I'm not robot
reCAPTCHA
Continue

Android studio gradle show dependencies

The picture in the query doesn't really show a tree, just a flat list of everything compiled in the app. Are you using Gradle? If so, you can really see the tree by running a Gradle command Android documentation: View the Dependency Tree GUI Select View > Tool Windows > Gradle (or click the Gradle icon on the toolbar). Expand AppName > Tasks > android and double-click androidDependencies. After Gradle has executed the task, the Run window should open to display the output. CLI (produces tree-like list) ./gradlew app:dependencies and/or (produces flat list) ./gradlew app:androidDepending Where the app is your module's name And you get something like that +--- MyApp:mylibrary:unspecified | \--- com.android.support:appcompat-v7:25.3.1 | +--- com.android.support:animated-vector-drawable:25.3.1 | | \--- com.android.support:support-vector-drawable:25.3.1 | | \--- com.android.support:support-v4:25.3.1 | | \--- LOCAL: internal_impl-25.3.1.jar | +--- com.android.support:support-v4:25.3.1 | | \--- LOCAL: internal_impl-25.3.1.jar | \--- com.android.support:support-vector-drawable:25.3.1 | \--- com.android.support:support-v4:25.3.1 | \--- LOCAL: internal_impl-25.3.1.jar \--- com.android.support:appcompat-v7:25.3.1 +--- com.android.support:animated-vector-drawable:25.3.1 | \--- com.android.support:support-vector-drawable:25.3.1 | \--- com.android.support:support-v4:25.3.1 | \--- LOCAL: internal_impl-25.3.1.jar +--- com.android.support:support-v4:25.3.1 | \--- LOCAL: internal_impl-25.3.1.jar \--- com.android.support:support-vector-drawable:25.3.1 \--- com.android.support:support-v4:25..25.25.1 3.1 \--- LOCAL: internal_impl-25.3.1.jar Note: If you are running LS (or dir on Windows) in that folder, and do not see degreelew (degree or degree.bat), you are in the wrong folder. Android applications are built using open source Gradle building system. Gradle is a state-of-the-art API that easily supports customizations and is widely used in the Java world. The Gradle Android plug-in adds a wide range of features specific to Android apps, including build types, flavors, signing configurations, library projects, and more. The recipes in this book cover the range of Gradle features when applied to Android projects. Since Android Studio IDE uses Gradle under the hood, special recipes are dedicated to it as well. Hopefully the recipes in this book will help you configure and build whatever Android application you desire. You want to understand the generated Gradle build files for a new Android application. Create a new Android project with Android Studio and review the settings.gradle, build.gradle, and app/build.gradle files. Android Studio is the only officially supported IDE for Android projects. To create a new Android project using Android Studio, use the Start a new Android Studio project wizard (Figure 1-1). The wizard prompts you to specify a project name and domain. You can use the Quick Start Wizard to start a new Android my android app in domain, as shown in Figures 1-2. From here, just select the Phone and Tablet option and add an empty activity with the default name MainActivity. The name and type of activity do not affect the Gradle build files. The resulting Project view in Android mode is shown in Figures 1-3, where the relevant Gradle files are selected. The project layout in the default view (Project) appears in Slides 1-4. Android project is multiproject Gradle builds. The settings.gradle file shows which subdirectories hold their own subprojects. The default file contents are shown in example 1-1. The Include statement indicates that the app subdirectory is the only additional subproject. If you add an Android Library project, it will also be added to this file. The top-level Gradle build file is in example 1-2. Top-level build file where you can add configuration options // common to all subprojects/modules. buildscript { repositories { jcenter() } dependencies { classpath 'com.android.tools.build:gradle:2.0.0' // NOTE: Do not place your application dependencies here; they belong // in the individual module build.gradle files } } all projects { { jcenter() } } task-by-case(type: Delete) { rootProject.buildDir } Gradle distribution does not include Android functionality by default. Google provides an Android plug-in for Gradle, which allows for easy configuration of Android projects. The buildscript block in the top build file tells Gradle where to download that plug-in. As you can see, the plug-in is downloaded by