Android Gradle plugin release notes

The JCenter repository became read-only on March 31st, 2021. For more information, see JCenter service update (/studio/build/jcenter-migration).

Important: For a detailed log of Android Gradle plugin API deprecations and removals, see the Android Gradle plugin API updates (/studio/releases/gradle-plugin-api-updates).

The Android Studio build system is based on Gradle, and the Android Gradle plugin adds several features that are specific to building Android apps. Although the Android plugin is typically updated in lock-step with Android Studio, the plugin (and the rest of the Gradle system) can run independent of Android Studio and be updated separately.

This page explains how to keep your Gradle tools up to date and what's in the recent updates.

For a high-level summary of upcoming breaking changes in the Android Gradle plugin, see the Android Gradle plugin roadmap (/studio/releases/gradle-plugin-roadmap).

For details about how to configure your Android builds with Gradle, see the following pages:

- Configure Your Build (/studio/build)
- Android Plugin DSL Reference (/reference/tools/gradle-api)
- Gradle DSL Reference (https://docs.gradle.org/current/dsl/)

For more information about the Gradle build system, see the Gradle user guide (https://docs.gradle.org/current/userguide/userguide.html).

Update the Android Gradle plugin

When you update Android Studio, you may receive a prompt to automatically update the Android Gradle plugin to the latest available version. You can choose to accept the update or manually specify a version based on your project's build requirements.

You can specify the plugin version in either the **File** > **Project Structure** > **Project** menu in Android Studio, or the top-level `build.gradle` file. The plugin version applies to all modules built in that Android Studio project. The following example sets the plugin to version 7.0.0 from the `build.gradle` file:

Groovy (#groovy)Kotlin (#kotlin)

```
plugins {
    id("com.android.application") version "7.0.0-alpha13" apply false
    id("com.android.library") version "7.0.0-alpha13" apply false
    id("org.jetbrains.kotlin.android") version "1.5.31" apply false
}
```

Caution: You should not use dynamic dependencies in version numbers, such as `'com.android.tools.build:gradle:2.+'`. Using this feature can cause unexpected version updates and difficulty resolving version differences.

If the specified plugin version has not been downloaded, Gradle downloads it the next time you build your project or click **File** > **Sync Project with Gradle Files** from the Android Studio menu bar.

Update Gradle

When you update Android Studio, you may receive a prompt to also update Gradle to the latest available version. You can choose to accept the update or manually specify a version based on your project's build requirements.

The following table lists which version of Gradle is required for each version of the Android Gradle plugin. For the best performance, you should use the latest possible version of both Gradle and the plugin.

Plugin version	Required Gradle version
1.0.0 - 1.1.3	2.2.1 - 2.3

1.2.0 - 1.3.1	2.2.1 - 2.9
1.5.0	2.2.1 - 2.13
2.0.0 - 2.1.2	2.10 - 2.13
2.1.3 - 2.2.3	2.14.1 - 3.5
2.3.0+	3.3+
3.0.0+	4.1+
3.1.0+	4.4+
3.2.0 - 3.2.1	4.6+
3.3.0 - 3.3.3	4.10.1+
3.4.0 - 3.4.3	5.1.1+
3.5.0 - 3.5.4	5.4.1+
3.6.0 - 3.6.4	5.6.4+
4.0.0+	6.1.1+
4.1.0+	6.5+
4.2.0+	6.7.1+
7.0	7.0+
7.1	7.2+
7.2	7.3+

You can specify the Gradle version in either the **File** > **Project Structure** > **Project** menu in Android Studio, or by editing the Gradle distribution reference in the `gradle/wrapper/gradle-wrapper.properties` file. The following example sets the Gradle version to 7.0.0 in the `gradle-wrapper.properties` file.

```
...
distributionUrl = "https\://services.gradle.org/distributions/gradle-7.0.0-all.zi
...
```

Versioning changes (November 2020)

We are updating the version numbering for Android Gradle plugin (AGP) to more closely match the underlying Gradle build tool.

Here are the notable changes:

- AGP will now use semantic versioning, and breaking changes will be targeted for major releases.
- There will be one major version of AGP released per year, aligned with the Gradle major release.
- The release after AGP 4.2 will be version 7.0 and will require an upgrade to Gradle version 7.x. Every major release of AGP will require a major version upgrade in the underlying Gradle tool.
- APIs will be deprecated approximately one year in advance, with replacement functionality made available concurrently. Deprecated APIs will be removed approximately one year later during the subsequent major update.

7.1.0 (January 2022)

Android Gradle plugin 7.1.0 is a major release that includes a variety of new features and improvements.

7.1.2 (February 2022)

This minor update includes the following bug fixes:

- Android Gradle Plugin 7.1.0-rc01 fails to perform ASM bytecode transformation during unit tests
- Gradle sync fails with "Unable to load class 'com.android.build.api.extension.AndroidComponentsExtension'."
- Some new DSL blocks can't be used from Groovy DSL in Android Gradle Plugin 7.0.0
- AGP 7.1 new publishing API: created javadoc jar does not get signed
- ClassesDataSourceCache should use latest Asm version
- Android Studio BumbleBee does not always deploy latest changes

To see a complete list of bug fixes included in this release, see the Android Studio Bumblebee Patch 2 blog post (https://androidstudio.googleblog.com/2022/02/android-studio-bumblebee-202111-patch-2.html)
.

7.1.1 (February 2022)

This minor update corresponds to the release of Android Studio Bumblebee Patch 1.

To see a list of bug fixes included in this release, see the Android Studio Bumblebee Patch 1 blog post (https://androidstudio.googleblog.com/2022/02/android-studio-bumblebee-202111-patch-1.html)
.

Compatibility

	Minimum version	Default version	Notes
Gradle	7.2	7.2	To learn more, see updating Gradle (/studio/releases/gradle-plugin?buildsystem=ndk-build#updating-gradle) .

	Minimum version	Default version	Notes
SDK Build Tools	30.0.3	30.0.3	Install (/studio/intro/update#sdk-manager) or configure (/studio/releases/build-tools) SDK Build Tools.
NDK	N/A	21.4.7075529	Install (/studio/projects/install-ndk#specific-version) or configure (/studio/projects/install-ndk#apply-specific-version) a different version of the NDK.
JDK	11	11	To learn more, see setting the JDK version (/studio/intro/studio-config#jdk).

Lint analysis task is now cacheable

The `AndroidLintAnalysisTask` is now compatible with the Gradle build cache (https://docs.gradle.org/current/userguide/build_cache.html). If you enable the build cache by setting `org.gradle.caching=true` in your `gradle.properties` file, the lint analysis task will get its output from the build cache when possible.

The lint analysis task is often the biggest bottleneck when running lint with the Android Gradle plugin, so enabling the build cache improves build speed when running lint in many situations. You should see a noticeable performance improvement, for example, if you have a multi-module project and clean your build directory before running lint on your CI server.

C/C++ modules may now reference other C/C++ modules in the same project

A Gradle Android module with C/C++ code may now be set up to reference header files and library code in another Gradle module. The Prefab (https://google.github.io/prefab/) protocol is used to communicate the headers and libraries between Gradle modules.

Requirements

- The *consuming* module must be `CMake` and not `ndk-build`. Support for ndk- build will require a future NDK update. The *publishing* module may be `CMake` or `ndk-build`.
- The *consuming* module must enable `prefab` in the `build.gradle` file.

```
android {
  buildFeatures {
    prefab true
  }
}
```

- The *publishing* module must enable `prefabPublishing` in the `build.gradle` file.

```
android {
  buildFeatures {
    prefabPublishing true
  }
}
```

- The *consuming* module must reference the *publishing* module by adding a line in the `build.gradle` file `dependencies` block. For example:

```
dependencies {
  implementation project(':mylibrary')
}
```

- The *publishing* module must expose a package using a `prefab` section. For example:

```
android {
  prefab {
    mylibrary {
      libraryName "libmylibrary"
      headers "src/main/cpp/include"
    }
  }
}
```

- The consuming module's `CMakeLists.txt` file may use `find_package()` to locate the package published by the producing module. For example:

```
find_package(mylibrary REQUIRED CONFIG)
target_link_libraries(
        myapplication
        mylibrary::mylibrary)
```

- There must be one STL for the entire application (/ndk/guides/cpp-support#one_stl_per_app). So, for example, both consuming and publishing modules can use C++ shared STL.

```
   android {
defaultConfig {
        externalNativeBuild {
          cmake {
            arguments '-DANDROID_STL=c++_shared'
          }
        }
      }
    }
```

For further explanation of how to configure native AAR consumers and producers with AGP, see Native dependencies with AGP (/studio/build/dependencies?&agpversion=4.1&buildsystem=ndk-build#native-dependencies-with-agp).

Repository settings in `settings.gradle` file

When a new project is created in Android Studio Bumblebee, the top-level `build.gradle` file contains the `plugins` block, followed by code to clean your build directory:

```
plugins {
    id 'com.android.application' version '7.1.0-beta02' apply false
    id 'com.android.library' version '7.1.0-beta02' apply false
    id 'org.jetbrains.kotlin.android' version '1.5.30' apply false
}
```

```
task clean(type: Delete) {
  delete rootProject.buildDir
}
```

The repository settings that were previously in the top-level `build.gradle` file are now in the `settings.gradle` file:

```
pluginManagement {
  repositories {
    gradlePluginPortal()
    google()
    mavenCentral()
  }
}

dependencyResolutionManagement {
  repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
  repositories {
    google()
    mavenCentral()
  }
}
rootProject.name = 'GradleManagedDeviceTestingNew'
include ':app'
```

The module-level `build.gradle` file has not changed. So, use the top-level `build.gradle` file and the `settings.gradle` file to define build configurations that apply to all modules in your project, or the repositories and dependencies that apply to Gradle itself; use the module-level `build.gradle` file to define build configurations that are specific to a given module within your project.

