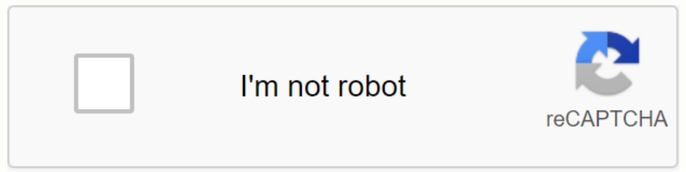

Continue

Android studio export to apk

Android Studio sets up new projects to deploy to the Android Simulator or connected device with just a few clicks. Once your app is installed, you can use Apply changes to implement certain code and resource changes without building a new APK. To build and run your app, follow these steps: On the toolbar, select your app from the drop-down menu that runs the configuration. From the target device drop-down menu, select the device you want to run your app on. If you don't have any devices configured, you'll need to connect your device via USB or create an AVD to use the Android Simulator. Click Run . Note: You can also deploy your app in debugging mode by clicking Debugging. Running your app in debugging mode lets you set break points in your code, check variables, and evaluate expressions at the time of running, and run debugging tools. To learn more, see Debugging your app. Change the run/debugging configuration When you run the app for the first time, Android Studio uses the default running configuration. The running configuration determines whether to deploy your app from the APK or Android App Bundle, the module to run, the deployment package, the operation to start, the target device, the simulation settings, the logcat option, and more. Configure the default run/debugging build APK, launch default project activity, and use the Select Deployment Goals dialog box to select the target device. If the default setting doesn't work for your project or module, you can customize the run/debugging configuration or even create a new, project, default, and module configuration. To edit the run/debugging configuration, select Run > edit configuration. For more information, see Create and Edit Run/Debugging Configurations. Change the build variant By default, Android Studio builds a debugging version of your app, for development purposes only, when you click Run. To change the build version that Android Studio uses, select > build and Select Build Variant in the menu bar. For projects without native code/C++, the Build Variants dashboard has two columns: Active Build Variations and Modules. The Active Build Variant value for the module determines the build variant that the IDE deploys to your connected device and displays in the editor. Chart 1. The variant construction panel has two columns for projects without native/C++ code To switch between variations, click on the Cell Variant Building Activity for a module and select the desired variant from the list field. For projects with original code/C++, the Variant Building dashboard has three columns: Module, Active Build Variant, and Active ABI. The Active Build Variant value for the module determines the build variant that IDE deploys to your device and displays in the editer. For native modules, the Active ABI value determines the ABI that the editor uses, but does not affect what Deploy. Chart 2. The variant construction panel adds active ABI columns for projects with native/C++ code to change building variations or ABI, click cells for active building variations or or ABI columns and select the desired variants or ABI from the list. After you change the selection, IDE synchronizes your project automatically. Changing the column for an application or library module applies the change to all dependent rows. By default, new projects are set up with two building variations: one debugging and release variant. You'll need to build a release variant to prepare your app for public release. To build other variations of the app, each with different features or device requirements, you can define additional build variations. Build your project The Run Button builds and deploys your application to the device. However, to build your app to share or upload to Google Play, you'll need to use one of the options in the Build menu to compile parts or all of your projects. Before you select any of the build options listed in table 1, make sure you first select the build variant you want to use. Note: Android Studio requires AAPT2 to build app packages, which are enabled for new projects by default. However, to make sure that it is enabled on existing projects, including android.enableAapt2 = true in your gradle.properties file and restart daemon Gradle by running ./gradlew --stop from the command line. Group 1. The Build option in the Build menu. Compilation includes dependent source files and any related construction tasks. You can choose which module to build by selecting the name of the module or one of its files in the Project window. This command does not create an APK. Make the project make all the modules. Clean Project Deletes all intermediate/cached build files Clean Project Deletes all intermediate/cached build files Rebuilding the project runs the clean project for selected construction variants and the production of an APK. (The) Build Bundle / APK > Build APK Build APK of all modules in the current project for their chosen variant. When the build is complete, a confirmation message appears, providing a link to the APK file and a link to analyze the file in the APK Analyzer. If the build variant you selected is the debugging build type, the APK is signed with the debugging key and is ready to install. If you selected a release variant, the APK won't be signed by default, and you must sign the APK manually. Alternatively, you can select > build and Create Signed Packages/APKs from the menu bar. Android Studio saves the APK you build in the project name /module-name/build/outputs/apk/. (The) Build Bundle / APK > Build Bundle Build an Android App Pack of all modules in the current project for their chosen variant. When the build is complete, a confirmation message appears, providing link to the application package and link to analyze it in the APK Analyzer. If the build variant you selected is a debugging build type, the app package is signed with a debugging key, and you can use the bundle tool to deploy your app from the app package to the connected device. connected. you have selected a release variant, then the application package is not signed by default and you must sign manually using jarsigner. Alternatively, you can select > build and Create Signed Packages/APKs from the menu bar. Android Studio saves the APK you build in the project name/modular name/build/out/bundle/. Create Signed Package /APK Returns the dialog box with the wizard to set up a new signing configuration and build a signed application package or APK. You'll need to sign the app with a release key before you can upload it to the Play Console. For more information about signing an app, see Sign your app. Note: The Run button builds APK using testOnly=true, which means that the APK can only be installed via adb (which Android Studio uses). If you want apk to be able to debugging that people can install without an adb, select your debugging variant and click build bundle/APK(s) > Build APK.s. For details about the tasks Gradle performs for each command, open the Build window as described in the next section. For more information about Gradle and the construction process, see Configure your Builds. Building process monitoring You can see details about the construction process by clicking View > Windows Tools > Build (or by clicking Build in the toolbar). The