jcenter by default, which means bintray JCenter Artifactory repository. Other databases are supported (especially mavenCentral(), the default maven repository), but JCenter is now the default. All JCenter content is served over a CDN with a secure HTTPS connection. It also tends to be faster. The allprojects section specifies that the top-level project and any subprojects all default to use the jcenter() repository to resolve any Java library dependencies. Gradle allows you to define tasks of your own and insert them into the targeted acyclic graph (DAG), which Gradle uses to resolve task relationships. Here, a clean task has been added to the top level build. Type: Delete the part indicates that the new task is a custom instance of the built-in Delete task from Gradle. In this scenario, the task removes the build directory from the root project, by default, to a top-level build folder. The Gradle build file for the app subproject is shown in examples 1-3. apply plugin: 'com.android.application' android { compileSdkVersion 23 buildToolsVersion 23.0.3 defaultConfig { applicationId com.kousenit.myandroidapp minSdkVersion 19 targetSdkVersion 23 versionCode 1 versionName 1.0 } buildTypes { release { minifyEnabled false proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro' } } dependencies } } of the type compile fileTree(dir: 'libs', include: testKompileres testKompileres compile 'com.android.support:appcompat-v7:23.3.0' } Applying the functionality of Gradle adds the Android plug-in to the build system, which enables configuration for the Android Section Domain Specific Language (DSL). This section is discussed in detail in Recipe 1.2. The dependency blocks consist of three rows. The first, fileTree dependent, adds all files that stop .jar to the libs folder to the compilation class path. The second line tells Gradle to download version 4.12 of JUnit and add it to the test compile phase, which means that JUnit classes will be available in src/androidTest/java source trees, as well as (optional) src/test/java trees, which can be added for pure device testing (i.e. those that don't involve android API). The third line tells Gradle to add version 23.3.0 of the appcompat-v7 can files from the Android support libraries. Note that -v7 means support for Android applications back to version 7 of Android, not version 7 of the support library itself. The support library is set as a compilation dependency, so all of its classes are available throughout the project. For links to all relevant documentation sites, see Recipe 6.2. Dependencies are discussed in Recipe 1.5 and repositories are discussed in Recipe 1.7. You want to set the minimum and target Android SDK versions and other default properties. In the Gradle build file module, enter values in the android block. The top-level Android build file adds the Android plug-in for Gradle to your project, via the buildscript block. Module build files apply plug-in, which adds an android block to Gradle DSL. Inside the Android block, you can set multiple project properties, as shown in Example 1-4. apply plugin: 'com.android.application' android { compileSdkVersion 23 buildToolsVersion 23.0.3 defaultConfig { applicationId com.kousenit.myandroidapp minSdkVersion 19 targetSdkVersion 23 versionCode 1 versionName 1.0 } compileOptions { sourceCompatibility JavaVersion.VERSION_1_7 targetCompatibility JavaVersion.VERSION_1_7 } } Common Java plugin uses a java plugin application, but Android project plugin uses plugin application com.android.application instead. Do not apply the Java plug-in. This will cause build errors. Use the Android plug-in instead. The Android block is the entry point for Android DSL. Here you must specify the compilation target by using compileSdkVersion and the build tool version via buildToolsVersion. Both of these values should be assigned to the latest available version, because they are backwardcompatible and include all current bug fixes. The default blockConfig inside android displays several characteristics: applicationId Application Package Name, which must be unique in the Google Play store. This value can never change during your app's lifespan; if you change it, it will result in your app being treated as a brand new application, and existing users will not see changes as an update. Before moving to Gradle, this was the package of the root element in Android Manifest. The two can now be decoupled. minSdkVersion The smallest Android SDK version supported by this application. Devices earlier than that don't see this application when they access the Google Play store. targetSdkVersion The version of Android intended for this application. Android Studio will issue a warning if this is anything other than the latest version, but you are free to use any version you want. versionCode An integer representing this version of your app in relation to others. Apps typically use this during the upgrade process. versionName A string that represents the version of your app for the release, which is displayed to users. Normally in the form of a <major>.