Android Developers

CMake

Using Android Studio 2.2 and higher (https://developer.android.google.cn/studio/index.html), you can use the NDK and CMake (https://cmake.org/) to compile C and C++ code into a native library. Android Studio then packages your library into your APK using Gradle, the IDE's integrated build system (https://developer.android.google.cn/studio/build/index.html).

If you are new to using CMake with Android Studio, go to Add C and C++ Code to Your Project (https://developer.android.google.cn/studio/projects/add-native-code.html) to learn the basics of adding native sources to your project, creating a CMake build script, and adding your CMake project as a Gradle dependency. This page provides some additional information you can use to customize your CMake build.

On this page

Using CMake variables in Gradle
Understanding the CMake build command
YASM support in CMake
Reporting problems

Using CMake variables in Gradle

Once you link Gradle to your CMake project (https://developer.android.google.cn/studio/projects/add-native-code.html#link-gradle), you can configure certain NDK-specific variables that change the way CMake builds your native libraries. To pass an argument to CMake from your module-level `build.gradle` file, use the following DSL:

```
android {
  ...
  defaultConfig {
    ...
    // This block is different from the one you use to link Gradle
    // to your CMake build script.
    externalNativeBuild {
        // Use the following syntax when passing arguments to variables:
        // arguments "-DVAR_NAME=ARGUMENT".
        arguments "-DANDROID_ARM_NEON=TRUE",
        // If you're passing multiple arguments to a variable, pass them together:
        // arguments "-DVAR_NAME=ARG_1 ARG_2"
        // The following line passes 'rtti' and 'exceptions' to 'ANDROID_CPP_FEATURES'.
                  "-DANDROID_CPP_FEATURES=rtti exceptions"
      }
    }
  }
  buildTypes {...}

  // Use this block to link Gradle to your CMake build script.
  externalNativeBuild {
    cmake {...}
  }
}
```

The following table describes some of the variables you can configure when using CMake with the NDK.

Variable name	Arguments	Description
`ANDROID_TOOLCHAIN`	• `clang` (default) • `gcc` (deprecated)	Specifies the compiler toolchain CMak
`ANDROID_PLATFORM`	For a complete list of platform names and corresponding Android system images, see Android NDK Native APIs (https://developer.android.google.cn/ndk/guides/stable_apis.html).	Specifies the name of the target Andro example, `android-18` specifies Andro Instead of changing this flag directly, y `minSdkVersion` property in the `defau` `productFlavors` blocks of your modu `build.gradle` file (https://developer.android.google.cn/studio/ level). This makes sure your library is u

		Android. The CMake toolchain then ch platform version for the ABI you're buil following logic: 1. If there exists a platform version fo `minSdkVersion`, CMake uses that 2. Otherwise, if there exists platform v `minSdkVersion` for the ABI, CMake those platform versions. This is a re because a missing platform versior there were no changes to the native the previous available version. 3. Otherwise, CMake uses the next av version higher than `minSdkVersio`
`ANDROID_STL`	For a complete list of options, see Helper Runtimes (https://developer.android.google.cn/ndk/guides/cpp-support.html#hr) By default, CMake uses `gnustl_static`.	Specifies the STL CMake should use.
`ANDROID_PIE`	• `ON` (default when `ANDROID_PLATFORM = android-16` and higher) • `OFF` (default when `ANDROID_PLATFORM = android-15` and lower)	Specifies whether to use position-inde (PIE). Android's dynamic linker suppor (API level 16) and higher.
`ANDROID_CPP_FEATURES`	This variable is empty by default. However, the following are a few examples of arguments you can pass: • `rtti` (indicates that your code uses RTTI)	Specifies certain C++ features CMake compiling your native library, such as R Information) and C++ exceptions.

