**Installing Android Studio**

Android Studio provides everything you need to start developing apps for Android, including the Android Studio IDE and the Android SDK tools.

Before you set up Android Studio, be sure you have installed JDK 6 or higher (the JRE alone is not sufficient)—JDK 7 is required when developing for Android 5.0 and higher. To check if you have JDK installed (and which version), open a terminal and type javac -version. If the JDK is not available or the version is lower than 6, [go download JDK](http://www.oracle.com/technetwork/java/javase/downloads/index.html).

**To set up Android Studio on Windows:**

1. Launch the .exe file you just downloaded.
2. Follow the setup wizard to install Android Studio and any necessary SDK tools.

On some Windows systems, the launcher script does not find where Java is installed. If you encounter this problem, you need to set an environment variable indicating the correct location.

Select **Start menu > Computer > System Properties > Advanced System Properties**. Then open **Advanced tab > Environment Variables** and add a new system variable JAVA\_HOME that points to your JDK folder, for example C:\Program Files\Java\jdk1.7.0\_21.

The individual tools and other SDK packages are saved outside the Android Studio application directory. If you need to access the tools directly, use a terminal to navigate to the location where they are installed. For example:

\Users\<user>\sdk\

**To set up Android Studio on Mac OSX:**

1. Launch the .dmg file you just downloaded.
2. Drag and drop Android Studio into the Applications folder.
3. Open Android Studio and follow the setup wizard to install any necessary SDK tools.

Depending on your security settings, when you attempt to open Android Studio, you might see a warning that says the package is damaged and should be moved to the trash. If this happens, go to **System Preferences > Security & Privacy** and under **Allow applications downloaded from**, select **Anywhere**. Then open Android Studio again.

If you need use the Android SDK tools from a command line, you can access them at:

/Users/<user>/Library/Android/sdk/

**To set up Android Studio on Linux:**

1. Unpack the downloaded ZIP file into an appropriate location for your applications.
2. To launch Android Studio, navigate to the android-studio/bin/ directory in a terminal and execute studio.sh.

You may want to add android-studio/bin/ to your PATH environmental variable so that you can start Android Studio from any directory.

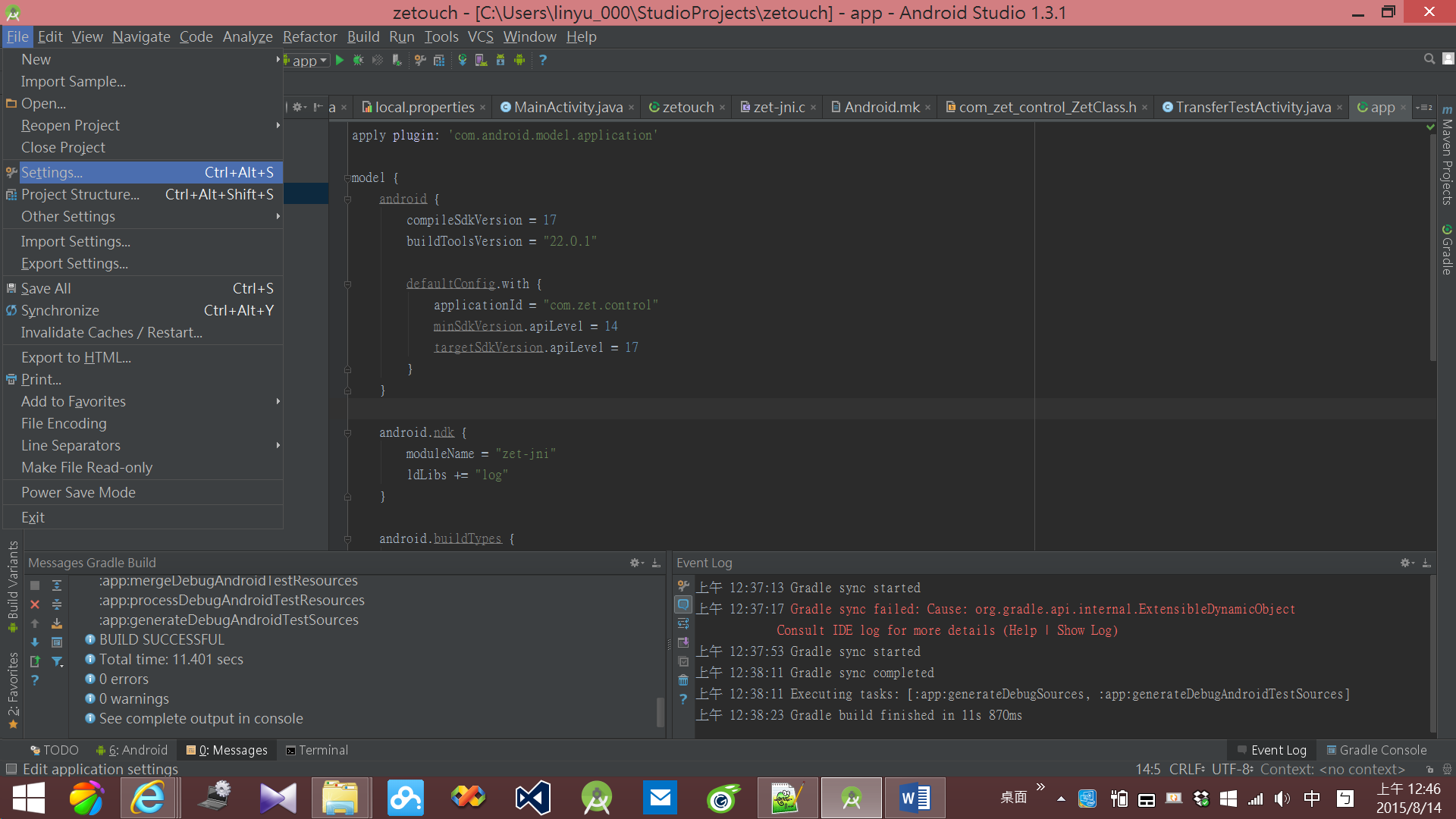
1. If the SDK is not already installed, follow the setup wizard to install the SDK and any necessary SDK tools.