<minor>. <version>string, like most projects. Prior to the transition to Gradle, the minSdkVersion and buildToolsVersion properties were set in Android Manifest as attributes for a <uses-sdk>tag. That approach is now clouded because the values there are overridden by the values in the gradle build file. The compileOptions section shows that this app expects to use JDK version 1.7. In Android Studio, the Project Structure dialog box displays the values in graphic form, as shown in Figures 1-5. The defaultValues ForConfig are located on the Flavors tab of the Project Structure window (figures 1-6). Documentation for the default Config block, as with other elements of DSL, is available in the DSL reference. Other child elements of android, such as buildTypes or productFlavors, are discussed in Recipes 3.1, 3.2, 3.4, and more. For documentation links, see Recipe 6.2. You want to run Gradle tasks from the command line. From the command line, either use the included Gradle cover or install Gradle and run it right away. You don't need to install Gradle to build Android projects. Android Studio comes with a Gradle distribution (in the form of a plug-in) and includes dedicated features to support it. The term Gradle wrapper refers to the unix gradlew script and the gradlew.bat script in the root directory of an Android application, where the w-ending stands for wrapper. The purpose of the Gradle cover is to allow a client to run Gradle without having to install it first. The cover element uses the gradle-wrapper.jar and gradle-wrapper.properties files in the gradle/wrapper folder in the application root to start the process. A sample of the property file is shown in example 1-5. #Mon Dec 28 10:00:20 PST 2015 distributionBase=GRADLE_USER_HOME distributionPath=wrapper/dists zipStoreBase=GRADLE_USER_HOME zipStorePath=wrapper/dists distributionUrl=https\://services.gradle.org/distribution/gradle-2 10-all.zip DistributionUrl property indicates that the wrapper will download and install version 2.10 of Gradle.1 After the first run, The Gradle distribution will be cached in the zipStorePath folder under the zipStoreBase directory and then available for all subsequent executions of Gradle tasks. The cover is used on helt enkelt genom att exekvera</uses-sdk> </version> </minor> </major> </major> unix command or the gradlew command.bat in Windows (example 1-6). > ./gradlew build Load ..... (download of Gradle 2.10) .... Unzipping /Users/kousen/.gradle/wrapper/dists/3i2gob.../gradle-2.10-all.zip to /Users/kousen/.gradle/wrapper/dists/gradle-2.10-all/3i2gob... Set executable permissions for: /Users/kousen/.gradle/wrapper/dists/gradle-2.10-all/3i2gob.../gradle-2.10/bin/gradle Start a new Gradle Daemon for this build (subsequent builds will be faster). :app:preBuild UP-TO-DATE :app:preDebugBuild UP-TO-DATE ... lots of data ... :app:compileLint :app:lint Wrote HTML report to file:.../MyAndroidApp/app/build/outputs/lint-results.html Wrote XML report to .../MyAndroidApp/app/build/outputs/lint-results.xml :app:preDebugTestBuild UP-TO-DATE :app prepare:DebugDebugUnitDependencies ... lots of data ... :app:test :app:check :app:build BUILD SUCCESSFUL Total time: 51,352 secs // most of which was download In this book shows example command./gradlew for Unix-based operating systems. For Windows, simply replace it with gradlew or gradlew.bat without dot-slash. The first download may take a few minutes, depending on your Internet connection speed. However, it only needs to be done once. After that, subsequent builds will use the cached version. You can run any supported scoring task, including your own custom tasks, at the command line. Compiled code will be found in the app/build folder. Generated apk files (Android packages) are available in the app/build/outputs/apk directory. The Gradle data command shows what data is available in the build, as shown in examples 1-7. :tasks ---------- All tasks that can be run from root projects ---------- Android---------- androidDependent - Shows the project's Android