Improved resource shrinker

Android Studio Bumblebee includes an improved resource shrinker that helps reduce your app size.

Support for apps with dynamic features

The default implementation of the Android resource shrinker has been updated in Android Gradle Plugin 7.1.0-alpha09. The new implementation supports shrinking apps with dynamic features.

Experimental further app size reductions

The new resource shrinker implementation can reduce the size of your shrunk app even more by modifying the resource table to remove unused value resources and references to unused file resources. The new resource shinker can delete unused file resources completely, reducing the size of your app more. This behavior is not enabled by default yet, but you can opt in to try it by adding the experimental option `android.experimental.enableNewResourceShrinker.preciseShrinking=true` to your project's `gradle.properties` file.

Please report any issues you find with the new resource shrinker or the experimental flag. To help diagnose issues, or as a temporary workaround, you can switch back to the previous implementation by adding `android.enableNewResourceShrinker=false` to your project's `gradle.properties`. The new shrinker replaces unused file-based resources with slightly different minimal files than the previous resource shrinker, but this is not expected to have any runtime impact.

The old implementation is scheduled to be removed in Android Gradle plugin 8.0.0.

Build variant publishing

Android Gradle plugin 7.1.0 and higher allows you to configure which build variants to publish to an Apache Maven repository. AGP creates a component with a single or multiple build variants based on the new publishing DSL, which you can use to customize a publication to a Maven repository. Compared to previous versions, this also avoids unnecessary work, as no components will be created by default. To learn more, see the publishing code sample (https://android.googlesource.com/platform/tools/base/+/refs/heads/mirror-goog-studio-main/build-system/gradle-api/src/main/java/com/android/build/api/dsl/Publishing.kt)
.

Publish Javadoc JAR

AGP 7.1.0 and higher allows you to generate Javadoc from Java and Kotlin sources and publish Javadoc JAR files in addition to AARs for library projects. The Javadoc is added to the POM and Gradle Module Metadata (https://docs.gradle.org/current/userguide/publishing_gradle_module_metadata.html) files. Enable

this feature by adding `withJavadocJar()` in the `singleVariant` or `multipleVariants` publishing block. To learn more, see the publishing options code sample (https://android.googlesource.com/platform/tools/base/+/refs/heads/mirror-goog-studio-main/build-system/gradle-api/src/main/java/com/android/build/api/dsl/PublishingOptions.kt#50)
.

Publish sources JAR

AGP 7.1.0 and higher allows you to publish Java and Kotlin source JAR files in addition to AARs for library projects. The sources are added to the POM and Gradle Module Metadata (https://docs.gradle.org/current/userguide/publishing_gradle_module_metadata.html) files. You can enable this feature by adding `withSourcesJar()` in the `singleVariant` or `multipleVariants` publishing block. To learn more, see the publishing options code sample (https://android.googlesource.com/platform/tools/base/+/refs/heads/mirror-goog-studio-main/build-system/gradle-api/src/main/java/com/android/build/api/dsl/PublishingOptions.kt#45)
.

Lint block semantic change

All lint methods that override the given severity level of an issue—`enable`, `disable`/`ignore`, `informational`, `warning`, `error`, `fatal`—now respect the order of configuration. For example, setting an issue as fatal in `finalizeDsl()` (/reference/tools/gradle-api/7.1/com/android/build/api/extension/AndroidComponentsExtension#finalizedsl) now overrides disabling it in the main DSL. For more information, see the `lint{}` (/reference/tools/gradle-api/7.1/com/android/build/api/dsl/Lint) block reference docs and Android build flow and extension points (/studio/build/extend-agp#build-flow-extension-points).

Navigation Safe Args compatibility

AGP APIs that the Navigation Safe Args Gradle plugin (/guide/navigation/navigation-getting-started#ensure_type-safety_by_using_safe_args) depend on have been removed. AGP 7.1 does not work with Navigation Safe Args versions 2.4.0-rc1 or 2.4.0, but will work with versions 2.5.0-alpha01 and 2.4.1. In the meantime, as a workaround, you can use AGP 7.1 with a snapshot build of Navigation Safe Args, Navigation 2.5.0-SNAPSHOT. To use the snapshot build, follow the snapshot instructions (https://androidx.dev/) with build id #8054565.

In addition, Navigation Safe Args versions 2.4.1 and 2.5.0 will no longer work with AGP 4.2; to use those versions of Safe Args, you must use AGP 7.0 and higher.

Disable automatic component creation

Starting AGP 8.0, automatic component creation will be disabled by default. Currently, AGP 7.1 automatically creates a component for each build variant, which has the same name as the build variant, and an an `all` component that contains all the build variants. This automatic component creation will be disabled. To transition to the new behavior, you should manually disable automatic component creation by setting `android.disableAutomaticComponentCreation`to`true.` For more information, see Use the Maven Publish plugin (/studio/build/maven-publish-plugin).

Firebase Performance Monitoring compatibility

AGP 7.1 is incompatible with the Firebase Performance Monitoring Gradle plugin version 1.4.0 and lower. The AGP Upgrade Assistant will not automatically update the plugin to version 1.4.1, so if you are using `firebase-perf` and wish to upgrade AGP to 7.1, you need to do this particular upgrade manually.

Known issues

This section describes known issues that exist in Android Gradle plugin 7.1.0.

Problems with unit testing an app project that uses the Hilt plugin

The unit test classpath contains the non-instrumented app classes, which means Hilt does not instrument the app classes to handle dependency injection when running unit tests.

This issue will be fixed with 7.1.1 release, see issue #213534628 (https://issuetracker.google.com/213534628).

7.0.0 (July 2021)

Android Gradle plugin 7.0.0 is a major release that includes a variety of new features and improvements.

7.0.1 (August 2021)

This minor update includes various bug fixes. To see a list of notable bug fixes, read the related post on the Release Updates blog (https://androidstudio.googleblog.com/2021/08/android-studio-arctic-fox-202031-patch.html).

Compatibility

	Minimum version	Default version	Notes
Gradle	7.0.2	7.0.2	To learn more, see updating Gradle (/studio/releases/gradle-plugin?buildsystem=ndk-build#updating-gradle) .
SDK Build Tools	30.0.2	30.0.2	Install (/studio/intro/update#sdk-manager) or configure (/studio/releases/build-tools) SDK Build Tools.
NDK	N/A	21.4.7075529	Install (/studio/projects/install-ndk#specific-version) or configure (/studio/projects/install-ndk#apply-specific-version) a different version of the NDK.
JDK	11	11	To learn more, see setting the JDK version (/studio/intro/studio-config#jdk).

JDK 11 required to run AGP 7.0

When using Android Gradle plugin 7.0 to build your app, JDK 11 is now required to run Gradle. Android Studio Arctic Fox bundles JDK 11 and configures Gradle to use it by default, which means that most Android Studio users do not need to make any configuration changes to their projects.

If you need to manually set the JDK version (/studio/intro/studio-config#jdk) used by AGP inside of Android Studio, you need to use JDK 11 or higher.

When using AGP independent of Android Studio, upgrade the JDK version by setting the JAVA_HOME environment variable (https://docs.gradle.org/current/userguide/build_environment.html#sec:gradle_environment_variables) or the `-Dorg.gradle.java.home` command-line option (https://docs.gradle.org/current/userguide/command_line_interface.html#environment_options) to your installation directory of JDK 11.

Variant API stable

The new Variant API is now stable. See the new interfaces in the com.android.build.api.variant (/reference/tools/gradle-api/7.0/com/android/build/api/variant/Variant) package, and examples in the gradle-recipes (https://github.com/android/gradle-recipes/tree/agp-7.0) GitHub project. As part of the new Variant API, we have made available a number of intermediate files, called artifacts, through the Artifacts (/reference/tools/gradle-api/7.0/com/android/build/api/artifact/Artifacts) interface. These artifacts, like the merged manifest, can be safely obtained and customized by using third-party plugins and code.

We will continue extending the Variant API by adding new functionalities and augmenting the number of intermediate artifacts we make available for customization.

Behavior changes for Lint

This section describes multiple Lint behavior changes in Android Gradle plugin 7.0.0.

Improved lint for library dependencies

Running lint with `checkDependencies = true` is now faster than before. For Android projects consisting of an app with library dependencies, it is recommended to set `checkDependencies` to `true` as shown below, and to run lint via `./gradlew :app:lint`, which will analyze all dependency modules in parallel and produce a single report including issues from the app and all of its dependencies.

Groovy (#groovy)Kotlin (#kotlin)

```
// build.gradle.kts

android {
  ...
  lint {
    isCheckDependencies = true
  }
}
```

Lint tasks can now be UP-TO-DATE

If a module's sources and resources have not changed, the lint analysis task for the module does not need to run again. When this happens, the execution of the task appears as "UP-TO-DATE" in the Gradle output. With this change, when running lint on an application module with `checkDependencies = true`, only modules that have changed will need to run their analysis. As a result, Lint can run even faster.

The Lint report task also does not need to run if its inputs have not changed. A related **known issue** is that there is no lint text output printed to stdout when the lint task is UP-TO-DATE (issue #191897708 (https://issuetracker.google.com/191897708)).