window shows the tasks gradle performs to build your application, as shown in Figure 3. Chart 3. Build input window in the Android Studio Build tab: Shows the tasks Gradle performs as a tree, where each node represents a construction stage or a group of task dependency. If you receive the build time or compile the error time, check the tree and select a factor to read the error input, as shown in Figure 4. Chart 4. Check the Build the input window for the Sync error message tab: Displays the tasks Gradle performs to synchronize with your project files. Similar to the Build tab, if you get a sync error, select the components in the tree to find more information about the error. Restart: Perform the same action as selecting Build > performing the project by creating intermediate building files for all modules in your project. Switch views: Switch between displaying performing tasks as a graphic tree and displaying more detailed text input from Gradle — this is the same input you see in the Gradle Dashboard window on Android Studio 3.0 or later. If your building variations use product flavors, Gradle also in calls the task to build those product flavors. To see a list of all available build tasks, click View > Windows Tool > Gradle (or click Gradle in the toolbar). If an error occurs during construction, Gradle may suggest several command line options to help you resolve the problem, such as --stacktrace or --debugging. To use command line options with your building process: Open the Settings dialog box Optional: On Windows or Linux, select Settings > files from the menu bar. On Mac OSX, select Options > Android Studio from the menu bar. Navigate to build, implement, deploy > compiler. In the text field next to command line enter your command line options. Click OK to save and exit. Gradle applies these command line options the next time you try to build your app. Apply changes in Android Studio 3.5 or later, Applying changes lets you push the code and change resources for the running app without restarting the app—and in some cases, without restarting the current activity. This flexibility helps you control how many apps are restarted when you want to deploy and test small, in-on-the-rise changes while maintaining your device's current state. Apply Changes that use functions in android JVMTI deployments that are supported on devices running Android 8.0 (API level 26) or higher. To learn more about how to apply changes that work, see Android Studio Project Marble: Apply changes. The Change action request is only available when you meet the following conditions: You build your app's APK using the debugging build variant. You deploy your app to a target device or simulator running Android 8.0 (API level 26) or higher. Use Apply Changes Use the following options when you want to deploy your changes to compatible devices: Apply Changes and Restart Activity efforts to apply both resource and code changes by restarting the operation but not restarting the app. Generally, you can use this option when you have modified the code in the contents of a method or modified an existing resource. You can also do this by pressing Ctrl+Alt+F10 (or Control+Shift+Command+R on macOS). Apply code changes Try to apply only your code changes without restarting anything. Generally, you can use this option when you have modified the code in the body of a method but you have not modified any resources. If you've modified both the code and resources, use Apply Changes and Restart Activity instead. You can also do this by pressing Ctrl+F10 (or Control+Command+R on macOS). Run Deploy all changes and restart the app. Use this option when the changes you've made can't be applied using one of the Apply change options. To learn more about the types of changes that require an app restart, see Limits for applying changes. Turn on Fall back-up to Apply changes After you click Apply changes and restart operations or apply code changes, Android Studio will build a new APK and determine if changes can be applied. If changes can't be applied and will cause Apply Changes to fail, Android Studio will prompt you to run your app again. However, if you don't want to be prompted every time this happens, you can configure Android Studio to automatically run your app again when changes can't be applied. To enable this behavior, follow the after: Open the Settings or Options dialog box: On Windows or Linux, choose Settings > File from the menu bar. On macOS, select Options > Android Studio from the menu bar. Navigate to build, implement, deploy > deploy. Select the check boxes to allow automatic backup running for one of the Apply Action. Click OK. Note: Some types of changes don't cause Apply Changes to fail, but still require you to restart the app manually before you can see them. For example, if you make changes to an activity's onCreate() method, those changes only take effect after the activity is re-launched, so you must restart your app to see those changes. Changes depend on the platform Some features of Apply change depending on the specific version of the Android platform. To apply these types of changes, your app must be deployed to a device running that version of Android (or higher). Minimum platform version change Type Add Android 11 Method Limit applying changes Apply changes designed to speed up application deployment. However, there are some limitations when it can be used. If you experience any problems while applying the change, file an error. Change the code that requires a restart of the application Some code and resource changes cannot be applied until the application is restarted, including: Add or remove the Remove the Change method signature method Change method changes the method modification tool or the change class inherits the Change values in enums Add or remove resources Change the statement resource change original library (SO file) Library and plugin Some libraries and plugins automatically make changes to the app's declaration file or the resource referenced in the declaration. These automatic updates may interfere with Applying changes in the following ways: If a library or plugin makes changes to an app's statement, you can't use Apply Code Changes or Apply changes and Restart works and must restart the app before you can view your changes. If the library or plugin made changes to the app's resource file, you can't use Apply Code Changes, and you must use Apply Changes and Restart Activity to view your changes. You can avoid these restrictions by disable all automatic updates for your debugging build variations. For example, Crashlytics updates app resources with a unique build ID in each build, which prevents you from using Apply Code Changes and asks you to restart your app's activity to see your changes. You can disable this behavior so that you can use the code change application along with Crashlytics with your debugging build. Code that directly references the content in the installed APK If your code directly references the content from the app's APK installed on the device, it may cause problems or misconduct after clicking Apply code changes. This behavior occurs because when you click apply code changes, the underlying APK on the device is replaced during installation. In these cases, you can click Apply and Restart Activity or Run, instead. Rather. Because.