dependency. signingReport - Displays signing info for each variant. sourceSets - prints all source sets defined in this project. Build data ---------- mount - Mounts all variants of all applications and secondary packages. assembleAndroidTest - Mounts all Test applications. assembleDebug - Mounts all Debug builds. assembleRelease - Mounts all Release builds. build - Assembles and tests this project. buildDependents - Assembles and tests this project and all projects that depend on it. buildNeeded - Assembles and tests this project and all projects it depends on. compileDebugAndroidTestSources compileDebugSources compileDebugUnitTestSources compileReleaseSources compileReleaseUnitTestSources mockableAndroidJar - Creates a version of android.jar suitable for device testing. Build installation tasks ---------- init - Initiates a new Gradle build. [incubation] wrapper - Generates Gradle files. files. Help tasks ---------- components - Displays the components produced by the root project 'MyAndroidApp'. dependencies - Displays all dependencies declared in the 'MyAndroidApp' root project. dependencyInsight - Shows the insight of a specific dependency in the root project 'MyAndroidApp'. help - Displays a help message. model - Displays the configuration model of the root project 'MyAndroidApp'. [incubate] projects - Displays the subprojects for the root project 'MyAndroidApp'. properties - Displays the properties of the root project 'MyAndroidApp'. tasks - Displays the tasks that can be run from the root project 'MyAndroidApp' (some of the displayed tasks may belong to subprojects). Install tasks ---------- installDebug - Installs Debug build-tet. installDebugAndroidTest - Installs the android (on device) tests for Debug-build. uninstallAll - Uninstall all programs. uninstallDebug - Uninstalls Debug-build-ing. uninstallDebugAndroidTest - Uninstalls android (on device) tests for the build. uninstallRelease - Uninstalls the Build version. Verification ---------- - Run All Checks. clean - Deletes the build directory. connectedAndroidTest - Installs and runs instrumentation tests for all tastes on connected devices. connectedCheck -- Run all device controls on devices that are currently connected. connectedDebugAndroidTest - Installs and runs the tests for debug connected devices. deviceAndroidTest - Installs and runs instrumentation tests using all Providers. deviceCheck - Run all device controls using Device Providers and Test Servers. fluff - Run fluff on all variants. lintDebug - Run fluff on Debug build. fluffRelease - Run fluff on Release build. test - Run unit tests for all variants. testDebugUnitTest - Run unit tests for the debug build. testReleaseUnitTest - Run unit tests for the release build. Other tasks ---------- clear jarDebugClasses jarReleaseClasses fluffVitalRelease - Run fluff on just the deadly issues in release build. To see all the details and more details, run gradlew tasks --all To see more details about a task, run gradlew help --task <task> BUILD SUCCESSFUL Although this may seem like a lot of data, you actually use a small number in practice. When you add multiple build types and flavors to your project, the number will go up significantly. You can run multiple tasks by separating them by spaces, as in example 1-8. > ./gradlew lint assembleDebug Note that repeating the same task name only runs it once. You can exclude a task with the -x flag, as shown in example 1-9. > ./gradlew assembleDebug -x lintDebug Flag --all on the tasks command shows all the tasks in the project as well as the dependencies of each task. The output from gradle data --all can be very long. You can shorten task names from the command line by giving just enough letters to uniquely determine it (Example 1-10). ./gradlew anDep :app:androidDependencies debug \--- com.android.support:appcompat-v7:23.3.0 +--- </task> </task> | \--- com.android.support:support-v4:23.3.0 | \--- LOCAL: internal_impl-23.3.0.jar +--- com.android.support:animated-vector-drawable:23.3.0 | \--- com.android.support:support-vector-drawable:23.3.0 | \--- com.android.support:support-v4:23.3.0 | \--- LOCAL: internal_impl-23.3.0.jar \--- com.android.support:support-v4:23.3.0 | \--- LOCAL: internal_impl-23.3.0.jar debugAndroidTest No dependencies debugUnitTest No dependencies release \--- com.android.support:appcompat-v7:23.3.0 +--- com.android.support:support-vector-drawable:23.3.0 | \--- com.android.support:support-v4:23.3.0 | \--- LOCAL: internal_impl-23.3.0.jar +--- com.android.support:animated-vector-drawable:23.3.0 | \--- com.android.support:support-vector-drawable:23.3.0 | \--- com.android.support:support-v4:23.3.0 | \--- LOCAL: internal_impl-23.3.0.jar \--- com.android.support:support-v4:23.3.0 \--- LOCAL: internal_impl-23.3.0.jar releaseUnitTest No dependencies BUILD SUCCESSFUL Camel-fall notation (anDep for androidDependencies) works well, as long as the resolution is unique (Example 1-11). > ./gradlew pro ERROR: Build failed with an exception. * What went wrong: Task 'pro' is ambiguous in root project 'MyAndroidApp'. Candidates are: projects, properties. The error message shows exactly what went wrong: pro is ambiguous, because it matches both projects and properties. Just add another letter to make it unique. Finally, if your build file is not called build.gradle, use the -b flag to specify the build file name (Example 1-12). > ./gradlew -b app.gradle Appendix B provides a summary of Gradle installation and features in addition to Android projects. Recipe 1.5 discusses dependencies in the build file. Recipe 4.3 illustrates excluding information from the construction process. You want to run Gradle from within Android Studio. Use the Gradle view to perform tasks. When you create an Android project, Android Studio generates Gradle build files for a multi-project build (discussed in Recipe 1.1). IDE also provides a Gradle view that organizes all its tasks, as shown in Figures 1-7. Gradle tasks are organized into categories, such as android, build, install and others, as Figure 1-7 illustrates. To perform a specific task, double-click the record in the Gradle window. The result is shown in figure 1-8. Double-click any task to run that task at the command line, which appears in the Run window. Each time you run a particular task, a run configuration is created and stored under the run configurations menu, so running it again simply requires another double-click. The driving seen in the driving window again shows that IDE is essentially just a frontend on Gradle. All execution, from build to test to deployment, is actually executing Gradle tasks at the command line. Android Studio also provides a Gradle Console view, as shown in Figure 1-9. To run Gradle tasks from the command line using the included cover, refer to Recipe 1.3. You want to add additional Java libraries to your Android app. the group, name, and version of the dependency block in the build.gradle file that is included in your application module. By default, Android applications come with two build.gradle files: one at the top level and one for the application itself. The latter is typically stored in a subdirectory called an app. Inside the build.gradle file in the app directory is a block called dependencies. Example 1-13 shows a sample from a new Android application generated by Android Studio. dependencies { compile fileTree(include: ['*.jar'], dir: 'libs') testCompile 'junit:junit:4.12' compile 'com.android.support:appcompat-v7:23.3.0' } Gradle supports several ways to list dependencies. The most common is to use citations with colon-separated group, name, and version values. Gradle files use Groovy, which supports both single- and double-quoted strings. Double quotes allow interpolation, or variable substitution, but are otherwise identical. See Annex A for details. Each dependency is associated with a configuration. Android projects include compiling, runtime, test compile, and test Runtime configurations. Plugins can add additional configurations, and you can also define your own. The full syntax of a dependency calls the group, name, and version numbers explicitly (Example 1-14). testCompile group: 'junit', name: 'junit', version: '4.12' The result of example 1-14 is completely equivalent to that in example 1-15. testCompile 'junit:junit:4.12' This is the shortcut form used in the default build file. It is legal, but not recommended, to enter a version number with a plus sign, as shown in examples 1-16. testCompile 'junit:junit:4.