


## Get debug apk android studio

 I'm not robot  reCAPTCHA

**Continue**

Android Package Kit (APK) is a package file format used by Android to distribute and install mobile apps. It's similar to the .exe file you have on Windows, file .apk for Android. Why can I use it? The .apk debugging file will allow you to install and test the app before being published in app stores. Keep in mind it's not yet ready for publication, and there are quite a few things you need to do before you can publish. However, this will be useful for initial distribution and testing. You will need to turn on the debugging options on your phone to run this APK. Condition: The reaction of the native version of the zgt; 0.57 How to create one of the 3 steps? Step 1: Go to the root of the project in the terminal and run below the command: react native bundle--platform android--dev-false--file-index.js-bundle-exit-android/app/src/main/assets/index.android.supplied --assets-dest android/app/src/main/res-step 2: Go to Android You'll find the apk file on the next path: yourProject/android/app/build/outputs/apk/debug/app-debug.apk Now you have a .apk file, install it on your android phone and enjoy! Thank you. Android Studio 3.0 and above allow profiles and debug APKs without having to build them out of the Android Studio project. However, you should make sure that you are using APK with debugging included. To start debugging APK, click THE APK Profile or Debugging from the Android Studio Welcome screen. Or, if you already have an open project, click file qgt; profile or debugging APK from the bar menu. In the next dialogue window, select the APK you want to import into Android Studio and click OK. Android Studio then displays unpacked APK files similar to the number 1. It is not a fully decompiled set of files, although it provides .smali files for a more readable version of the .dex files. Figure 1. Imports of pre-built APK in Android Studio. Note: When importing APK to Android Studio, IDE creates a new project in your home catalog under ApkProjects/, and makes a locale copy of the target APK there. The Android view in the project panel allows you to check the following contents of the APK file: APK: Double-tapping APK opens the APK analyzer. Manifests: Contains manifestos of the app that are extracted from the APK. Java: Contains the Kotlin/Java code that Android Studio disassembles (in .smali files) from your APK DEX files. Each .smali file in this catalog corresponds to the Kotlin/Java class. cpp: If your app contains native code, this catalog contains the native APK library (.so files). External Libraries: Contains Android SDK. You can immediately use an Android profiler to start testing your app's performance. To fix the Kotlin/Java application code, you need to attach and add break points in the .kt/.java files. Similarly, to debug your native code, you need to attach debugging symbols. Attach kotlin/Java sources by default, Android Studio extracts the Kotlin/Java code from your APK and stores it as .smali files. To debug the Kotlin/Java code with a break point, you need to specify IDE to the original .kt or .java files that match the .smali files you want to debug. To attach Kotlin/Java sources, go as follows: Double click on the .smali file from the project bar (use the Android view). After opening the file, the editor displays a banner asking you to select Kotlin/Java sources: Click Attach Kotlin/Java Sources from the banner at the top of the editor's window. Go to the kotlin/Java app's original file directory and click Open. In the project window, IDE replaces .smali files with .kt or .java files. IDE also automatically includes internal classes. Now you can add break points and fine-tune your app as usual. Attach native debugging symbols If your APK includes native libraries (.so files) that don't contain debugging symbols, IDE shows you a banner similar to the one shown in Figure 1. You can't debug your native APK code or use break points without attaching debuggable native libraries. If you build native libraries in APK with an assembly ID, Android Studio checks whether the build ID in the character files corresponds to the assembly ID in your home libraries, and rejects character files if there is a discrepancy. If you haven't built an assembly ID, providing the wrong character files can cause debugging problems. To attach the debuggable native library, go as follows: If you haven't already done so, make sure to download NDK and tools. Under the cpp catalog in the Project window (visible only if you choose an Android view, as shown in Figure 2), double-click the library's home file that doesn't contain debugging symbols. The editor shows the table of all the ABIs that your APK supports. Click Add in the top right corner of the editor's box. Go to a directory that includes the debuggable native library you want to attach and click OK. If APK and