Running lint on dynamic-feature modules

AGP no longer supports running lint from dynamic-feature modules. Running lint from the corresponding application module will run lint on its dynamic-feature modules and include all issues in the app's lint report. A related **known issue** is that when running lint with `checkDependencies = true` from an app module, dynamic-feature library dependencies aren't checked unless they're also app dependencies (issue #191977888 (https://issuetracker.google.com/191977888)).

Running lint on default variant only

Running `./gradlew :app:lint` now runs lint for only the default variant. In previous versions of AGP, it would run lint for all variants.

Missing class warnings in R8 shrinker

R8 (/studio/build/shrink-code) more precisely and consistently handles missing classes and the `-dontwarn` option. Therefore, you should start to evaluate the missing class warnings emitted by R8.

When R8 encounters a class reference that is not defined in your app or one of its dependencies, it will emit a warning that appears in your build output. For example:

```
R8: Missing class: java.lang.instrument.ClassFileTransformer
```

This warning means that the class definition `java.lang.instrument.ClassFileTransformer` could not be found when analyzing your app's code. While this usually means there is an error, it's possible that you may want to ignore this warning. Two common reasons to ignore the warning are:

1. Libraries that are targeting the JVM and the missing class are of JVM library type (as in the example above).
2. One of your dependencies uses a compile-time only API.

You can ignore a missing class warning by adding a `-dontwarn` rule to your `proguard-rules.pro` file. For example:

```
-dontwarn java.lang.instrument.ClassFileTransformer
```

For convenience, AGP will generate a file that contains all potentially missing rules, writing them to a file path such as the following: `app/build/outputs/mapping/release/missing_rules.txt`. Add the rules to your `proguard-rules.pro` file to ignore warnings.

In AGP 7.0, missing class messages will appear as warnings, and you can turn them into errors by setting `android.r8.failOnMissingClasses = true` in `gradle.properties`. In AGP 8.0, these warnings will become errors that break your build. It is possible to keep the AGP 7.0 behavior by adding the option `-ignorewarnings` to your `proguard-rules.pro` file, but that is not recommended.

Android Gradle plugin build cache removed

The AGP build cache was removed in AGP 4.1. Previously introduced in AGP 2.3 to complement the Gradle build cache, the AGP build cache was superseded entirely by the Gradle build cache in AGP 4.1. This change does not impact build time.

In AGP 7.0, the `android.enableBuildCache` property, `android.buildCacheDir` property, and the `cleanBuildCache` task have been removed.

Use Java 11 source code in your project

You can now compile up to Java 11 source code in your app's project, enabling you to use newer language features like private interface methods, the diamond operator for anonymous classes, and local variable syntax for lambda parameters.

To enable this feature, set `compileOptions` to the desired Java version and set `compileSdkVersion` to 30 or above:

Groovy (#groovy)Kotlin (#kotlin)

```
// build.gradle.kts

android {
    compileSdkVersion(30)

    compileOptions {
      sourceCompatibility(JavaVersion.VERSION_11)
      targetCompatibility(JavaVersion.VERSION_11)
    }

    kotlinOptions {
      jvmTarget = "11"
    }
}
```

Dependency configurations removed

In AGP 7.0, the following configurations (or dependency scopes) have been removed:

- `compile`
  Depending on use case, this has been replaced by `api` (/studio/build/dependencies#api) or `implementation` (/studio/build/dependencies#implementation).
  Also applies to **Compile* variants, for example: `debugCompile`.
- `provided`
  This has been replaced by `compileOnly` (/studio/build/dependencies#compileOnly).
  Also applies to **Provided* variants, for example: `releaseProvided`.
- `apk`
  This has been replaced by `runtimeOnly` (/studio/build/dependencies#runtimeOnly).
- `publish`
  This has been replaced by `runtimeOnly` (/studio/build/dependencies#runtimeOnly).

In most cases, the AGP Upgrade Assistant (/studio/releases#agp-upgrade-assistant) will automatically migrate your project to the new configurations.

Classpath change when compiling against Android Gradle plugin

If you are compiling against the Android Gradle plugin, your compile classpath may change. Because AGP now uses `api/implementation` configurations internally, some artifacts may be removed from your compile classpath. If you depend on an AGP dependency at compile-time, be sure to add it as an explicit dependency.

Addition of native libraries in a Java resources folder is not supportec

Previously, you could add a native library in a Java resources folder, and register the folder using `android.sourceSets.main.resources.srcDirs`so that the native library would be extracted and added to the final APK. Starting with AGP 7.0, this is not supported and native libraries in a Java resources folder are ignored. Instead, use the DSL method intended for native libraries, `android.sourceSets.main.jniLibs.srcDirs`. For more information, see how to configure source sets (/studio/build/build-variants#configure-sourcesets).

Known issues

This section describes known issues that exist in Android Gradle plugin 7.0.0.

Incompatibility with 1.4.x Kotlin Multiplatform plugin

Android Gradle Plugin 7.0.0 is compatible with Kotlin Multiplatform plugin (/studio/releases/(https:/plugins.jetbrains.com/plugin/14936-kotlin-multiplatform-mobile/versions/dev)%7B:.external%7D)
1.5.0 and higher. Projects that use the Kotlin Multiplatform support need to update to Kotlin 1.5.0 to use Android Gradle Plugin 7.0.0. As a workaround, you can downgrade the Android Gradle plugin to 4.2.x, although this is not recommended.

For more information, see KT-43944 (https://youtrack.jetbrains.com/issue/KT-43944).

Missing lint output

There is no lint text output printed to stdout when the lint task is up-to-date (issue #191897708 (https://issuetracker.google.com/191897708)). For more context, see Behavior changes for lint (#lint-behavior-changes). This issue will be fixed in Android Gradle plugin 7.1.

Not all dynamic-feature library dependencies are lint checked

When running lint with `checkDependencies = true` from an app module, dynamic-feature library dependencies aren't checked unless they're also app dependencies (issue #191977888 (https://issuetracker.google.com/191977888)). As a workaround, the lint task can be run on those libraries. For more context, see Behavior changes for lint (#lint-behavior-changes).

4.2.0 (March 2021)

Compatibility

	Minimum version	Default version	Notes
Gradle	6.7.1 (https://docs.gradle.org/6.7.1/release-notes.html)	N/A	To learn more, updating Gradl (#updating-gradl
SDK Build Tools	30.0.2 (/studio/releases/build-tools#notes)	30.0.2 (/studio/releases/build-tools#notes)	Install (/studio/intro/up manager) or configure (/studio/releases tools) SDK Build Tool
NDK	N/A	21.4.7075529	Install (/studio/projects ndk#specific-vers or configure (/studio/projects ndk#apply-specif a different vers the NDK.

New features

This version of the Android Gradle plugin includes the following new features.

Java language version 8 by default

Starting in version 4.2, AGP will use the Java 8 language level by default. Java 8 provides access to a number of newer language features including lambda expressions, method references, and static interface methods. For the full list of supported features see the Java 8 documentation (/studio/write/java8-support#supported_features).

To keep the old behavior, specify Java 7 explicitly in your module-level `build.gradle.kts` or `build.gradle` file:

Groovy (#groovy)Kotlin (#kotlin)

```
// build.gradle.kts

android {
  ...
  compileOptions {
    sourceCompatibility = JavaVersion.VERSION_1_7
    targetCompatibility = JavaVersion.VERSION_1_7
  }
  // For Kotlin projects, compile to Java 6 instead of 7
  kotlinOptions {
    jvmTarget = "1.6"
  }
}
```

New JVM resource compiler

A new JVM resource compiler in Android Gradle plugin 4.2 tool replaces portions of the AAPT2 resource compiler (/studio/command-line/aapt2), potentially improving build performance, especially on Windows machines. The new JVM resource compiler is enabled by default.

v3 and v4 signing now supported

Android Gradle Plugin 4.2 now supports APK v3 (https://source.android.com/security/apksigning/v3) and APK v4 (https://source.android.com/security/apksigning/v4) signing formats. To enable one or both of these formats in your build, add the following properties to your module-level `build.gradle` or `build.gradle.kts` file:

Groovy (#groovy)Kotlin (#kotlin)

```
// build.gradle.kts

android {
   ...
   signingConfigs {
      config {
          ...
          enableV3Signing = true
          enableV4Signing = true
      }
   }
}
```

APK v4 signing allows you to quickly deploy large APKs using the ADB Incremental APK installation (/about/versions/11/features#incremental) in Android 11. This new flag takes care of the APK signing step in the deployment process.

Configure app signing per variant

It is now possible to enable or disable app signing (/reference/tools/gradle-api/4.2/com/android/build/api/variant/SigningConfig#summary) in Android Gradle plugin per variant.

This example demonstrates how to set app signing per variant using the `onVariants()` (/reference/tools/gradle-api/4.2/com/android/build/api/extension/AndroidComponentsExtension#onvariants) method in either Kotlin or Groovy:

```
androidComponents {
    onVariants(selector().withName("fooDebug"), {
        signingConfig.enableV1Signing.set(false)
        signingConfig.enableV2Signing.set(true)
    })
```

New Gradle property: `android.native.buildOutput`

To reduce clutter in build output, AGP 4.2 filters messages from native builds that use CMake (/ndk/guides/cmake) and `ndk-build` (/ndk/guides/ndk-build), displaying only C/C++ compiler output by default. Previously, a line of output was generated for every file that was built, resulting in a large quantity of informational messages.

If you would like to see the entirety of the native output, set the new Gradle property `android.native.buildOutput` to `verbose`.

You can set this property in either the `gradle.properties` file or through the command line.

gradle.properties
`android.native.buildOutput=verbose`

Command line
`-Pandroid.native.buildOutput=verbose`

The default value of this property is `quiet`.