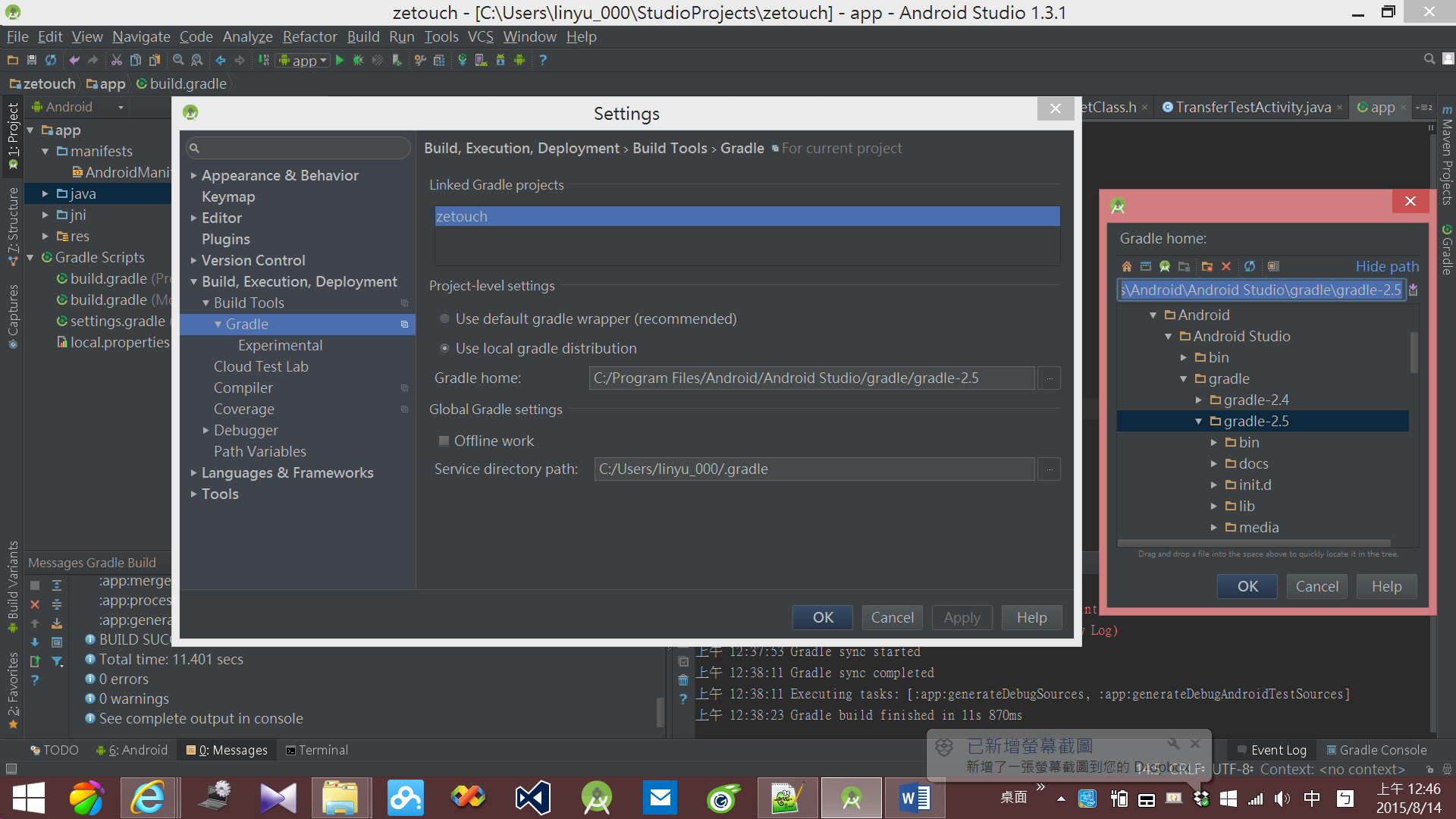
**Note:** You may also need to install the ia32-libs, lib32ncurses5-dev, and lib32stdc++6 packages. These packages are required to support 32-bit apps on a 64-bit machine.