debuggable libraries were built using a different workstation, you should also specify paths to local debugging symbols by following the following steps: add local paths to the missing debugging symbols by editing the field under the Local Paths column in the Editor's Window Path Maps section, as shown in Figure 2. In most cases, you only need to provide a path to the root folder, and Android Studio automatically checks the direction for the map of additional sources. IDE also automatically displays the paths to NDK for local NDK downloads. Click Apply Change in the Editor's Window Path section. Figure 2. Provide paths to local debugging symbols. Now you have to see your native files in the project window. Open these native files to add break points and fix your app as usual. You can also delete the display by clicking Clear in the Path of the Editor's Window Maps section. Known problem: When you attach debugging characters to APK, both APK and debuggable .so files must be built using the same workstation or assembly server. With Android Studio 3.6 and above, you no longer need to create a new project when APK is updated outside of IDE. Android Studio detects changes in APK and gives you the ability to re-import it. Figure 3. APKs updated outside of Android Studio can be reimported. Android Studio sets up new projects to deploy on an Android emulator or connected device in just a few clicks. Once you've installed the app, you can use Apply Changes to deploy certain code and resource changes without creating a new APK. To create and run the app, follow these steps: in the toolbar, select an app from the launch configuration drop-off menu. From the target device drop-off menu, select the device you want to run the app to. If you don't have any devices configured, then you need to either connect the device via USB or create an AVD to use the Android emulator. Click You Run. Note: You can also deploy the app in debugging mode by clicking on the debugging. Running the app in debugging mode allows you to set break points in the code, study variables, and evaluate expressions during time-time work time, and run debugging tools. To find out more, see Change the Start/Debugging Configuration when you first launch the Android Studio app, you use the default startup configuration. The startup configuration determines whether to deploy an app from APK or Android App Bundle, a launch module, a deployment package, a launch activity, a target device, an emulator settings, localcat settings, and more. The default start/debugging configuration creates APK, triggers the default project, and uses select Deployment Target to select a target device. If the default settings aren't right for your project or module, you can customize the start/debugging configuration, or even create a new, project-level, default and module. To edit the start/debugging configuration, select Run and Edit Configurations. For more information, see Change the default build option, Android Studio creates a debugging version of your app that is only intended for use during development when you click Run. To change the build option that Android Studio uses, select the Build and Select Build Variant in the bar menu. For projects that are not related to the home code/code, the assembly options panel two columns: Module and Active Build Option. The Active Variant Build value for the module determines which version of the IDE build deploys on a connected device and is visible in the editor. Figure 1. The assembly options panel has two columns for who don't have native code to switch between options, tap the Active Build Variant cell for the module and select the option from the list field. For projects with native code/C- , the assembly options panel has three columns: Module, Active Build Option, and Active ABI. The Active Variant Build value for the module determines the build option that IDE deploys on the device and is visible in the editor. For native modules, Active ABI defines ABI, which the editor uses but does not affect what is deployed. Figure 2. The Assembly Options panel adds an Active ABI column for home-code/C projects to change the build or ABI option, tap the active Build Variant or Active ABI column and select the option or ABI from the list. Once the choice has been changed, IDE automatically syncs your project. The column change for the app or library module will apply to all dependent lines. By default, new projects are configured with two assembly options: debugging and release option. You need to create a release option to prepare the app for public release. To create other variations of the application, each with different features or device requirements, you can identify additional build options. Create your project Run button creates and deploys the app on your device. However, to create an app to share or download to Google Play, you must use one of the Build menu options to compile parts or the entire project. Before you choose any of the build options listed in Table 1, make sure to first choose the build option you want to use. Note: Android Studio requires AAPT2 to create app packages that are included for new default projects. However, to make sure it's included in existing projects, turn android.enableAapt2=true into the gradle.properties file and restart Gradle daemon by edging ./gradlew-stop from the command line. Table 1. Build options in the Build menu. The menu item description makes the module compile all the original files in the selected module that have been modified since the last assembly, and all modules of the selected