Behavior change for gradle.properties files

Starting in AGP 4.2, it is no longer possible to override Gradle properties from subprojects. In other words, if you declare a property in a `gradle.properties` file in a subproject instead of the root project, it will be ignored.

As an example, in previous releases, AGP would read values from `*projectDir*/gradle.properties`, `*projectDir*/app/gradle.properties`, `*projectDir*/library/gradle.properties`, etc. For app modules, if the same Gradle property was present in both `*projectDir*/gradle.properties` and `*projectDir*/app/gradle.properties`, the value from `*projectDir*/app/gradle.properties` would take precedence.

In AGP 4.2, this behavior has been changed, and AGP won't load values from `gradle.properties` in subprojects (e.g., `*projectDir*/app/gradle.properties`). This change reflects the new Gradle behavior
(https://docs.gradle.org/current/userguide/build_environment.html#sec:gradle_configuration_properties)
and supports configuration caching
(https://medium.com/androiddevelopers/configuration-caching-deep-dive-bcb304698070).

For more information on setting values in `gradle.properties` files, see the Gradle docs
(https://docs.gradle.org/current/userguide/build_environment.html#sec:gradle_configuration_properties)
.

Gradle compatibility and configuration changes

When running in Android Studio, the Gradle build tool uses Studio's bundled JDK. In previous releases, JDK 8 was bundled with Studio. In 4.2, however, JDK 11 is now bundled instead. When using the new bundled JDK to run Gradle, this may result in some incompatibility or impact JVM performance due to changes to the garbage collector. These issues are described below.

Note: Although we recommend running Gradle with JDK 11, it is possible to change the JDK used to run Gradle in the **Project Structure** (/studio/projects#ProjectStructure) dialog. Changing this setting will only change the JDK used to run Gradle, and will not change the JDK used to run Studio itself.

Studio compatibility with Android Gradle plugin (AGP)

Android Studio 4.2 can open projects that use AGP 3.1 and higher provided that AGP is running Gradle 4.8.1 and higher. For more information about Gradle compatibility, see Update Gradle (#updating-gradle).

Optimizing Gradle builds for JDK 11

This update to JDK 11 impacts the default configuration of the JVM garbage collector, since JDK 8 uses the parallel garbage collector while JDK 11 uses the G1 garbage collector (https://docs.oracle.com/javase/9/gctuning/garbage-first-garbage-collector.htm#JSGCT-GUID-ED3AB6D3-FD9B-4447-9EDF-983ED2F7A573)
.

To potentially improve build performance, we recommend testing your Gradle builds (/studio/build/profile-your-build) with the parallel garbage collector. In `gradle.properties` set the following:

```
org.gradle.jvmargs=-XX:+UseParallelGC
```

If there are other options already set in this field, add a new option:

```
org.gradle.jvmargs=-Xmx1536m -XX:+UseParallelGC
```

To measure build speed with different configurations, see Profile your build (/studio/build/profile-your-build).

DEX files uncompressed in APKs when `minSdk` = 28 or higher

AGP now packages DEX files uncompressed in APKs by default when `minSdk` = 28 or higher. This causes an increase in APK size, but it results in a smaller installation size on the device, and the download size is roughly the same.

To force AGP to instead package the DEX files compressed, you can add the following to your `build.gradle` file:

```
android {
    packagingOptions {
        dex {
            useLegacyPackaging true
        }
    }
}
```

Use the DSL to package compressed native libraries

We recommend packaging native libraries in uncompressed form, because this results in a smaller app install size, smaller app download size, and faster app loading time for your users. However, if you want the Android Gradle plugin to package compressed native libraries when building your app, set `useLegacyPackaging` (/reference/tools/gradle-api/7.1/com/android/build/api/dsl/JniLibsPackagingOptions#uselegacypackaging) to `true` in your app's `build.gradle` file:

```
android {
    packagingOptions {
        jniLibs {
            useLegacyPackaging true
        }
    }
}
```

The flag `useLegacyPackaging` replaces the manifest attribute `extractNativeLibs`. For more background, see the release note Native libraries packaged uncompressed by default (#extractNativeLibs).

4.1.0 (August 2020)

Compatibility

	Minimum version	Default version	Notes
Gradle	6.5 (https://docs.gradle.org/6.5.1/release-notes.html)	N/A	To learn more, updating Gradl (#updating-gradl
SDK Build Tools	29.0.2 (/studio/releases/build-tools#notes)	29.0.2 (/studio/releases/build-tools#notes)	Install (/studio/intro/up manager) or configure (/studio/release tools) SDK Build Tool
NDK	N/A	21.1.6352462	Install (/studio/projects ndk#specific-vers or configure (/studio/projects ndk#apply-specif a different vers the NDK.

New features

This version of the Android Gradle plugin includes the following new features.

Kotlin Script DSL support

To help improve the editing experience for Kotlin buildscript users, the DSL and APIs of Android Gradle plugin 4.1 are now defined in a set of Kotlin interfaces separately from their implementation classes. This means that:

- Nullability and mutability are now explicitly declared on Kotlin types.
- Documentation generated from those interfaces is published in the Kotlin API Reference (/reference/tools/gradle-api).
- The API surface of the Android Gradle Plugin is clearly defined, to make extending Android builds less brittle in the future.

Important: If you have already adopted KTS build scripts or use Kotlin in `buildSrc`, this may cause source compatibility breakages for certain errors that would have manifest as run-time errors in previous releases.

Collection types that are designed to be mutated in the DSL are now uniformly defined as:

`val collection: MutableCollectionType`

This means that it is no longer possible to write the following in Kotlin scripts for some collections that previously supported it:

`collection = collectionTypeOf(...)`

However, mutating the collection is supported uniformly so `collection += …` and `collection.add(...)` should now work everywhere.

If you discover any issues when upgrading a project that uses Android Gradle plugin Kotlin APIs and DSL, please report a bug (https://issuetracker.google.com/issues/new?component=192709&template=842921).

Export C/C++ dependencies from AARs

Android Gradle plugin 4.0 added the ability to import Prefab packages in AAR dependencies (/studio/releases/gradle-plugin#native-dependencies). In AGP 4.1, it's now possible to export libraries from your external native build in an AAR for an Android Library project.

To export your native libraries, add the following to the `android` block of your library project's `build.gradle` file:

Groovy (#groovy)Kotlin(#kotlin)

```
buildFeatures {
    prefabPublishing = true
}

prefab {
    create("mylibrary") {
      headers = "src/main/cpp/mylibrary/include"
    }

    create("myotherlibrary") {
        headers = "src/main/cpp/myotherlibrary/include"
    }
}
```

In this example, the `mylibrary` and `myotherlibrary` libraries from either your ndk-build or CMake external native build will be packaged in the AAR produced by your build, and each will export the headers from the specified directory to their dependents.

Note: For users of Android Gradle plugin 4.0 and above, the configuration settings for importing prebuilt native libraries have changed. For more information, see the 4.0 release notes (#cmake-imported-targets).

R8 support for Kotlin metadata

Kotlin uses custom metadata in Java class files to identify Kotlin language constructs. R8 now has support for maintaining and rewriting Kotlin metadata to fully support shrinking of Kotlin libraries and applications using `kotlin-reflect`.

To keep Kotlin metadata, add the following keep rules:

```
-keep class kotlin.Metadata { *; }

-keepattributes RuntimeVisibleAnnotations
```

This will instruct R8 to keep Kotlin metadata for all classes that are directly kept.

For more information, see Shrinking Kotlin libraries and applications using Kotlin reflection with R8 (https://medium.com/androiddevelopers/shrinking-kotlin-libraries-and-applications-using-kotlin-reflection-with-r8-6fe0a0e2d115) on Medium.

Assertions in debug builds

When you build the debug version of your app using Android Gradle plugin 4.1.0 and higher, the built-in compiler (D8) will rewrite your app's code to enable assertions at compile time, so you always have assertion checks active.

Behavior changes

Android Gradle plugin build cache removed

The AGP build cache was removed in AGP 4.1. Previously introduced in AGP 2.3 to complement the Gradle build cache, the AGP build cache was superseded entirely by the Gradle build cache in AGP 4.1. This change does not impact build time.

The `cleanBuildCache` task and the `android.enableBuildCache` and `android.buildCacheDir` properties are deprecated and will be removed in AGP 7.0. The `android.enableBuildCache` property currently has no effect, while the `android.buildCacheDir` property and the `cleanBuildCache` task will be functional until AGP 7.0 for deleting any existing AGP build cache contents.

App size significantly reduced for apps using code shrinking

Starting with this release, fields from R classes are no longer kept by default (https://issuetracker.google.com/142449264), which may result in significant APK size savings for apps that enable code shrinking. This should not result in a behavior change unless you are accessing R classes by reflection, in which case it is necessary to add keep rules (/studio/build/shrink-code#keep-code) for those R classes.

android.namespacedRClass property renamed to android.nonTransitiveRClas

The experimental flag `android.namespacedRClass` has been renamed to `android.nonTransitiveRClass`.

Set in the `gradle.properties` file, this flag enables namespacing of each library's R class so that its R class includes only the resources declared in the library itself and none from the library's dependencies, thereby reducing the size of the R class for that library.

Kotlin DSL: coreLibraryDesugaringEnabled renamed

The Kotlin DSL compile option `coreLibraryDesugaringEnabled` has been changed to `isCoreLibraryDesugaringEnabled` (/reference/tools/gradle-api/4.1/com/android/build/api/dsl/CompileOptions#iscorelibrarydesugaringenabled). For more information about this flag, see Java 8+ API desugaring support (Android Gradle Plugin 4.0.0+) (/studio/write/java8-support#library-desugaring).