	• `exceptions` (indicates that your code uses C++ exceptions)	
`ANDROID_ALLOW_UNDEFINED_SYMBOLS`	• `TRUE` • `FALSE` (default)	Specifies whether to throw an undefine CMake encounters an undefined refere your native library. To disable these typ variable to `TRUE`.
`ANDROID_ARM_MODE`	• `arm` • `thumb` (default)	Specifies whether to generate ARM tar `thumb` mode. In `thumb` mode, each ins wide and linked with the STL libraries i directory. Passing `arm` tells CMake to g object files in 32-bit arm mode.
`ANDROID_ARM_NEON`	• `TRUE` • `FALSE` (default)	Specifies whether CMake should build with NEON support.
`ANDROID_DISABLE_NO_EXECUTE`	• `TRUE` • `FALSE` (default)	Specifies whether to enable NX bit (https://en.wikipedia.org/wiki/NX_bit) , or feature. To disable this feature, pass `T`
`ANDROID_DISABLE_RELRO`	• `TRUE` • `FALSE` (default)	Specifies whether to enable read-only r
`ANDROID_DISABLE_FORMAT_STRING_CHECKS`	• `TRUE` • `FALSE` (default)	Specifies whether to compile your sou string protection. When enabled, the co error if a non-constant format string is style function.

When debugging CMake build issues, it's helpful to know the specific build arguments that Android Studio uses when cross-compiling for Android.

Android Studio saves the build arguments it uses for executing a CMake build, in a `cmake_build_command.txt` file. For each Application Binary Interface (ABI) that your app targets, and each build type (https://developer.android.google.cn/studio/build/build-variants.html) for those ABIs (namely, *release* or *debug*), Android Studio generates a copy of the `cmake_build_command.txt` file for that specific configuration. Android Studio then places the files it generates in the following directories:

```
<project-root>/<module-root>/.externalNativeBuild/cmake/<build-type>/<ABI>/
```

Tip: In Android Studio, you can quickly view these files by using the search keyboard shortcut (`shift+shift`) and entering *cmake_build_command.txt* in the input field.

The following snippet shows an example of the CMake arguments to build a debuggable release of the `hello-jni` `(https://github.com/googlesamples/android-ndk/tree/master/hello-jni)` sample targeting the `armeabi-v7a` architecture.

```
Executable : /usr/local/google/home/{$USER}/Android/Sdk/cmake/3.6.3155560/bin/cmake
arguments :
-H/usr/local/google/home/{$USER}/Dev/github-projects/googlesamples/android-ndk/hello-jni/app/src/main/cpp
-B/usr/local/google/home/{$USER}/Dev/github-projects/googlesamples/android-ndk/hello-jni/app/.externalNativeBuild/cmake
-GAndroid Gradle - Ninja
-DANDROID_ABI=armeabi-v7a
-DANDROID_NDK=/usr/local/google/home/{$USER}/Android/Sdk/ndk-bundle
-DCMAKE_LIBRARY_OUTPUT_DIRECTORY=/usr/local/google/home/{$USER}/Dev/github-projects/googlesamples/android-ndk/hello-jni
-DCMAKE_BUILD_TYPE=Debug
-DCMAKE_MAKE_PROGRAM=/usr/local/google/home/{$USER}/Android/Sdk/cmake/3.6.3155560/bin/ninja
-DCMAKE_TOOLCHAIN_FILE=/usr/local/google/home/{$USER}/Android/Sdk/ndk-bundle/build/cmake/android.toolchain.cmake
-DANDROID_NATIVE_API_LEVEL=23
-DANDROID_TOOLCHAIN=clang
jvmArgs :
```

Build arguments

The following table highlights the key CMake build arguments for Android. These build arguments are not meant to be set by developers. Instead, the Android Plugin for Gradle (https://developer.android.google.cn/studio/releases/gradle-plugin.html) sets these arguments based on the `build.gradle` configuration in your project.

Build Arguments	Description
`-G <build-system>`	Type of build files that CMake generates. For projects in Android Studio with native code, the `<build-system>` is set to `Android Gradle - Ninja`. This setting indicates that CMake uses the ninja build system (https://ninja-build.org/) to compile and link the C/C++ sources for your app. CMake also generates a `android_gradle_build.json` file which contains metadata for the Gradle plugin about the CMake build such as compiler flags and names of targets. This setting indicates that CMake uses Gradle (http://tools.android.com/tech-docs/new-build-system/user-guide) together with the ninja (https://ninja-build.org/) build system to compile and link the C/C++ sources for your app. The ninja build system is the only generator that Studio supports.
`-DANDROID_ABI <abi>`	The target ABI. The NDK supports a set of ABIs, as described in ABI Management (https://developer.android.google.cn/ndk/guides/abis.html#sa). This option is similar to the `APP_ABI` variable that the `ndk-build (https://developer.android.google.cn/ndk/guides/ndk-build.html)` tool uses. By default, Gradle builds your native library into separate `.so` files for the ABIs that NDK supports, and then packages them all into your APK. If you want Gradle to build only for certain ABI configurations, follow the instructions in Add C and C++ Code to Your Project (https://developer.android.google.cn/studio/projects/add-native-code.html#specify-abi). If the target ABI is not specified, CMake defaults to using `armeabi-v7a`. Valid target names are:

	• `armeabi`: ARMv5TE based CPU with software floating point operations. • `armeabi-v7a`: ARMv7 based devices with hardware FPU instructions (VFPv3_D16). • `armeabi-v7a with NEON`: Same as armeabi-v7a, but enables NEON floating point instructions. This is equivalent to setting `-DANDROID_ABI=armeabi-v7a` and `-DANDROID_ARM_NEON=ON`. • `arm64-v8a`: ARMv8 AArch64 instruction set. • `x86`: IA-32 instruction set. • `x86_64` - Instruction set for the x86-64 architecture.
`-DANDROID_NDK <path>`	Absolute path to the root directory of the NDK installation on your host.
`-DCMAKE_LIBRARY_OUTPUT_DIRECTORY <path>`	Location on your host where CMake puts the `LIBRARY` target files when built.
`-DCMAKE_BUILD_TYPE <type>`	Similar to the build types for the ndk-build (https://developer.android.google.cn/ndk/guides/ndk-build.html) tool. The valid values are `Release` and `Debug`. To simplify debugging, CMake does not strip the release or debug version as part of the build. However, Gradle strips binaries when it packages them in the APK.
`-DCMAKE_MAKE_PROGRAM <program-name>`	Tool to launch the native build system. The Gradle plugin sets this value to the CMake `ninja` generator bundled with the Android SDK.
`-DCMAKE_TOOLCHAIN_FILE <path>`	Path to the `android.toolchain.cmake` file that CMake uses for cross-compiling for Android. Typically, this file is located in the `$NDK/build/cmake/` directory, where `$NDK` is the NDK installation directory on your host. For more information about the toolchain file, see Cross Compiling for Android (https://cmake.org/cmake/help/v3.7/manual/cmake-toolchains.7.html#cross-compiling-for-android) .
`-DANDROID_NATIVE_API_LEVEL <level>`	Android API level that CMake compiles for.
`-DANDROID_TOOLCHAIN <type>`	The compiler toolchain that CMake uses. Defaults to `clang`

YASM support in CMake

NDK r15 and higher provides CMake support for building assembly code written in YASM (//yasm.tortall.net) to run on x86 and x86-64 architectures. YASM is an open-source assembler for x86 and x86-64 architectures, based on the NASM assembler.

You may find it useful to link assembly language programs or routines with C code in order to access C libraries or functions from your assembly code. You can also include short assembly routines in your compiled C code to take advantage of the better machine performance that assembly code affords.

To build assembly code with CMake, make the following changes in your project's `CMakeLists.txt`:

1. Call `enable_language` (//cmake.org/cmake/help/latest/command/enable_language.html) with the value set to `ASM_NASM`.
2. Depending on whether you are building a shared library or an executable binary, call `add_library` (//cmake.org/cmake/help/latest/command/add_library.html) or `add_executable` (//cmake.org/cmake/help/latest/command/add_executable.html) . In the arguments, pass in a list of source files consisting of the `.asm` files for the assembly program in YASM and the `.c` files for the associated C libraries or functions.

The following snippet shows how you might configure your `CMakeLists.txt` to build a YASM program as a shared library.

```
cmake_minimum_required(VERSION 3.6.0)

enable_language(ASM_NASM)

add_library(test-yasm SHARED jni/test-yasm.c jni/print_hello.asm)
```

For an example of how to build a YASM program as an executable, see the `yasm`

Reporting problems

If you run into any issues that aren't due to the open source version of CMake, report them via the `android-ndk/ndk (https://github.com/android-ndk/ndk/issues)` issue tracker on GitHub.