Android studio kotlin gradle

There have been quite a few months already since Gradle announced that they are working to support Cotlin to write Gradle scripts, using a version of the language that was recently updated to Kotlin DSL. At the beginning of things where quite difficult, but now, with the latest versions of Kotlin DSL (at the time of writing this version 0.12) the idea is more mature. So, knowing how, it's not too hard to start using Kotlin to create Gradle files in Android. One of the main problems with this is the lack of documentation, so I decided to write about my experience converting Gradle Bandhook-Kotlin files so that you can reproduce it in your project. If you're still starting with Kotlin, you may be interested in checking out my previous articles about Kotlin. UPDATE: I recently posted a YouTube video after the Process of Converting the Gradle file from Groovy to Cotlin, you can take a look: Using Kotlin DSL on your Gradle files. Is it worth it? I think this is the first issue to solve. To date, should I spend my time converting my files into Kotlin DSL? Check out my free guide to create my first project in 15 minutes! My answer is probably a little counterproductive to encourage you to keep reading this article, but I don't want you hyped because of this: you probably shouldn't. These are of course some pluses: You can use language that you're more familiar with, so it's easier to start doing more complex things. I never did anything on the buildSrc folder and it was pretty easy for me to create my own class and use it in the rest of the script file. IDE will help you a lot more: a good autocomperation, errors are detected by the compiler, imports are added automatically... Everything you know and love from your usual Kotlin code is extrapolated here. But there are also some downsides: There's no easy way to convert from Groovy to Cotlin files. I'll explain to you how to make it easier, although you need to know how the plugin is implemented to be able to use it: where in Groovy you just use an equal assign value for each configuration, here you need to know whether it is a feature or a property to know how to install it. Glad that IDE can help. The Gradle team says this will only be resolved when people write plugins thinking well on Kotlin DSL. There are some rules to follow. Example (we'll see more later): applicationId and Config.Android.applicationId minSdkVersion (Config.Android.minSdkVersion) targetSdkVersion (Config.Android.targetSdkVersion) versionCode and Config.Android.versionName and Config.Android.versionName There's not much documentation or examples: I think that's the main problem. If you get stuck, it's hard to continue. It took me quite a while (and asking for a big Cotlin Slack) to know how to make Things. So the big bounce here: be sure in you do if you use it in the production code. It can be an interesting thing to do in your pet projects though. I'm not saying it's not mature enough, just that it's not easy to use. As you convert your files I want to give you here the fastest route, missing all the pain points I had to solve. So if I had to convert another project to use Kotlin on gradle files, that's what I would do. Use the latest version of the Gradle new version of Gradle, the better, because it will include the latest version of kotlin DSL. When I converted this project, the last one was 4.5.1. You can check out the latest release here. Change your file to gradle-wrapper.properts to use it: distributionBase-GRADLE_USER_HOME distributionPath'wrapper/dists zipStoreBase-GRADLE_USER_HOME zipStorePath'wrapper/dists distributionUrl' change I must admit that since the first time I've tried everything. You used to need to add a lot of configurations that you don't need when using Groovy. Now you just need to add an extension to the build.gradle files, and Gradle will be able to use Kotlin files. Rename them to build.gradle.kts. Compiling using the IDE terminal won't help you much here in understanding what happened, so I recommend you use the cmd and -info flag: ./gradlew to collectDebug--info With this, you get a better idea of things that don't work. You can do it now if you want to, but it obviously won't succeed. Set up the buildSrc folder One of the pain points that I found was a way to reproduce an ext object in Groovy, allowing you to exchange variables between Groovy files: ext / Android configuration androidBuildToolsVersion - 27.0.3 ... You can have that in your Root Gradle file and then use it in your modules: buildToolsVersion parent.ext.androidBuildToolsVersion It's pretty easy and works fine. The alternative to Kotlin DSL is that for every variable you need to create an extra like