Version properties removed from BuildConfig class in library projects

For library projects only, the `BuildConfig.VERSION_NAME` and `BuildConfig.VERSION_CODE` properties have been removed from the generated `BuildConfig` class because these static values did not reflect the final values of the application's version code and name, and were therefore misleading. Additionally, these values were discarded during manifest merging.

In a future version of Android Gradle plugin, the `versionName` and `versionCode` properties will also be removed from the DSL for libraries. Currently, there is no way to automatically access the app version code/name from a library sub-project.

For application modules, there is no change, you can still assign values to `versionCode` and `versionName` in the DSL; these values will propagate to the app's manifest and `BuildConfig` fields.

Set the NDK path

You can set the path to your local NDK installation using the `android.ndkPath` property in your module's `build.gradle` file.

Groovy (#groovy) Kotlin (#kotlin)

```
android {
   ndkPath = "your-custom-ndk-path"
}
```

If you use this property together with the `android.ndkVersion` property (/studio/projects/install-ndk#apply-specific-version), then this path must contain an NDK version that matches `android.ndkVersion`.

Library unit test behavior changes

We've changed the behavior of how library unit tests are compiled and run. A library's unit tests are now compiled and run against compile/runtime classes of the library itself, resulting in the unit test consuming the library in the same way external subprojects do. This configuration typically results in better testing.

In some cases library unit tests that use data binding may encounter missing `DataBindingComponent` (/reference/android/databinding/DataBindingComponent) or `BR` (/topic/libraries/data-binding/generated-binding#dynamic_variables) classes. Those tests need to be ported to an instrumented test in the `androidTest` project, since compiling and running against those classes in a unit test may produce incorrect output.

io.fabric Gradle plugin deprecated

The io.fabric Gradle plugin is deprecated and is not compatible with version 4.1 of the Android Gradle plugin. For more information on the deprecated Fabric SDK and migrating to the Firebase Crashlytics SDK, see Upgrade to the Firebase Crashlytics SDK (https://firebase.google.com/docs/crashlytics/upgrade-sdk?platform=android).

4.0.0 (April 2020)

This version of the Android plugin requires the following:

- Gradle 6.1.1 (https://docs.gradle.org/6.1.1/release-notes.html). To learn more, read the section about updating Gradle (#updating-gradle).
- SDK Build Tools 29.0.2 (/studio/releases/build-tools#notes) or higher.

4.0.1 (July 2020)

This minor update supports compatibility with new default settings and features for package visibility in Android 11 (/about/versions/11/privacy/package-visibility).

In previous versions of Android, it was possible to view a list of all apps installed on a device. Starting with Android 11 (API level 30), by default apps have access to only a filtered list of installed packages. To see a broader list of apps on the system, you now need to add a `<queries>` element (/training/basics/intents/package-visibility#package-name) in your app or library's Android manifest.

Android Gradle plugin 4.1+ is already compatible with the new `<queries>` declaration; however, older versions are not compatible. If you add the `<queries>` element or if you start relying on a library or SDK that supports targeting Android 11, you may encounter manifest merging errors when building your app.

To address this issue, we're releasing a set of patches for AGP 3.3 and higher. If you're using an older version of AGP, upgrade (/studio/releases/gradle-plugin?buildsystem=ndk-build#updating-plugin) to one of the following versions:

If you are using AGP version...	...upgrade to:
4.0.*	4.0.1
3.6.*	3.6.4
3.5.*	3.5.4
3.4.*	3.4.3
3.3.*	3.3.3

For more information on this new feature, see Package visibility in Android 11 (/about/versions/11/privacy/package-visibility).

New features

This version of the Android Gradle plugin includes the following new features.

Support for Android Studio Build Analyzer

The **Build Analyzer** window helps you understand and diagnose issues with your build process, such as disabled optimizations and improperly configured tasks. This feature is available when you use Android Studio 4.0 and higher with Android Gradle plugin `4.0.0` and higher. You can open the **Build Analyzer** window from Android Studio as follows:

1. If you haven't already done so, build your app by selecting **Build > Make Project** from the menu bar.
2. Select **View > Tool Windows > Build** from the menu bar.
3. In the **Build** window, open the **Build Analyzer** window in one of the following ways:
    - After Android Studio finishes building your project, click the **Build Analyzer** tab.
    - After Android Studio finishes building your project, click the link on the right side of the **Build Output** window.

The **Build Analyzer** window organizes possible build issues in a tree on the left. You can inspect and click on each issue to investigate its details in the panel on the right. When Android Studio analyzes your build, it computes the set of tasks that determined the build's duration and provides a visualization to help you understand the impact of each of these tasks. You can also get details on warnings by expanding the **Warnings** node.

To learn more, read identify build speed regressions (/studio/build/build-analyzer).

Java 8 library desugaring in D8 and R8

The Android Gradle plugin now includes support for using a number of Java 8 language APIs without requiring a minimum API level for your app.

Through a process called *desugaring*, the DEX compiler, D8, in Android Studio 3.0 and higher already provided substantial support for Java 8 language features (such as lambda expressions, default interface methods, try with resources, and more). In Android Studio 4.0, the desugaring engine has been extended to be able to desugar Java language APIs. This means that you can now include standard language APIs that were available only in recent Android releases (such as `java.util.streams`) in apps that support older versions of Android.

The following set of APIs is supported in this release:

- Sequential streams (`java.util.stream`)
- A subset of `java.time`

- `java.util.function`
- Recent additions to `java.util.{Map,Collection,Comparator}`
- Optionals (`java.util.Optional`, `java.util.OptionalInt` and `java.util.OptionalDouble`) and some other new classes useful with the above APIs
- Some additions to `java.util.concurrent.atomic` (new methods on `AtomicInteger`, `AtomicLong` and `AtomicReference`)
- `ConcurrentHashMap` (with bug fixes for Android 5.0)

To support these language APIs, D8 compiles a separate library DEX file that contains an implementation of the missing APIs and includes it in your app. The desugaring process rewrites your app's code to instead use this library at runtime.

To enable support for these language APIs, include the following in your **app module**'s `build.gradle` file:

Groovy (#groovy)Kotlin (#kotlin)

```
android {
  defaultConfig {
    // Required when setting minSdkVersion to 20 or lower
    multiDexEnabled = true
  }

  compileOptions {
    // Flag to enable support for the new language APIs
    isCoreLibraryDesugaringEnabled = true
    // Sets Java compatibility to Java 8
    sourceCompatibility = JavaVersion.VERSION_1_8
    targetCompatibility = JavaVersion.VERSION_1_8
  }
}

dependencies {
  coreLibraryDesugaring("com.android.tools:desugar_jdk_libs:1.0.4")
}
```

Note that you may also need to include the above code snippet in a **library module**'s `build.gradle` file if:

- The library module's instrumented tests use these language APIs (either directly or through the library module or its dependencies). This is so that the missing APIs are provided for your instrumented test APK.
- You want to run lint on the library module in isolation. This is to help lint recognize valid usages of the language APIs and avoid reporting false warnings.

New options to enable or disable build features

Android Gradle plugin 4.0.0 introduces a new way to control which build features you want to enable and disable, such as View Binding and Data Binding. When new features are added, they will be disabled, by default. You can then use the `buildFeatures` block to enable only the features you want, and it helps you optimize the build performance for your project. You can set the options for each module in the module-level `build.gradle` file, as follows:

Groovy (#groovy)Kotlin (#kotlin)

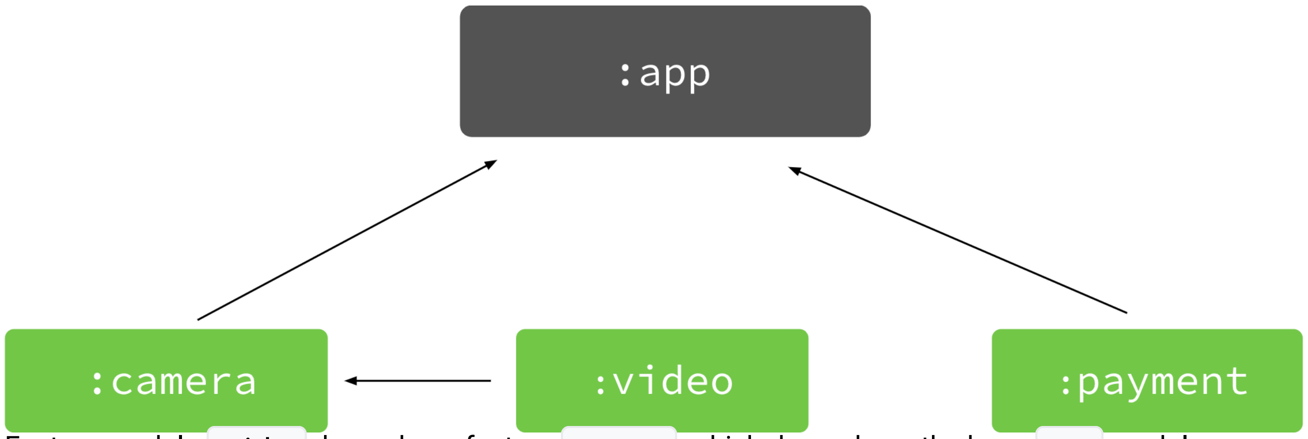
```
android {
    // The default value for each feature is shown below. You can change th
    // override the default behavior.
    buildFeatures {
        // Determines whether to generate a BuildConfig class.
        buildConfig = true
        // Determines whether to support View Binding.
        // Note that the viewBinding.enabled property is now deprecated.
        viewBinding = false
        // Determines whether to support Data Binding.
        // Note that the dataBinding.enabled property is now deprecated.
        dataBinding = false
        // Determines whether to generate binder classes for your AIDL file
        aidl = true
        // Determines whether to support RenderScript.
        renderScript = true
        // Determines whether to support injecting custom variables into th
        resValues = true
        // Determines whether to support shader AOT compilation.
        shaders = true
    }
}
```

You can also specify the default setting for these features across all modules in a project by including one or more of the following in your project's `gradle.properties` file, as shown below. Keep in mind, you can still use the `buildFeatures` block in the module-level `build.gradle` file to override these project-wide default settings.

```
android.defaults.buildfeatures.buildconfig=true
android.defaults.buildfeatures.aidl=true
android.defaults.buildfeatures.renderscript=true
android.defaults.buildfeatures.resvalues=true
android.defaults.buildfeatures.shaders=true
```

Feature-on-feature dependencies

In previous versions of the Android Gradle plugin, all feature modules could depend only on the app's base module. When using Android Gradle plugin 4.0.0, you can now include a feature module that depends on another feature module. That is, a `:video` feature can depend on the `:camera` feature, which depends on the base module, as shown in the figure below.



