(https://google.com/racialequity)

Add C and C++ code to your project

You can add C and C++ code to your Android project by placing the code into a **cpp** directory in your project module. When you build your project, this code is compiled into a native library that Gradle can package with your APK. Your Java or Kotlin code can then call functions in your native library through the Java Native Interface (JNI). To learn more about using the JNI framework, read JNI tips for Android (/training/articles/perf-jni).

Android Studio supports CMake, which is good for cross-platform projects, and ndk-build (/ndk/guides/ndk-build), which can be faster than CMake but only supports Android. Using both CMake and ndk-build in the same module is not currently supported.

If you want to import an existing ndk-build library into your Android Studio project, learn how to link Gradle to your native library project (/studio/projects/gradle-external-native-builds).

This page shows you how to set up Android Studio (#download-ndk) with the necessary build tools, create a new project (#new-project) with C/C++ support, and add new C/C++ files (#create-sources) to your project.

If instead you want to add native code to an existing project, you need to follow these steps:

1. Create new native source files (#create-sources) and add them to your Android Studio project.
    - You can skip this step if you already have native code or want to import a prebuilt native library.
2. Configure CMake (/studio/projects/configure-cmake) to build your native source code into a library. You also require this build script if you are importing and linking against prebuilt or platform libraries.
    - If you have an existing native library that already has a `CMakeLists.txt` build script, or uses ndk-build and includes an `Android.mk` (/ndk/guides/android_mk) build script, you can skip this step.
3. Configure Gradle (/studio/projects/gradle-external-native-builds) by providing a path to your CMake or ndk-build script file. Gradle uses the build script to import source code into

your Android Studio project and package your native library (the SO file) into the APK.

Once you configure your project, you can access your native functions from Java or Kotlin code using the JNI framework (http://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/jniTOC.html). To build and run your app, simply click **Run** ▶ .

Note: If your existing project uses the deprecated **`ndkCompile`** tool, you should migrate to using either CMake or ndk-build. To learn more, go to the section about how to Migrate from ndkCompile (#ndkCompile).

Attention experimental Gradle users: Consider migrating to plugin version 2.2.0 or higher (http://tools.android.com/tech-docs/new-build-system/gradle-experimental/migrate-to-stable), and using CMake or ndk-build to build your native libraries if any of the following apply to you: Your native project already uses CMake or ndk-build; you would rather use a stable version of the Gradle build system; or you want support for add-on tools, such as CCache (https://ccache.samba.org/). Otherwise, you can continue to use the experimental version of Gradle and the Android plugin (http://tools.android.com/tech-docs/new-build-system/gradle-experimental).

Download the NDK and build tools

To compile and debug native code for your app, you need the following components:

- *The Android Native Development Kit (NDK)* (/ndk): a toolset that allows you to use C and C++ code with Android, and provides platform libraries that allow you to manage native activities and access physical device components, such as sensors and touch input.
- *CMake* (https://cmake.org/): an external build tool that works alongside Gradle to build your native library. You do not need this component if you only plan to use ndk-build.
- *LLDB* (http://lldb.llvm.org/): the debugger Android Studio uses to debug native code (/studio/debug).

For information on installing these components, see Install and configure the NDK, CMake, and LLDB (/studio/projects/install-ndk).

Create a new project with C/C++ support

Creating a new project with support for native code is similar to creating any other Android Studio project (/studio/projects/create-project), but there is an additional step:

1. In the **Choose your project** section of the wizard, select the **Native C++** project type.
2. Click **Next**.
3. Complete all other fields in the next section of the wizard.
4. Click **Next**.
5. In the **Customize C++ Support** section of the wizard, you can customize your project with the **C++ Standard** field. Use the drop-down list to select which standardization of C++ you want to use. Selecting **Toolchain Default** uses the default CMake setting.
6. Click **Finish**.