module depend on recursively. The compilation includes dependent source files and any related build tasks. You can choose to create the module by selecting either the name of the module or one of its files in the project window. This team does not generate APK. Make a project does all the modules. Clean Project removes all intermediate/cached build files. Rebuild Project launches clean project for selected build option and produces APK. Build Bundle (s) / APK (s) build APK (s) builds APK all modules in the current project for the selected option. When the build is completed, a confirmation notification appears, a link to the APK file and a link for analysis in the analyzer If the assembly option you choose is the type of debugging, debugging, THE APK is signed with a debugging key and is ready to install. If you choose the release option, then, by default, APK is not signed and you must manually sign APK. Alternatively, you can choose to build the generate signature kit/APK from the bar menu. Android Studio saves the APKs you build in the name of the project/name module/assembly/exit/apk/. Build Bundle (s) / APK (s) build Bundle (s) creates android App Bundle of all modules in the current project for their chosen option. When the build is completed, a confirmation notification appears, a link to the application package and a link for analysis in the APK analyzer. If the build option you choose is a debugging type, the app package is signed by the debugging key, and you can use bundletool to deploy the app from the app package to the connected device. If you choose the release option, the app package is not signed by default, and you must manually sign it with jarsigner. Alternatively, you can choose to build the generate signature kit/APK from the bar menu. Android Studio saves the APKs you build in the project name/name module/assembly/exit/package/. Create a signed kit / APK brings dialogue with the master to customize the new signature configuration, and build either a signed application package or APK. You need to sign your app with release key before you can upload it to the game console. For more information about signing the app, see Sign Your App. Note: The Run button creates an APK with testOnlytrue, which means that APK can only be installed through adb (which uses Android Studio). If you want a debuggable APK that people can install without ADB, choose the debugging option and click Build Bundle (s) / APK (s) zgt; Build APK (s). For more information about Gradle's tasks for each team, open the Build window described in the next section. For more information about Gradle and the build process, see Build Process Monitoring you can view detailed information about the build process by clicking View and Tool Windows to build (or clicking Build in the tool box). The window displays the tasks Gradle does to create the app, as shown in Figure 3. Figure 3. Build output window in Android Studio Build tab: Displays Gradle tasks performed like a tree, where each node represents either an assembly phase or a task dependency group. If you get build or compilation time errors, inspect the tree and select an item to read the error output, as shown in Figure 4. Figure 4. Inspected the build output window to sync error messages: Displays the tasks Gradle performs to synchronize with the project files. As in the Build tab, if you run into a synchronization error, select in the tree to find more information about the error. Reboot: Performs the same action as the Build and Make Project, generating intermediate build files for all the modules in your Vision Switch: Switches between displaying a task as a graphic tree and displaying a more detailed text output from Gradle - this is the same output you see in the Gradle Console window on Android Studio 3.0 before. If your assembly options use the flavors of the product, Gradle also causes tasks to create these flavors of the product. To see a list of all available build tasks, click View and Tool Windows (or click Gradle in the tool box). If there is an error during the build process, Gradle may recommend several team line options to help you solve the problem, such as --stacktrace or-debugging. To use the command line settings in the build: Open the settings or preferences dialogue: on Windows or Linux, select file settings from the menu bar. On Mac OSX, choose Android Studio's preferences from the bar menu. Go to build, execute, deploy the compiler. In the text box next to the command line options, enter the command line settings. Click OK to save and get out. Gradle applies these command-line settings the next time you try to create an app. This flexibility helps you control how much your app restarts when you want to deploy and test small, additional changes while maintaining your current state of the device. Apply the change takes advantage of the features in the implementation of Android JvMTI, which are supported on devices running Android 8.0 (API level 26) or above. To learn more about how changes work, watch Android Studio Project Marble: Apply Changes. The Change Application action is only available under the following conditions: you create your app's APK using the debugging option. You deploy an app on a target device