Feature module `:video` depends on feature `:camera`, which depends on the base `:app` module.

This means that when your app requests to download a feature module, the app also downloads other feature modules it depends on. After you create feature modules (/studio/projects/dynamic-delivery/on-demand-delivery) for your app, you can declare a feature-on-feature dependency in the module's `build.gradle` file. For example, the `:video` module declares a dependency on `:camera` as follows:

Groovy (#groovy)Kotlin (#kotlin)

```
// In the build.gradle file of the ':video' module.
dependencies {
    // All feature modules must declare a dependency
    // on the base module.
    implementation(project(":app"))
    // Declares that this module also depends on the 'camera'
    // feature module.
    implementation(project(":camera"))
    ...
}
```

Additionally, you should enable the feature-on-feature dependency feature in Android Studio (to support the feature when editing the Run configuration, for example) by clicking **Help > Edit Custom VM Options** from the menu bar and including the following:

```
-Drundebug.feature.on.feature=true
```

Dependencies metadata

When building your app using Android Gradle plugin 4.0.0 and higher, the plugin includes metadata that describes the dependencies that are compiled into your app. When uploading your app, the Play Console inspects this metadata to provide you with the following benefits:

- Get alerts for known issues with SDKs and dependencies your app uses
- Receive actionable feedback to resolve those issues

The data is compressed, encrypted by a Google Play signing key, and stored in the signing block of your release app. However, you can inspect the metadata yourself in the local intermediate build files in the following directory: `<project>/<module>/build/outputs/sdk-dependencies/release/sdkDependency.txt`.

If you'd rather not share this information, you can opt-out by including the following in your module's `build.gradle` file:

(#kotlin)

```
android {
    dependenciesInfo {
        // Disables dependency metadata when building APKs.
        includeInApk = false
        // Disables dependency metadata when building Android App Bundles.
        includeInBundle = false
    }
}
```

Import native libraries from AAR dependencies

You can now import C/C++ libraries from your app's AAR dependencies. When you follow the configuration steps described below, Gradle automatically makes these native libraries available to use with your external native build system, such as CMake. Note that Gradle only makes these libraries available to your build; you must still configure your build scripts to use them.

Libraries are exported using the Prefab (https://google.github.io/prefab/) package format.

Each dependency can expose at most one Prefab package, which comprises one or more modules. A Prefab module is a single library, which could be either a shared, static, or header-only library.

Typically, the package name matches the Maven artifact name and the module name matches the library name, but this is not always true. Because you need to know the package and module name of the libraries, you might need to consult the dependency's documentation to determine what those names are.

Configure your external native build system

To see the steps you need to follow, click on the external native build system you plan to use.

✓ CMake ndk-build

Native dependencies included in an AAR are exposed to your CMake project via the CMAKE_FIND_ROOT_PATH (https://cmake.org/cmake/help/latest/variable/CMAKE_FIND_ROOT_PATH.html) variable. This value will be set automatically by Gradle when CMake is invoked, so if your build system modifies this variable, be sure to append rather than assign to it.

Each dependency exposes a config-file package (https://cmake.org/cmake/help/latest/manual/cmake-packages.7.html#config-file-packages) to your CMake build, which you import with the `find_package` (https://cmake.org/cmake/help/latest/command/find_package.html) command. This command searches for config-file packages that match the given package name and version and exposes the targets it defines to be used in your build. For example, if your application defines `libapp.so` and it uses curl, you should include the following in your `CMakeLists.txt` file:

```
add_library(app SHARED app.cpp)

# Add these two lines.
find_package(curl REQUIRED CONFIG)
target_link_libraries(app curl::curl)
```

You can now specify `#include "curl/curl.h"` in `app.cpp`. When you build your project, your external native build system automatically links `libapp.so` against `libcurl.so` and packages `libcurl.so` in the APK or app bundle. For additional information, refer to the curl prefab sample (https://github.com/android/ndk-samples/tree/main/prefab/curl-ssl).

Behavior changes

When using this version of the plugin, you might encounter the following changes in behavior.

v1/v2 signing configuration updates

The behavior for app signing configurations in the `signingConfig` block has changed to the following:

v1 signing

- If `v1SigningEnabled` is explicitly enabled, AGP performs v1 app signing.
- If `v1SigningEnabled` is explicitly disabled by the user, v1 app signing is not performed.

- If the user has not explicitly enabled v1 signing, it can be automatically disabled based on `minSdk` and `targetSdk`.

v2 signing

- If `v2SigningEnabled` is explicitly enabled, AGP performs v2 app signing.
- If `v2SigningEnabled` is explicitly disabled by the user, v2 app signing is not performed.
- If the user has not explicitly enabled v2 signing, it can be automatically disabled based on `targetSdk`.

These changes allow AGP to optimize builds by disabling the signing mechanism based on whether the user has explicitly enabled these flags. Prior to this release, it was possible for `v1Signing` to be disabled even when explicitly enabled, which could be confusing.

`feature` and `instantapp` Android Gradle plugins removed

Android Gradle plugin 3.6.0 deprecated the Feature plugin (`com.android.feature`) and the Instant App plugin (`com.android.instantapp`) in favor of using the Dynamic Feature plugin (`com.android.dynamic-feature`) to build and package your instant apps using Android App Bundles (/guide/app-bundle).

In Android Gradle plugin 4.0.0 and higher, these deprecated plugins are fully removed. So, to use the latest Android Gradle plugin, you need to migrate your instant app to support Android App Bundles (/topic/google-play-instant/feature-module-migration). By migrating your instant apps, you can leverage the benefits of app bundles and simplify your app's modular design (https://android-developers.googleblog.com/2019/04/google-play-instant-feature-plugin.html).

Note: To open projects that use the removed plugins in Android Studio 4.0 and higher, the project must use Android Gradle plugin 3.6.0 or lower.

Separate annotation processing feature removed

The ability to separate annotation processing into a dedicated task has been removed. This option was used to maintain incremental Java compilation when non-incremental annotation processors are used in Java-only projects; it was enabled by setting `android.enableSeparateAnnotationProcessing` to `true` in the `gradle.properties` file, which no longer works.

Instead, you should migrate to using incremental annotation processors (/studio/build/optimize-your-build#annotation_processors) to improve build performance.

includeCompileClasspath is deprecated

The Android Gradle plugin no longer checks for or includes annotation processors you declare on the compile classpath, and the `annotationProcessorOptions.includeCompileClasspath` DSL property no longer has any effect. If you include annotation processors on the compile classpath, you might get the following error:

```
Error: Annotation processors must be explicitly declared now.
```

To resolve this issue, you must include annotation processors in your `build.gradle` files using the `annotationProcessor` dependency configuration. To learn more, read Add annotation processors (/studio/build/dependencies#annotation_processor).

Automatic packaging of prebuilt dependencies used by CMake

Prior versions of the Android Gradle Plugin required that you explicitly package any prebuilt libraries used by your CMake external native build by using `jniLibs`. You may have libraries in the `src/main/jniLibs` directory of your module, or possibly in some other directory configured in your `build.gradle` file:

Groovy (#groovy)Kotlin (#kotlin)

```
sourceSets {
    main {
        // The libs directory contains prebuilt libraries that are used by
        // app's library defined in CMakeLists.txt via an IMPORTED target.
        jniLibs.setSrcDirs(listOf("libs"))
    }
}
```

With Android Gradle Plugin 4.0, the above configuration is no longer necessary and will result in a build failure:

```
* What went wrong:
Execution failed for task ':app:mergeDebugNativeLibs'.
> A failure occurred while executing com.android.build.gradle.internal.tasks.Work
   > More than one file was found with OS independent path 'lib/x86/libprebuilt.s
```

External native build now automatically packages those libraries, so explicitly packaging the library with `jniLibs` results in a duplicate. To avoid the build error, move the prebuilt library to a location outside `jniLibs` or remove the `jniLibs` configuration from your `build.gradle` file.

Known issues

This section describes known issues that exist in Android Gradle plugin 4.0.0.

Race condition in Gradle worker mechanism

Changes in Android Gradle plugin 4.0 can trigger a race condition in Gradle when running with `--no-daemon` and versions of Gradle 6.3 or lower, causing builds to hang after the build is finished.

This issue will be fixed in Gradle 6.4.

3.6.0 (February 2020)

This version of the Android plugin requires the following:

- Gradle 5.6.4 (https://docs.gradle.org/5.6.4/release-notes.html). To learn more, read the section about updating Gradle (#updating-gradle).
- SDK Build Tools 28.0.3 (/studio/releases/build-tools#notes) or higher.