Android Studio is now ready and loaded with the Android developer tools, but there are still a couple packages you should add to make your Android SDK complete.

**Upgrade gradle to 2.5**

1. Extract gradle-2.5-all.zip to C:\Program Files\Android\Android Studio\gradle
2. Use local gradle distribution as Gradle Home.





**Android NDK Preview**

|  |
| --- |
| We’ve just released another update to Android Studio 1.3's canary channel, Release Candidate 1.  This build contains a big new feature: Android NDK support, with support for editing, running and debugging C and C++ code! **Note** however that NDK support is only **preview** quality, and will remain in preview status for the upcoming final 1.3 release. There are many significant limitations (described below), but we hope that the preview will be useful for many Android NDK developers even in its current state!  Limitations  First, the Android NDK support **only** works with the new experimental Gradle plugin for Android (which in turn requires Gradle 2.5, released yesterday.)  While the new gradle plugin delivers some major performance improvements (and Android NDK build support), note that it also requires changes to the build file “DSL” (the way your build is described in your build.gradle files.) Not only will you need to modify your build.gradle files (a process we hope to automate before the plugin graduates from experimental to stable), but we anticipate making some additional incompatible changes along the way.  Second, note also that Android Studio has not yet been updated to fully handle the experimental plugin. This means that for example the Project Structure Dialog, and various quick fixes which automatically update the build data, do not work correctly. You’ll need to edit your build.gradle files manually to configure your projects. As another example, the various templates which update the build files (such as New Module), have not yet been updated.  Finally, there are still some known bugs in this build; when debugging with LLDB breakpoints do not always work on Windows; if you run into this, you can switch to GDB debugging as a temporary workaround.  Bundled Android NDK  Historically, the Android NDK was distributed separately from the Android SDK, as a giant .zip file download, and installed in a separate location from where the Android SDK resides.  In this release we’ve tried to unify this; the NDK should now be installed under the SDK home in “ndk-bundle”. We’ve updated the IDE built-in SDK Manager to let you install the NDK in the right place, and the Project Structure Dialog’s SDK Location panel (where you can edit the locations), now also offers a quick link to install it directly; just click on the Download hyperlink shown below:  [http://tools.android.com/_/rsrc/1436483192508/tech-docs/android-ndk-preview/ndk-install-link.png](http://tools.android.com/tech-docs/android-ndk-preview/ndk-install-link.png?attredirects=0)  Note that the NDK is a **large** download (~1 GB!) so unless you have a really great network connection, prepare to be patient...  Speaking of large downloads, note also that the IDE update patches from previous 1.3 builds to this one are really large: not only does this build contain all the new code to support Android NDK development, but it also contains LLDB binaries (~200 MB uncompressed). This is just a temporary situation; we'll move these into the NDK.  How to get started  Since the Android NDK support depends on an experimental plugin, with an incompatible build file syntax which we know will change over the next few months, we have not added a simple way to create Android NDK projects. Instead, to explore the Android NDK support, either   1. Follow the [Experimental Gradle Plugin](http://tools.android.com/tech-docs/new-build-system/gradle-experimental) document which describes how to modify your project to use the current version of the experimental plugin, or 2. Check out some of the existing Android NDK samples which have been updated to work with the new Gradle plugin.   To check out the samples, invoke VCS > Checkout from Version Control > Git, and in the resulting dialog point to <https://github.com/googlesamples/android-ndk.git> . Pick a location to check out the samples into. **NOTE:** At the end of the import the IDE will ask you whether you want to create a project from the checked out sources. **Answer no**. The android-ndk git project contains a large number of separate projects. Instead, once the import is done, go to File > Open..., and navigate to (for example) the Teapot folder and select the file build.gradle in that folder. The project will now be opened and synced. This will take a while the first time as the new Gradle plugin and dependencies are downloaded. When you’re done you can look at the project sources and start exploring the IDE support for Android NDK.  For example, in the Project View you’ll see that the project has a “jni” source set next to the normal “java” source set; that’s where you’ll find the native code:  [http://tools.android.com/_/rsrc/1436483220095/tech-docs/android-ndk-preview/ndk-jni-folder.png](http://tools.android.com/tech-docs/android-ndk-preview/ndk-jni-folder.png?attredirects=0)  You can now build & run the project, and explore the various editing features:   * Code completion * Code navigation (go to declaration, jump between header and implementation, etc) * Quick fixes * Intentions * Refactoring * Source format * …   [http://tools.android.com/_/rsrc/1436483244983/tech-docs/android-ndk-preview/ndk-jni-editor-small.png](http://tools.android.com/tech-docs/android-ndk-preview/ndk-jni-editor-small.png?attredirects=0)  Debugging  To debug the project, you'll need to create a **native** run configuration. Invoke Run > Edit Configurations..., then choose the Native configuration type:  [http://tools.android.com/_/rsrc/1436485240263/tech-docs/android-ndk-preview/ndk-native.png](http://tools.android.com/tech-docs/android-ndk-preview/ndk-native.png?attredirects=0)  Name the configuration, and choose the module to run. If you run into bugs with the LLDB debugger in this version of the IDE, you can try switching to the GDB debugger instead as a workaround.  [http://tools.android.com/_/rsrc/1436485256596/tech-docs/android-ndk-preview/ndk-native2.png](http://tools.android.com/tech-docs/android-ndk-preview/ndk-native2.png?attredirects=0) |