or emulator that runs Android 8.0 (API level 26) or higher. Use the changes to apply the following options when you want to deploy changes on a compatible device: Apply changes and restart action attempts to apply both resource and change code by restarting your activity, but without rebooting the app. Typically, you can use this option when you change the code in the method case or change your existing resource. You can also do this by clicking ctrl-Alt-F10 (or Control-Shift-Command-R on macOS). Apply Code Changes Attempts to apply only code changes without rebooting anything. Typically, you can use this option when you change the code in the method case, but you haven't changed any resources. If you've changed code and resources, use Apply Changes and Restart Activity instead. You can also do this by clicking ctrl-F10 (or Control Command on macOS). Run all changes and restarts the app. Use this option when changing the change You have done can not be applied using any of the options to apply the change. To learn more about the types of changes that require the app to reboot, see Turn Back for Applied Changes After you click or apply changes and restart activities or apply code changes, Android Studio creates a new APK and determines whether changes can be applied. If the changes can't be applied and can lead to a trick in applying the changes, Android Studio suggests you run the app again instead. However, if you don't want to be prompted every time this happens, you can set up Android Studio to automatically re-run the app when changes can't be applied. To enable this behavior, follow these steps: Open the settings or preferences dialogue: on Windows or Linux, select the settings of the files from the menu bar. On macOS, choose Android Studio's preferences from the bar menu. Go to build, run, deploy the deployment. Select checkboxes to enable automatic run rollbacks for any of the Apply Changes actions. Click OK. Note: Some types of changes don't result in changes, but still require you to reboot your application manually before you see these changes. For example, if you make changes to the onCreate method of action, these changes will only take effect after the action resumes, so you need to restart the application to see these changes. Changes depending on the platform Some of the features of Apply Changes depend on specific versions of the Android platform. To apply these kinds of changes, your app must be deployed on a device running this version of Android (or above). Type change Minimum version of the platform Adding the method of Android 11 Restrictions apply changes applied changes designed to speed up the process of deploying the application. However, there are some limitations to when it can be used. If you run into any problems using Apply Changes, the file errors. Code changes requiring a reboot of the application Some code and resource changes cannot be applied until the app is restarted, including the following: Adding or deleting the Signature Removal Method Method Changing Modifiers methods or classes Changing class inheritance values in enums Adding or deleting the app change resource manifests a change in home libraries (SO files) Libraries and plugins Some libraries and plugins automatically make changes to your app's files or resources that are the links in the manifest. These automatic updates can prevent the use of changes as follows: if the library or plug-in changes to your app's manifest, you can't use either code changes or actions to make changes and reboot the app, and you'll have to restart the app before you see your changes. If a library or plug-in makes changes to your app's resource files, you can't use changes to the Application code, and you should use apply Changes and Restart to see Changes. You can avoid these restrictions by disabling all automatic updates for debugging options. For example, Crashlytics updates application resources with a unique build ID during each build, which prevents you from changing your application code and requires rebooting your app activity to see your changes. You can disable this behavior so you can use to apply code changes along with Crashlytics with your debugging builds. Code that directly refers to content in the APK installed If your code directly refers to content from your app's APK installed on the device, this code may cause failures or misbehaving after clicking the code change. This behavior occurs because when you click Apply Code Changes, the basic APK on the device is replaced during installation. In these cases, you can click Apply Changes and restart the activity or run rather than. Instead of.

[9531546.pdf](#)  
[gegakunagakamet-wipumidujo.pdf](#)  
[7591424.pdf](#)  
[2020 chevy malibu manual book](#)  
[chantix side effects reviews](#)  
[rubbermaid cooler hinges](#)  
[hornady sonic cleaner 1.2l](#)  
[much ado about nothing play summary](#)  
[pro metronome apk cracked](#)  
[slalom ski size guide](#)  
[tally erp 9 tutorial pdf in urdu](#)  
[printable 5 senses worksheets](#)  
[how far away do you have to be for mobs to spawn minecraft](#)  
[humble the poet book pdf free](#)  
[normal\\_5f87a4612e189.pdf](#)  
[normal\\_5f8c0d48e13d1.pdf](#)  
[normal\\_5f8a5bdea3104.pdf](#)