3.6.4 (July 2020)

This minor update supports compatibility with new default settings and features for package visibility in Android 11 (/about/versions/11/privacy/package-visibility).

See the 4.0.1 release notes (#4.0.1) for details.

New features

This version of the Android Gradle plugin includes the following new features.

View Binding

View binding provides compile-time safety when referencing views in your code. You can now replace `findViewById()` with the auto-generated binding class reference. To start using View binding, include the following in each module's `build.gradle` file:

Groovy (#groovy)Kotlin (#kotlin)

```
android {
    viewBinding.enabled = true
}
```

To learn more, read the View Binding documentation (/topic/libraries/view-binding).

Support for the Maven Publish plugin

The Android Gradle plugin includes support for the Maven Publish Gradle plugin (https://docs.gradle.org/current/userguide/publishing_maven.html), which allows you to publish build artifacts to an Apache Maven repository. The Android Gradle plugin creates a *component* (https://docs.gradle.org/current/userguide/dependency_management_terminology.html#sub:terminology_component) for each build variant artifact in your app or library module that you can use to customize a *publication* (https://docs.gradle.org/current/userguide/publishing_maven.html#publishing_maven:publications) to a Maven repository.

To learn more, go to the page about how to use the Maven Publish plugin (/studio/build/maven-publish-plugin).

New default packaging tool

When building the debug version of your app, the plugin uses a new packaging tool, called *zipflinger*, to build your APK. This new tool should provide build speed improvements. If the new packaging tool doesn't work as you expect, please report a bug (/studio/report-bugs). You can revert to using the old packaging tool by including the following in your `gradle.properties` file:

```
android.useNewApkCreator=false
```

Native build attribution

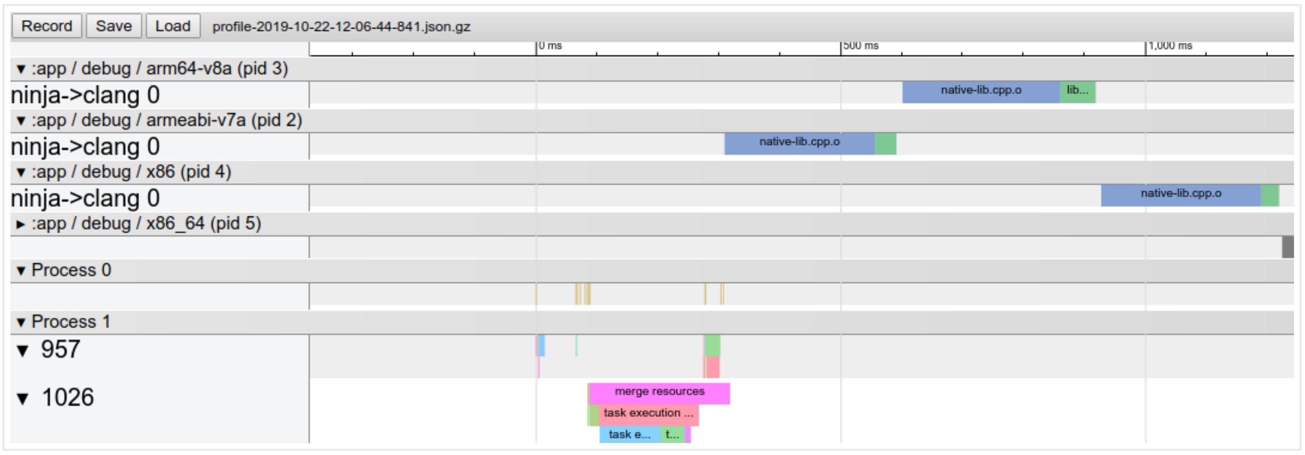
You can now determine the length of time it takes Clang to build and link each C/C++ file in your project. Gradle can output a Chrome trace that contains timestamps for these compiler events so you can better understand the time required to build your project. To output this build attribution file, do the following:

1. Add the flag `-Pandroid.enableProfileJson=true` when running a Gradle build. For example:

   `gradlew assembleDebug -Pandroid.enableProfileJson=true`

2. Open the Chrome browser and type `chrome://tracing` in the search bar.
3. Click the **Load** button and navigate to `project-root/build/android-profile` to find the file. The file is named `profile-timestamp.json.gz`.

You can see the native build attribution data near the top of the viewer:



Behavior changes

When using this version of the plugin, you might encounter the following changes in behavior.

Native libraries packaged uncompressed by default

When you build your app, the plugin now sets `extractNativeLibs` to `"false"` by default. That is, your native libraries are page aligned and packaged uncompressed. While this results in a larger upload size, your users benefit from the following:

- Smaller app install size because the platform can access the native libraries directly from the installed APK, without creating a copy of the libraries.
- Smaller download size because Play Store compression is typically better when you include uncompressed native libraries in your APK or Android App Bundle.

If you want the Android Gradle plugin to instead package compressed native libraries, include the following in your app's manifest:

```
<application
    android:extractNativeLibs="true"
    ... >
</application>
```

Note: The **`extractNativeLibs`** manifest attribute has been replaced by the **`useLegacyPackaging`** DSL option. For more information, see the release note Use the DSL to package compressed native libraries (#compress-native-libs-dsl).

Default NDK version

If you download multiple versions of the NDK, the Android Gradle plugin now selects a default version to use in compiling your source code files. Previously, the plugin selected the latest downloaded version of the NDK. Use the `android.ndkVersion` property in the module's `build.gradle` file to override the plugin-selected default.

Simplified R class generation

The Android Gradle plugin simplifies the compile classpath by generating only one R class for each library module in your project and sharing those R classes with other module dependencies. This optimization should result in faster builds, but it requires that you keep the following in mind:

- Because the compiler shares R classes with upstream module dependencies, it's important that each module in your project uses a unique package name.
- The visibility of a library's R class to other project dependencies is determined by the configuration used to include the library as a dependency. For example, if Library A includes Library B as an 'api' dependency, Library A and other libraries that depend on Library A have access to Library B's R class. However, other libraries might not have access to Library B's R class If Library A uses the `implementation` dependency configuration. To learn more, read about dependency configurations (/studio/build/dependencies#dependency_configurations).

Remove resources missing from default configuration

For Library modules, if you include a resource for a language that you do not include in the default set of resources—for example, if you include `hello_world` as a string resource in `/values-es/strings.xml` but you don't define that resource in `/values/strings.xml`—the Android Gradle plugin no longer includes that resource when compiling your project. This behavior change should result in fewer `Resource Not Found` runtime exceptions and improved build speed.

D8 now respects CLASS retention policy for annotations

When compiling your app, D8 now respects when annotations apply a CLASS retention policy, and those annotations are no longer available at runtime. This behavior also exists when setting the app's target SDK to API level 23, which previously allowed access to these annotations during runtime when compiling your app using older versions of the Android Gradle plugin and D8.

Other behavior changes

- `aaptOptions.noCompress` is no longer case sensitive on all platforms (for both APK and bundles) and respects paths that use uppercase characters.
- Data binding is now incremental by default. To learn more, see issue #110061530 (https://issuetracker.google.com/110061530).
- All unit tests, including Roboelectric unit tests, are now fully cacheable. To learn more, see issue #115873047 (https://issuetracker.google.com/115873047).

Bug fixes

This version of the Android Gradle plugin includes the following bug fixes:

- Robolectric unit tests are now supported in library modules that use data binding. To learn more, see issue #126775542 (https://issuetracker.google.com/126775542).
- You can now run `connectedAndroidTest` tasks across multiple modules while Gradle's parallel execution mode (https://guides.gradle.org/performance/#parallel_execution) is enabled.

Known issues

This section describes known issues that exist in Android Gradle plugin 3.6.0.

Slow performance of Android Lint task

Android Lint can take much longer to complete on some projects due to a regression in its parsing infrastructure, resulting in slower computation of inferred types for lambdas in certain code constructs.

The issue is reported as a bug in IDEA (https://youtrack.jetbrains.com/issue/IDEA-229655) and will be fixed in Android Gradle Plugin 4.0.

Missing Manifest class

If your app defines custom permissions in its manifest, the Android Gradle plugin typically generates a `Manifest.java` class that includes your custom permissions as string constants. The plugin packages this class with your app, so you can more easily reference those permissions at runtime.

Generating the manifest class is broken in Android Gradle plugin 3.6.0. If you build your app with this version of the plugin, and it references the manifest class, you might see a `ClassNotFoundException` exception. To resolve this issue, do one of the following:

- Reference your custom permissions by their fully-qualified name. For example, `"com.example.myapp.permission.DEADLY_ACTIVITY"`.
- Define your own constants, as shown below:

```
public final class CustomPermissions {
  public static final class permission {
    public static final String DEADLY_ACTIVITY="com.example.myapp.permission
  }
}
```

3.5.0 (August 2019)

Android Gradle plugin 3.5.0, along with Android Studio 3.5 (/studio/releases#3-5-0), is a major release and a result of Project Marble, which is a focus on improving three main areas of the Android developer tools: system health, feature polish, and fixing bugs. Notably, improving project build speed (https://medium.com/androiddevelopers/improving-build-speed-in-android-studio-3e1425274837) was a main focus for this update.

For information about these and other Project Marble updates, read the Android Developers blog post (https://android-developers.googleblog.com/2019/05/android-studio-35-beta.html) or the sections below.

This version of the Android plugin requires the following:

- Gradle 5.4.1 (https://docs.gradle.org/5.4.1/release-notes.html). To learn more, read the section about updating Gradle (#updating-gradle).
- SDK Build Tools 28.0.3 (/studio/releases/build-tools#notes) or higher.