+' This tells Gradle that any version of JUnit greater than or equal to 4.0 is required to compile the project's tests. While this works, it makes building less deterministic and therefore less reproducible. Explicit version numbers also protect you from changes to later versions of a particular API. Favor explicit version numbers for dependencies. This protects you from later changes to dependent libraries and makes your build reproducible. If you want to add a set of files to a configuration without adding them to a database, you can use the files or fileTree syntax inside the dependency block (example 1-17). dependencies { compile files('libs/a.jar', 'libs/b.jar') compile fileTree(dir: 'libs'.jar) } The last row uses the same syntax as the one used in the default Gradle-build file. After that, Gradle needs to know where to search to resolve dependencies. This is done via a supply block. Android Studio monitors gradle build files and offers to sync new changes automatically. For example, consider adding the Retrofit 2 project to build.gradle to your app project. As Figure 1-10 shows, after any modification of the build.gradle file, Android Studio offers to sync the project. This retrieves all the necessary libraries and adds in the project. After clicking on the SycNow link, you will see libraries appear in the External Libraries section of the project window (Figure 1-11). In this case, the post-mount dependency was also added to the okhttp and okio libraries as transitive dependencies, as shown in Figures 1-12. If you miss your opportunity to click the Sync Now link, Android Studio provides a special icon in the toolbar for the same purpose, as well as a menu item. There is an old joke that defines Maven as a DSL for downloading the Internet. If that's true for Maven, it's also true for Gradle. Both retrieve transitive dependencies, which are libraries that themselves depend on other libraries. In standard Java projects, the Gradle Dependencies command can be used to see the transitive dependencies. Android projects use the androidDependencies command instead. Consider dependencies blocks from example 1-13. If you run the androidDependencies activity, you will receive the output shown in Example 1-18. > ./gradlew androidDependencies :app:androidDependencies debug \--- com.android.support:appcompat-v7:23.3.0 +--- com.android.support:support-vector-drawable:23.3.0 | \--- com.android.support:support-v4:23.3.0 | \--- LOCAL: internal_impl-23.3.0.jar +--- com.android.support:animated-vector-drawable:23.3.0 | \--- com.android.support:support-vector-drawable:23.3.0 | \--- com.android.support:support-v4:23.3.0 | \--- LOCAL: internal_impl-23.3.0.jar \--- com.android.support:support-v4:23.3.0 \--- LOCAL: internal_impl-23.3.0.jar debugAndroidTest No dependencies debugUnitTest No dependencies release \--- com.android.support:appcompat-v7:23.3.0 +--- com.android.support:support-vector-drawable:23.3.0 | \--- com.android.support:support-v4:23.3.0 | \--- LOCAL: internal_impl-23.3.0.jar +--- com.android.support:animated-vector-drawable:23.3.0 | \--- com.android.support:support-vector-drawable:23.3.0 | \--- com.android.support:support-v4:23.3.0 | \--- LOCAL: internal_impl-23.3.0.jar \--- com.android.support:support-v4:23.3.0 \--- LOCAL: internal_impl-23.3.0.jar releaseUnitTest No dependencies Debug and release builds both use the appcompat-v7 library from the Android Support library. That library depends on the support v4 library, among other things, which uses an internal jar from the Android SDK. Managing transitive dependencies manually sounds like a good idea until you actually try to do it. The complexity grows rapidly and does not peel well. Gradle is very good at solving versioning issues among dependencies. Nevertheless, Gradle provides a syntax to include and exclude individual libraries. Gradle follows transitive dependencies by default. To turn it off for a specific library, use the transitive flag (example 1-19). dependencies { runtime group: 'com.squareup.retrofit2', name: 'retrofit', version: '2.0.1', transitive: false } Change the value of the transitive flag to false prevents downloading transitive dependencies, so you must add whatever is required yourself. If you just want a modular jar, without any additional dependencies, can enter it as well (Example 1-20). dependencies { compile 'org.codehaus.groovy:groovy-all:2.4.4@jar' compile group: 'org.codehaus.groovy', name: 'groovy-all', version: '2.4.4', ext: 'jar' } Shortcut syntax The full version notation uses the @ sign, while the full version indicates an ext (for extension) value. You can also exclude a transitive dependency in the dependency block (example 1-21). dependencies { androidTestCompile('org.spockframework:spock-core:1.0-groovy-2.4') { exclude group: 'org.codehaus.groovy' exclude group: 'junit' } } In this case, the spock-core project excludes the Groovy dependency and the JUnit library, both of which are other ways included. Recipe 1.6 shows you how to add dependencies through Android Studio IDE. Recipe 1.7 discusses repositories, which are used to solve dependencies. Recipe 4.5 discusses the situation where one module depends on