C++ source not showing up in Android Studio for pre-built libs

STATUS UPDATE No update yet. Edit

DESCRIPTION ss...@gmail.com created issue #1 Au

I have a unified build workflow for my C++ project across Android and several other platforms. After the C++ build, it runs a generated Gradle project to build the APK. It places my .so file in app/s
v8a. Everything builds together and runs fine. However, there is no option to select symbols so I can't set breakpoints in the C++ code. Setting jni.srcDir does't seem to do anything.

I would expect it to automatically load the symbols it finds and also provide an option for me to specify the location in Gradle so it's ready for debugging.

My current workaround is to use File > Profile or debug APK, and then click 'Add debug symbols' to add the symbols directory. Then, it shows all the source files and debugging is easy. Unfortuna
clicks and forces me into a different project than I'm using to build the C++ so it's preventing me from setting up a full C++/Java/APK dev loop in Android Studio.

Links (1)

Links (1)

"...possible for your native build tool to create a compile_commands.json file, which contains flags used to compile the source file (In theory we can make Android Studio provide many IDE features

COMMENTS All comments

uc...@google.com <uc...@google.com>

Assigned to an...@google.com.

tg...@google.com <tg...@google.com> #2 Aug

Reassigned to tg...@google.com.

Hi, just to make I understand correctly. So you are building the .so file yourself and not using the Android Gradle Plugin. The .so files are then included directly in build.gradle. Is this correct?

ss...@gmail.com <ss...@gmail.com> #3 Aug

Correct, the .so is built with a CMake-generated Ninja project whose build is called as a prebuild step in Gradle. We aren't using Gradle's externalNativeBuild CMake integration because that w
platform workflow of having CMake be the source of our build configuration as well as post-build steps such as copying assets for packaging.

tg...@google.com <tg...@google.com> #4 Aug

Thanks! Since you don't use `externalNativeBuild`, Android Studio would not be able to figure out where the native source files are located. Do expect to manually link them to the .so files i
through some UI in Android Studio, just like the APK debugging workflow?

ss...@gmail.com <ss...@gmail.com> #5 Aug

Yes, that would be great!

tg...@google.com <tg...@google.com> #6 Se

Hi, actually when you use APK debugging, APK reloading introduced in Android Studio 3.6 and above should automatically pick up your updated APK and preserve all your existing settings. W
your workflow?

ss...@gmail.com <ss...@gmail.com> #7 Se

Are you talking about the 'Profile or debug APK' option?

If so, it's not so great as a replacement for a typical development environment because I have to build by other means before I can use it. I also won't be able to navigate through C++ files if I
off for, say, a Release build.

tg...@google.com <tg...@google.com> #8 Sep

Thanks! Could you elaborate a bit more on your workflow?

- Do you expect to be able to browse through C++ files with some IDE features like smart navigation, symbol search, etc?

- Do you want to edit the files with IDE completion and real time inspections?
- Do you use some other IDE or text editors to edit the C++ files?
- Is it possible for your native build tool to create a compile_commands.json file, which contains flags used to compile the source file (In theory we can make Android Studio provide m
  it.)?

Message last modified on Sep 15, 2020 04:26AM

ss...@gmail.com <ss...@gmail.com> #9 Sep

Yes, I would expect to be able to browse and search C++ files. I would also expect IDE completion and real-time inspections.

I often use Visual Studio or Visual Studio Code and sometimes Xcode to edit C++ files. Since I work on multiplatform C++ software, much of the development can happen on Windows or mac
emphasize C++ workflows. Currently, I only use Android Studio when I'm testing or debugging an Android-specific problem. I'd be interested in that changing, though. By supporting C++ work
Java/Kotlin, it could become an excellent developer experience for those of us trying to write most of our application in C++ so it can be written once and run everywhere.

CMake is already generating compile_commands.json. If it needs to be copied to a specific location in the gradle project, it would be easy to do in CMake as well.

tg...@google.com <tg...@google.com> #10 Sep

Thanks! We will evaluate adding support for this use case. In the mean time, would it work for you if you create a placeholder build.gradle file that uses `externalNativeBuild` block to refere
CMakeLists.txt as a temporary workaround?

ss...@gmail.com <ss...@gmail.com> #11 Sep

Thank you!

We already use externalNativeBuild along with a CMake variable that, when present, forces CMake to return immediately so it won't try to build, but allows us to navigate the C++ source tree.

ar...@google.com <ar...@google.com>

Reassigned to jo...@google.com.

jo...@google.com <jo...@google.com>

Reassigned to an...@google.com.

gi...@google.com <gi...@google.com>

Status: New