After Android Studio finishes creating your new project, open the **Project** pane from the left side of the IDE and select the **Android** view. As shown in figure 2, Android Studio adds the **cpp** group:

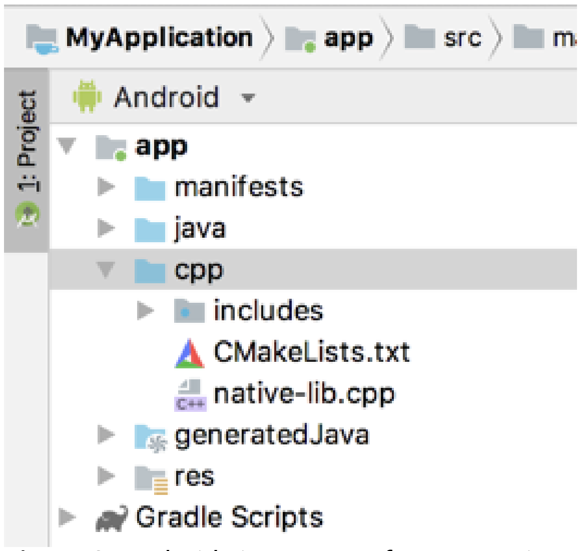


Figure 2. Android view groups for your native sources and external build scripts.

Note: This view does not reflect the actual file hierarchy on disk, but groups similar files to simplify navigating your project.

The **cpp** group is where you can find all the native source files, headers, build scripts for CMake or ndk-build, and prebuilt libraries that are a part of your project. For new projects, Android Studio creates a sample C++ source file, `native-lib.cpp`, and places it in the `src/main/cpp/` directory of your app module. This sample code provides a simple C++ function, `stringFromJNI()`, that returns the string "Hello from C++". You can learn how to add additional source files to your project in the section about how to Create new native source files (#create-sources).

Similar to how `build.gradle` files tell Gradle how to build your app, CMake and ndk-build require a build script to know how to build your native library. For new projects, Android Studio creates a CMake build script, `CMakeLists.txt`, and places it in your module's root directory. To learn more about the contents of this build script, read Configure CMake (/studio/projects/configure-cmake).

Build and run the sample app

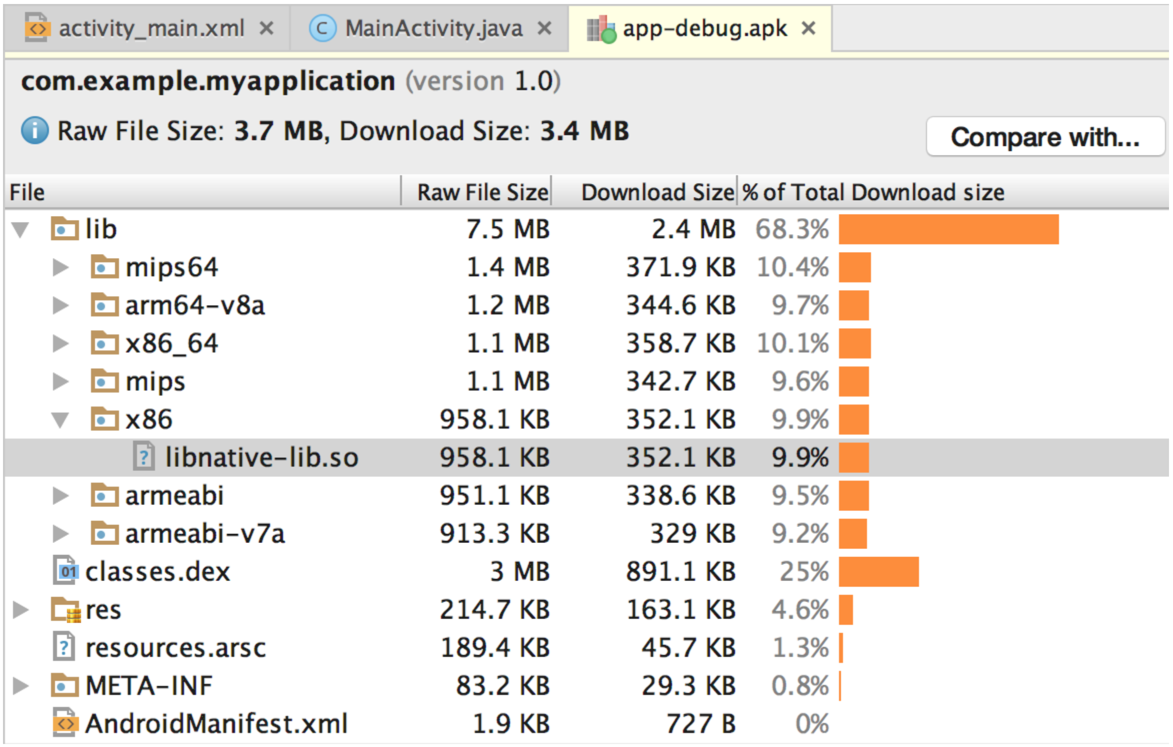
When you click **Run** ▶, Android Studio builds and launches an app that displays the text "Hello from C++" on your Android device or emulator. The following overview describes the events that occur in order to build and run the sample app:

1. Gradle calls upon your external build script, `CMakeLists.txt`.
2. CMake follows commands in the build script to compile a C++ source file, `native-lib.cpp`, into a shared object library and names it `libnative-lib.so`, which Gradle then packages into the APK.
3. During runtime, the app's `MainActivity` loads the native library using `System.loadLibrary()` (/reference/java/lang/System#loadLibrary(java.lang.String)). The library's native function, `stringFromJNI()`, is now available to the app.
4. `MainActivity.onCreate()` calls `stringFromJNI()`, which returns "Hello from C++", and uses it to update the `TextView` (/reference/android/widget/TextView).

Note: Instant Run (/studio/run#instant-run) is not compatible with components of your project written in native code.

If you want to verify that Gradle packages the native library in the APK, you can use the APK Analyzer (/studio/build/apk-analyzer):

1. Select **Build > Build Bundles(s) / APK(s) > Build APK(s)**
2. Select **Build > Analyze APK**.
3. Select the APK from the `app/build/outputs/apk/` directory and click **OK**.
4. As shown in figure 3, you can see `libnative-lib.so` in the APK Analyzer window under `lib/<ABI>/`.



com.example.myapplication (version 1.0)

Raw File Size: 3.7 MB, Download Size: 3.4 MB

File	Raw File Size	Download Size	% of Total Download size
lib	7.5 MB	2.4 MB	68.3%
mips64	1.4 MB	371.9 KB	10.4%
arm64-v8a	1.2 MB	344.6 KB	9.7%
x86_64	1.1 MB	358.7 KB	10.1%
mips	1.1 MB	342.7 KB	9.6%
x86	958.1 KB	352.1 KB	9.9%
libnative-lib.so	958.1 KB	352.1 KB	9.9%
armeabi	951.1 KB	338.6 KB	9.5%
armeabi-v7a	913.3 KB	329 KB	9.2%
classes.dex	3 MB	891.1 KB	25%
res	214.7 KB	163.1 KB	4.6%
resources.arsc	189.4 KB	45.7 KB	1.3%
META-INF	83.2 KB	29.3 KB	0.8%
AndroidManifest.xml	1.9 KB	727 B	0%

Figure 3. Locating a native library using the APK Analyzer.

Tip: If you want to experiment with other Android apps that use native code, click **File > New > Import Sample** and select a sample project from the **Ndk** list.

Create new C/C++ source files

To add new C/C++ source files to an existing project, proceed as follows:

1. If you don't already have a `cpp/` directory in the main source set of your app, create one as follows:

   a. Open the **Project** pane from the left side of the IDE and select the **Project** view from the drop-down menu.
   b. Navigate to ***your-module* > src**, right-click on the **main** directory, and select **New > Directory**.
   c. Enter `cpp` as the directory name and click **OK**.
2. Right-click on the `cpp/` directory and select **New > C/C++ Source File**.
3. Enter a name for your source file, such as `native-lib`.
4. From the **Type** drop-down menu, select the file extension for your source file, such as `.cpp`.
   - You can add other file types to the drop-down menu, such as `.cxx` or `.hxx`, by clicking **Edit File Types**
     . In the **C/C++** dialog box that pops up, select another file extension from the **Source Extension** and **Header Extension** drop-down menus and click **OK**.
5. If you also want to create a header file, check the **Create an associated header** checkbox.
6. Click **OK**.

After you add new C/C++ files to you project, you still need to configure CMake (/studio/projects/configure-cmake) to include them in your native library.

Additional resources

To learn more about supporting C/C++ code in your app, try the following resource.

Codelabs

- Create Hello-CMake with Android Studio (https://codelabs.developers.google.com/codelabs/android-studio-cmake/), a codelab that shows you how to use the Android Studio CMake template to start Android NDK project development

Last updated 2020-10-28 UTC.