this: var androidBuildToolsVersion: Line on extra androidBuildToolsVersion - 27.0.3 And then you use it with this: val androidBuildToolsVersion: A string of extra buildToolsVersion (androidBuildToolsVersion) As you can imagine, it doesn't scale very well. Having all this code for each variable is a pain. So the alternative I found is to create a configuration file in buildSrc, and add everything you need to an object that you can instantly in any Gradle files. If you don't know about it, buildSrc is basically the place where you put all the code you want to use when creating project scripts. You can find details at Gradle Docs. To set it up, just create this folder structure under the buildSrc folder: Forget about .gradle and create folders and create the rest. Under this folder, also create a new build.gradle.kts with this content: content: The Kotlin-edle Config File will use variables that can be used in the project. You can use any structure you want. While searching for information on how to do this, I found a repository from Arturo Gutierrez that uses this structure and I liked it. I moved to use objects rather than classes though, which makes more sense here: the Config object and the Object BuildPlugins object Android object Libs object TestLibs - Then, each child object has its own meanings. For example, Android one: Android object and val buildToolsVersionVersion - 27.0.3 val minSdkVersion - 19 val targetSdkVersion - 27 val compileSdkVersion - 27 val-appId com.antonioleiva.bandhookkotlin val versionCode - 1 val versionName - 0.1 - You can also use some top variables, To facilitate editing: private const val supportVersion - 27.0.2... Object Libs - val appcompat - com.android.support:appcompat-v7:$supportVersion val recyclerview - com.android.support:recyclerview-v7:$supportVersion val cardview com.android.support:cardview-v7:$supportVersion val palette - com.android.support:palette-v7:$supportVersion $supportVersion Then using it in build.gradle files is pretty simple: defaultConfig and appId and Config.android.applicationId minSdkVersion (Config.Android.minSdkVersion) targetSdkVersion (Config. Android.targetSdkVersion) version Code - Config.Android.versionCode versionImage - Config.Android.versionImsim testInstructionRunner - android.support.test.runner.AndroidJUnitRunner it looks much cleaner. Keep converting Gradle files until the first time it fully compiles, you'll have to rely on what the terminal builds to say, IDE won't be very useful here. So keep changing the details a little timely, building a project and interpreting the output. As a link, I can leave you some parts of the Gradle files here (and you can of course check out the full project on Github). For root build.gradle: buildscript - repository - jcenter () Google () - dependency - classpath (Config.BuildPlugins.androidGradle) classpath (Config.BuildPlugins.kotlinGradlePlugin) File module is a little more complicated. For plugins, you do it this way: plugins and id (com.android.application) kotlin (android) kotlin (kapt) - For regular plugins, you just use the ID feature, and Kotlin plugins use the kotlin feature. Then android section, as you saw above. You should try to find out whether it is a feature or a property. Check out the repository of the most typical ones. Then, for build types: buildTypes - getByName (release) - thisMinifyEnabled - false You can't just create a release block, but instead it has to find it by name and then you can customize it. Addictions are simple, just a function where you set the name of addiction: dependency and compilation (Config.Libs.kotlin_std)... Continue to build and polish until the assembly is successful. It's not easy, but it's cool! It's really awesome to see how Kotlin reaches all the developmental environments: JVM, JS, Gradle... and potentially everywhere with Kotlin/Native. This is just another example of the versatility of the language, and gives an insight into how enthusiastic different development communities are becoming about it. In the case of Gradle, maybe it's not yet production ready (although I know people who use it without many problems), but it's worth giving it a try and checking out how well it works as soon as everything is set up. Making time errors and auto-seing helps a lot when we build our Gradle files, which otherwise requires just tight memory or searching. What do you think of Kotlin DSL? Have you tried it? Do you use it in your projects? Let me know in the comments. Comments. android studio kotlin gradle plugin version. android studio gradle kotlin dsl. android studio update kotlin gradle plugin. android studio could not download kotlin-gradle-plugin.jar. android studio kotlin build.gradle