: '.../hello-libs/distribution/gmath/lib/arm64-v8a/libgmath.a', needed by
'../hello-libs/app/build/intermediates/cmake/debug/obj/arm64-v8a/libhello-libs.so',
missing and no known rule to make it
```

Is there some dependency missing? As far as I understand, `libhello-libs.so` must depend on `libgmath.a` and `libgperf.so` , but we somehow don't have such a dependency. Can you fix it please?

Supplementary 1: when doing `Build -> Make Module 'gen-libs'` first, project build works fine. But this is just a workaround. We need to come up with proper dependency instead.

Supplementary 2: I found one possible solution to this problem (not sure if it's the best, though). Just added next code in the end of `app/build.gradle` and now build seems to work correctly:

```
tasks.whenTaskAdded { task ->
    if (task.name == 'externalNativeBuildRelease') {
        task.dependsOn ":gen-libs:externalNativeBuildRelease"
    } else if (task.name == 'externalNativeBuildDebug') {
        task.dependsOn ":gen-libs:externalNativeBuildDebug"
    }
}
```

Also I found this solution using `sourceSets {}` . Didn't test it yet, though.

Please advice if there is a best approach or we can use something better?

❤️ 2

Assignees

 ggfan

Labels

None yet

Projects

None yet

Milestone

No milestone

Notifications

2 participants

ggfan commented on 13 Nov 2017 • edited

Contributor

Thanks for reporting the issue. Filed a bug https://b.corp.google.com/issues/69616088, let's follow up there to see if there is something could be done inside Studio.

A couple of points for this sample:

- the sample was aimed to show how to use 3rd party libs, not meant to make a typical case of producing 3rd party lib and distributing.
- because of the above goal, the producing portion was customized to make really independent (using $user_name/tmp as build directory etc), and finally copying out the header/libs into a common directory ("distribution") so app uses libs from that directory, not from project dependency location.
- leaving the lib source code in the sample is for reference purpose: easy to see what lib does. Looks like it causes confusion, I will look into removing lib generating build script in the future; do let me know if you do not want it to be removed.

Now for your 2 solutions

- sourceSet. It affects the packing of the lib into APK, I do not think it will affect the dependency logic -- whatever you copy to jniLibs directory will be copied and merged into APK's libs directory. Not very relevant to our problem here.
- forced dependency (task.dependsOn ":gen-libs:externalNativeBuildRelease") that is good one to use. If Android Studio is NOT missing anything, this would be one to go for now (again if you want to enforce a project dependency)

If the lib is purely native code(no java code at all), I would go with CMake's add_subdirectory() to add dependency into Application's CMakeLists.txt: everything is on native side, easier to maintain and less confusion.

👍 1

ggfan self-assigned this on 22 Nov 2017

ggfan referenced this issue on 22 Nov 2017

hello-libs: Update project for Gradle Plugin 3.0.0 #449 Closed

joe-skb7 commented on 22 Nov 2017 • edited by ggfan

Contributor

Thank you for your comments. Please see my notes inline.

> Thanks for reporting the issue. Filed a bug https://issuetracker.google.com/issues/69616088, let's follow up there to see if there is something could be done inside Studio.

Can't see it, it asks me for google.com login (and I don't have one).

> the sample was aimed to show how to use 3rd party libs, not meant to make a typical case of producing 3rd party lib and distributing.

I understand and agree that this issue is "out of scope" for this project. But as I understand it's often the case in real Android apps development (having some C library of yours and Java code depending on it, which you'd like to build together). So it would be great to have some reference example for this case, too.

Just my two cents: if you have some free time for this (or some other folks), I guess it's a good idea to add such a project to `googlesamples/android-ndk` . It could be a good reference point for catching Android Studio bugs, too.

> because of the above goal, the producing portion was customized to make really independent (using $user_name/tmp as build directory etc), and finally copying out the header/libs into a common directory ("distribution") so app uses libs from that directory, not from project dependency location.

In case when I have my own C libary, and Java code using it, how do you think, which way would be the best to handle such project structure? I mean, independently distributing ("installing") libs, as in this project, or just using `.so` library from built directory?

> Leaving the lib source code in the sample is for reference purpose: easy to see what lib does.
> Looks like it causes confusion, I will look into removing lib generating build script in the future; do let me know if you do not want it to be removed.

Well, from VCS point of view, it's bad too keep built libraries under Git control, yes. And I think you correct about confusion: in common situation user has some 3rd party sources, and he still has to build it before using. So I'd say let's leave this action up to him in this project as well.

> sourceSet. It affects the packing of the lib into APK, I do not think it will affect the dependency logic -- whatever you copy to jniLibs directory will be copied and merged into APK's libs directory. Not very relevant to our problem here.

Thanks for cleaning this up. I'm not a Java / Android apps developer, it's just my side project, so you must forgive me for misleading :)

> forced dependency (task.dependsOn ":gen-libs:externalNativeBuildRelease") that is good one to use. If Android Studio is NOT missing anything, this would be one to go for now (again if you want to enforce a project dependency)

Well, in *most* cases it works for me. But one time I noticed some issue (which is probably Android Studio related): some source files for my library weren't built, which led to linking error. Maybe it was some misconfiguration of my project... Will file the bug, shall it happen again.

> If the lib is purely native code(no java code at all), I would go with CMake's add_subdirectory() to add dependency into Application's CMakeLists.txt: everything is on native side, easier to maintain and less confusion.

I will try to do so, thank you!

ggfan commented on 22 Nov 2017 Contributor

thank you for the follow-up. In the case you have a native lib and supporting Android Java side. It might benefit in the long run to:

1. create a thin java library on top of it
2. let your java lib calls into your own native code: this native code would be another thin layer wrapping your pure native lib (jni calls that support java side have interface of java package and class name baked in)
3. your thin native wrapper build together with your own pure C/C++ lib (this will get into one shared lib)
4. the whole thing would be an AAR lib, and distribute for Android Applications. But all applications needs your service only deal with your Java thin layer lib (interface).

If there is future maintenance, you core lib would be common for all different platforms/OSes (desktop OS, mobiles OSes): one fix in common lib, every platform would have it.

my view may not be the best approach :-)

joe-skb7 commented on 22 Nov 2017 • edited Contributor

Btw, I just updated to Android Studio 3.0.1 (and Gradle plugin 3.0.1), and now the mentioned bug is reproducing every time. I noticed that `libnative.so` is being built for more architectures than `gen-lib` module library. Next I noticed that `app/build.gradle` has `abiFilters`, but `gen-lib/build.gradle` doesn't.

So I just added the same `abiFilters` line to `gen-lib/build.gradle` and this fixed the issue:

```
--- a/gen-libs/build.gradle
+++ b/gen-libs/build.gradle
```

```
@@ -8,7 +8,9 @@ android {
         targetSdkVersion 26
         versionCode 1
         versionName "1.0"
-
+        ndk {
+            abiFilters 'x86', 'x86_64', 'armeabi', 'armeabi-v7a', 'arm64-v8a'
+        }
         externalNativeBuild {
             cmake {
                 arguments '-DANDROID_PLATFORM=android-18',
```

This change and `task.dependsOn` change are fixing all build problems, and everything start to work just fine in all cases. I can send 2 pull requests, if you want me to.

ggfan commented on 22 Nov 2017 • edited — Contributor

that is good point. should update to get it in sync. The background is this:

- when distributing libs, we would like to support all ABIs, so by default, it should build for all (x86, arm and mips)
- application could choose which one it wants to use: so that was reason inside application, ABI is specified. (libs have more capability than apps)
  if upgrade lib to support android-18 and above, application could not support anything below android-18. So keeping libs in lowest possible api level supports more apps.

ggfan commented on 22 Nov 2017 — Contributor

copied the wrong link for the bug, this one might be accessible to you?
https://issuetracker.google.com/issues/69616088

joe-skb7 commented on 22 Nov 2017 — Contributor

Yes, that one opens fine for me, thanks.

As I understand, you want to keep things as they are right now, so I won't send pull requests. But anyway, this information may be useful for someone. Let's keep it here.

ggfan commented on 22 Nov 2017 — Contributor

for PR, yeah, please upload. armeabi is also deprecating, so we do not have to build that one either in this sample. By default, we probably also do not want to trigger the build of the libs. thanks!

ggfan commented on 22 Nov 2017 — Contributor

we do like to update it to Android Studio 3.0 to build, please create PR when you get a chance (even when there will be a fix in studio, we still need the workaround for the time being). thanks!

joe-skb7 added a commit to joe-skb7/android-ndk that referenced this issue on 23 Nov 2017

`hello-libs: Add abiFilters to gen-libs module` ... `12b5639`

joe-skb7 added a commit to joe-skb7/android-ndk that referenced this issue on 23 Nov 2017

`hello-libs: Fix dependencies issue with new Gradle plugin` ... `561ca31`

joe-skb7 referenced this issue on 23 Nov 2017

hello-libs: Update to Gradle plugin 3.0.1 and fix collateral issues #455 Merged

joe-skb7 commented on 23 Nov 2017 • edited — Contributor

Here you go! :) The whole patch series is #455.

ggfan added a commit that referenced this issue on 27 Nov 2017

 `Patch for #450 to disable lib build ...` ✓ `507cc49`

ggfan added a commit that referenced this issue on 27 Nov 2017

 `Patch for #450 to disable lib build ...` ✓ `1d853dc`

ggfan added a commit that referenced this issue on 27 Nov 2017

 `Merge pull request #456 from googlesamples/issue-450-fixup ...` Verified `5ce57be`

ggfan commented on 27 Nov 2017 • edited — Contributor

thank you so much! please take a look at the latest code: the main difference is to whether keep binaries in the repo. I like to keep them because the main goal is to demo "the use of 3rd party lib". If it is pure native lib, the correct usage model is to use add_subdirectory().

ggfan commented on 28 Nov 2017 — Contributor

All code are merged: it builds and runs fine for Pixel XL + Android 8.0. For example of adding java/jni-wrapper on top of native lib, refer to TensorFlow inference module.
Thank you for your help along the way!

 **ggfan** closed this on 28 Nov 2017

joe-skb7 commented on 28 Nov 2017 — Contributor

Great! Thank you for the link and for suggestion about `add_subdirectory()`. I will try it later in my project.