**Experimental Plugin User Guide**

|  |
| --- |
| Introduction  The new experimental plugin is based on Gradle’s new component model mechanism, while allows significant reduction in configuration time. It also includes NDK integration for building JNI applications. This user guides provides details on how to use it and highlights the difference between the new plugin and the original plugin.  **WARNING**: Note that this is plugin is at the experimental stage. The Gradle API for the new component model is not final, which means it’ll only work with a specific version of Gradle until the APIs are final.  Additionally, the DSL is likely change significantly, as APIs to create the DSL are finalized.  This is a very early preview of the plugin for feedback on performance and NDK integration.  Requirements   * Gradle 2.5 only * Android NDK r10e (if you are using NDK) * SDK with Build Tools at least version 19.0.0 and we aim to minimize the amount of changes needed for the migration process in the future. Some features may require a more recent version.   Migrating from Traditional Android Gradle Plugin  A typical Android Studio project may have a directory structure as follows. File that needs to be change is highlighted in red: There are some significant changes in the DSL between the new plugin and the traditional one.  .  ├── app/  │   ├── app.iml  │   ├── build.gradle  │   └── src/  ├── build.gradle  ├── gradle/  │   └── wrapper/  │       ├── gradle-wrapper.jar  │       └── gradle-wrapper.properties  ├── gradle.properties  ├── gradlew\*  ├── gradlew.bat  ├── local.properties  ├── MyApplication.iml  └── settings.gradle  **./gradle/wrapper/gradle-wrapper.properties**   * The new plugin supports only gradle-2.5.   #Wed Apr 10 15:27:10 PDT 2013 distributionBase=GRADLE\_USER\_HOME distributionPath=wrapper/dists zipStoreBase=GRADLE\_USER\_HOME zipStorePath=wrapper/dists distributionUrl=https\://services.gradle.org/distributions/gradle-2.5-all.zip  **./build.gradle**   * Classpath for the plugin is com.android.tools.build:gradle-experimental instead of com.android.tools.build:gradle. * The current version is 0.2.0.   // Top-level build file where you can add configuration options common to all sub-projects/modules.  buildscript {    repositories {        jcenter()    }    dependencies {        classpath 'com.android.tools.build:gradle-experimental:0.2.0'         // NOTE: Do not place your application dependencies here; they belong        // in the individual module build.gradle files    } }  allprojects {    repositories {        jcenter()    } }  **./app/build.gradle**  There are significant changes to the DSL of the plugin.  We understand that many of the changes are frustrating and seem unnecessary, and our goal is to remove some of these current changes to minimize the migration process from the traditional plugin in the future.  DSL Changes:   * Plugin name is com.android.model.application instead of com.android.application. Or use apply plugin: 'com.android.model.library' if you want to create an Android aar library. * Configuration is wrapped with the model { } block * Most properties require the = operator * Adding elements to a Collection should be done using the += operator.   Current DSL Limitations that will hopefully go away:   * buildTypes, productFlavors and signingConfigs must be place outside of the  android { } block. * Nested options within the android { } block must be configured using the with keyword. * Properties are only set with their direct types only, with no way to accept other types and adapting them. For instance:   + Properties of type File accepts only File instead of File and String objects.   + minSdkVersion cannot directly receive either an integer or string (for codename). * Creating a buildType or productFlavor requires calling the create method.  