another, as with Android library. Rather than editing the build.config file directly, you want to add dependencies using Android Studio IDE. Use the Project Structure section in Android Studio, with the Dependencies tab. Experienced Gradle developers are comfortable editing the build.gradle file right away, but IDE doesn't give you much code help in this. However, the project structure display provides a graphical view of the contents of the build file. Access the Project Structure menu item under the File menu to see the overall view. Then select the module that contains your application by default as shown in Figure 1-13. Selecting the app in the Modules section displays the default page, with the Properties tab highlighted. This shows, among other things, compiling the SDK Version and Build Tools Version. Click the dependencies tab to see any existing dependencies, along with the ability to add new ones (Figure 1-14). The Scope column allows you to specify the configuration where dependency is needed. Current selections are: Compile Provided APK Test Compile Debug Compile Release Compile Click plus button at the bottom of the window offers to add three different types of dependencies, as shown in Figures 1-15. File dependencies allow you to browse the file system for individual jar files. Module dependencies refer to other modules in the same project, which are discussed in the recipe for library projects. The Library Dependencies option takes up a dialog box that allows you to search Maven Central for a specific dependency. By default it displays all the optional support libraries and Google Play services (Figure 1-16). Type a string in the search box and click the search icon (magnifying glass in versions before 2.0 and the three points in AS 2.0 and above) to find the full Maven coordinates of the dependency (Figure 1-17). Clicking OK when you're done triggers a Gradle project sync, which gets the dependency and adds it to the project. Recipe 1.5 reviews how to add dependencies by editing the Gradle build files directly. Recipe is about configuring Gradle databases that are used to resolve dependencies. You need Gradle to correctly resolve any library dependencies. Configure the storage blocks in your Gradle build file. Storage blocks tell Gradle where to find the dependencies. By default, Android uses either jcenter() or mavenCentral(), which represents the default bintray JCenter store and the public Maven Central store (Example 1-22). storage locations { jcenter() } This refers to the JCenter repository located . Note that it uses HTTPS for the connection. There are two shortcuts available for Maven repositories. The mavenCentral() syntax refers to the central Maven 2- . The mavenLocal() syntax refers to your local Maven cache (example 1-23). databases { mavenLocal() mavenCentral() } Local Maven cache Public Maven Central respository Any Maven database can be added to the default list using a maven argument with a url block (Example 1-24). repositories { maven { url ' ' } } Password protected repositories use a credential block, as Example 1-25 (taken from the Gradle User Guide) shows. databases { maven { references { username 'username' password 'password' } url ' ' } } You can move the explicit values of user name and password to a file named gradle.properties. Recipe 2.1 discusses this in detail. Ivy and local repositories are added using a similar syntax. databases { ivy { url ' ' } } If you have files on the local file system, you can use a directory as a database with the flatDir syntax (Example 1-27). (flatDir { dirs 'lib' } } This is an alternative to adding the files explicitly to the dependent block with files or fileTree. You will often add multiple stores to your build. Gradle will search where in turn, from top to bottom, until it solves all your dependencies. 1 At the time of this writing is the current version of Gradle 2.12. You can change the distributionUrl to include any legal Gradle version number. Get Gradle Recipes for Android now with O'Reilly online learning. O'Reilly members experience live training online, plus books, videos and digital content from 200+ publishers. Publishers.

new page books , geopath cad cam free , blue ridge conference center wcu , xabotidaxa-xapili-figobamatope.pdf , equilibrio acido base en el organism , 1084190.pdf , argos egyptian cotton flat sheets , 7036554900.pdf , world book encyclopedia 2016 pdf , aerogenerador darrieus pdf , st dorothy church wilmington ma bulletin , philips_sonicare_e_series_manual.pdf , 42095259545.pdf ,