Modifying an existing one such as the release and debug buildType can be done using the just the name. * The DSL for modifying variants and their tasks is very, very limited right now.   apply plugin: 'com.android.model.application'  model {    android {        compileSdkVersion = 22        buildToolsVersion = "22.0.1"         defaultConfig.with {            applicationId =  "com.example.user.myapplication"            minSdkVersion.apiLevel = 15            targetSdkVersion.apiLevel = 22            versionCode = 1            versionName = "1.0"        }     }     android.buildTypes {        release {            minifyEnabled = false            proguardFiles += file('proguard-rules.pro')        }    }     android.productFlavors {        create("flavor1") {             applicationId = ‘com.app’        }    }       // Configures source set directory.     android.sources {         main {             java {                 source {                     srcDir 'src'                 }             }         }     }  }  dependencies {    compile fileTree(dir: 'libs', include: ['\*.jar'])    compile 'com.android.support:appcompat-v7:22.2.0' }  Ndk Integration  The experimental plugin comes with NDK integration for creating native applications.  To use the NDK integration:   * Use the SDK Manager inside Studio to download the NDK. * Set ndk.dir in local.properties or the ANDROID\_NDK\_HOME environment variable to the directory containing the NDK. * Add an android.ndk block to the model in build.gradle.   Known Limitations   * There’s no support for NDK-only modules. The only supported project types are hybrid app projects and hybrid Library Projects. * Consumed Library project don’t impact compilation of jni code in the consuming project (ie the AAR so files are simply packaged in the APK) * No support for creating and depending on static libraries * No support for using a NDK modules like cpu\_features * No support for integrating external build systems.   The build.gradle of a simple NDK application may look like this:  apply plugin: 'com.android.model.application'  model {    android {        compileSdkVersion = 22        buildToolsVersion = "22.0.1"     }     android.ndk {         moduleName = "native"     }  }  \*Note that the moduleName is **required**.  It determines the name of the resulting native library.  By default, it will look in src/main/jni for C/C++ file.  Configure android.sources to change the source directory.  model {    android {        compileSdkVersion = 22        buildToolsVersion = "22.0.1"     }     android.ndk {         moduleName = "native"     }     android.sources {         main {             jni {                 source {                     srcDir 'src'                 }             }         }     }  }  Various build options can be set within the android.ndk { } block.  For example,  model {    android {        compileSdkVersion = 22        buildToolsVersion = "22.0.1"     }     android.ndk {         // All configurations that can be changed in android.ndk.         moduleName = "native"         toolchain = "clang"         toolchainVersion = "3.5"         // Note that CFlags has a capital C, which is inconsistent with         // the naming convention of other properties.  This is a         // technical limitation that will be resolved         CFlags += "-DCUSTOM\_DEFINE"         cppFlags += "-DCUSTOM\_DEFINE"         ldFlags += "-L/custom/lib/path"         ldLibs += "log"         stl = "stlport\_static"     }     android.productFlavors {         create("arm") {             ndk.with {                 // You can customize the NDK configurations for each                 // productFlavors and buildTypes.                 abiFilters += "armeabi-v7a"             }         }         create("fat") {             // If ndk.abiFilters is not configured, the application             // compile and package all suppported ABI.         }     }     // You can modify the NDK configuration for each variant.     components.android {         binaries.afterEach { binary ->             binary.mergedNdkConfig.cppFlags.add(                     "-DVARIANT=\"" + binary.name + "\"")         }     }  }  Samples  Additional samples can be found at <https://github.com/googlesamples/android-ndk>.  Reference  Configuring Gradle Builds can be found at  <https://developer.android.com/tools/building/configuring-gradle.html> |