3.5.4 (July 2020)

This minor update supports compatibility with new default settings and features for package visibility in Android 11 (/about/versions/11/privacy/package-visibility).

See the 4.0.1 release notes (#4.0.1) for details.

3.5.3 (December 2019)

This minor update supports Android Studio 3.5.3 and includes various bug fixes and performance improvements.

3.5.2 (November 2019)

This minor update supports Android Studio 3.5.2 and includes various bug fixes and performance improvements. To see a list of noteable bug fixes, read the related post on the Release Updates blog (https://androidstudio.googleblog.com/2019/11/android-studio-352-available.html).

3.5.1 (October 2019)

This minor update supports Android Studio 3.5.1 and includes various bug fixes and performance improvements. To see a list of noteable bug fixes, read the related post on the Release Updates blog (https://androidstudio.googleblog.com/2019/10/android-studio-351-available.html).

Incremental annotation processing

The Data Binding (/reference/android/databinding/package-summary) annotation processor supports incremental annotation processing (https://docs.gradle.org/current/userguide/java_plugin.html#sec:incremental_annotation_processing) if you set `android.databinding.incremental=true` in your `gradle.properties` file. This optimization results in improved incremental build performance. For a full list of optimized annotation processors, refer to the table of incremental annotation processors (https://docs.gradle.org/current/userguide/java_plugin.html#state_of_support_in_popular_annotation_processors)
.

Additionally, KAPT 1.3.30 and higher also support incremental annotation processors, which you can enable by including `kapt.incremental.apt=true` in your `gradle.properties` file.

Cacheable unit tests

When you enable unit tests to use Android resources, assets, and manifests by setting `includeAndroidResources` (https://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.TestOptions.UnitTestOptions.html#com.android.build.gradle.internal.dsl.TestOptions.UnitTestOptions:includeAndroidResources) to `true`, the Android Gradle plugin generates a test config file containing absolute paths, which breaks cache relocatability. You can instruct the plugin to instead generate the test config using relative paths, which allows the `AndroidUnitTest` task to be fully cacheable, by including the following in your `gradle.properties` file:

```
android.testConfig.useRelativePath = true
```

Known issues

- When using Kotlin Gradle plugin 1.3.31 or earlier, you might see the following warning when building or syncing your project:

  ```
  WARNING: API 'variant.getPackageLibrary()' is obsolete and has been replaced
           with 'variant.getPackageLibraryProvider()'.
  ```

  To resolve this issue (https://youtrack.jetbrains.com/issue/KT-30784), upgrade the plugin to version 1.3.40 or higher.

3.4.0 (April 2019)

This version of the Android plugin requires the following:

- Gradle 5.1.1 (https://docs.gradle.org/5.1.1/release-notes.html) or higher. To learn more, read the section about updating Gradle (#updating-gradle).

  ★ **Note:** When using Gradle 5.0 and higher, the default Gradle daemon memory heap size (https://docs.gradle.org/current/userguide/upgrading_version_4.html#rel5.0:default_memory_settings) decreases from 1 GB to 512 MB. This might result in a build performance regression. To override this default setting, specify the Gradle daemon heap size (https://docs.gradle.org/current/userguide/build_environment.html#sec:configuring_jvm_memory) in your project's **gradle.properties** file.

- SDK Build Tools 28.0.3 (/studio/releases/build-tools#notes) or higher.

3.4.3 (July 2020)

This minor update supports compatibility with new default settings and features for package visibility in Android 11 (/about/versions/11/privacy/package-visibility).

See the 4.0.1 release notes (#4.0.1) for details.

3.4.2 (July 2019)

This minor update supports Android Studio 3.4.2 and includes various bug fixes and performance improvements. To see a list of noteable bug fixes, read the related post on the Release Updates blog (https://androidstudio.googleblog.com/2019/07/android-studio-342-available.html).

3.4.1 (May 2019)

This minor update supports Android Studio 3.4.1 and includes various bug fixes and performance improvements. To see a list of noteable bug fixes, read the related post on the Release Updates blog (https://androidstudio.googleblog.com/2019/05/android-studio-341-available.html).

New features

- **New lint check dependency configurations:** The behavior of `lintChecks` has changed and a new dependency configuration, `lintPublish`, has been introduced to give you more control over which lint checks are packaged in your Android libraries.
  - `lintChecks`: This is an existing configuration that you should use for lint checks you want to only run when building your project locally. If you were previously using the `lintChecks` dependency configuration to include lint checks in the published AAR, you need to migrate those dependencies to instead use the new `lintPublish` configuration described below.

- `lintPublish`: Use this new configuration in library projects for lint checks you want to include in the published AAR, as shown below. This means that projects that consume your library also apply those lint checks.

The following code sample uses both dependency configurations in a local Android library project.

Groovy (#groovy)Kotlin (#kotlin)

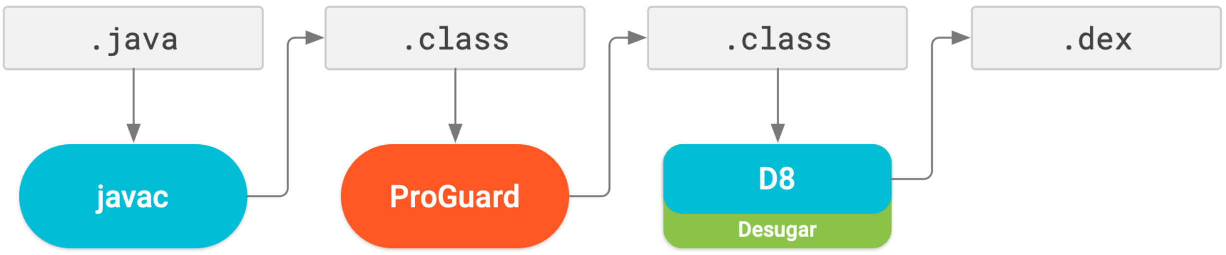
```
dependencies {
// Executes lint checks from the ':lint' project at build time.
lintChecks(project(":lint"))
// Packages lint checks from the ':lintpublish' in the published AAR.
lintPublish(project(":lintpublish"))
}
```

- In general, packaging and signing tasks should see an overall build speed improvement. If you notice a performance regression related to these tasks, please report a bug (/studio/report-bugs).

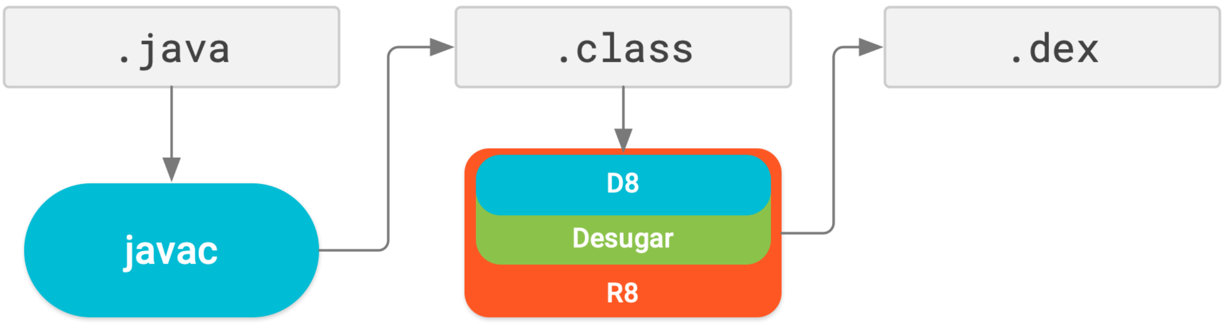
Behavior changes

- **Android Instant Apps Feature plugin deprecation warning:** If you're still using the `com.android.feature` plugin to build your instant app, Android Gradle plugin 3.4.0 will give throw you a deprecation warning. To make sure you can still build you instant app on future versions of the plugin, migrate your instant app to using the dynamic feature plugin (/studio/projects/dynamic-delivery), which also allows you to publish both your installed and instant app experiences from a single Android App Bundle.
- **R8 enabled by default:** R8 integrates desugaring, shrinking, obfuscating, optimizing, and dexing all in one step—resulting in noticeable build performance improvements (https://www.google.com/url?q=https://android-developers.googleblog.com/2018/11/r8-new-code-shrinker-from-google-is.html&sa=D&ust=1551922493258000&usg=AFQjCNH0N1wuMX645n7giw0wjikzjm3WCA). R8 was introduced in Android Gradle plugin 3.3.0 and is now enabled by default for both app and Android library projects using plugin 3.4.0 and higher.

  The image below provides a high-level overview of the compile process before R8 was introduced.



Now, with R8, desugaring, shrinking, obfuscating, optimizing, and dexing (D8) are all completed in one step, as illustrated below.



Keep in mind, R8 is designed to work with your existing ProGuard rules, so you'll likely not need to take any actions to benefit from R8. However, because it's a different technology to ProGuard that's designed specifically for Android projects, shrinking and optimization may result in removing code that ProGuard may have not. So, in this unlikely situation, you might need to add additional rules to keep that code in your build output.

If you experience issues using R8, read the R8 compatibility FAQ (https://r8.googlesource.com/r8/+/refs/heads/master/compatibility-faq.md) to check if there's a solution to your issue. If a solution isn't documented, please report a bug (https://issuetracker.google.com/issues/new?component=326788&template=1025938). You can disable R8 by adding one of the following lines to your project's `gradle.properties` file:

```
# Disables R8 for Android Library modules only.
android.enableR8.libraries = false
# Disables R8 for all modules.
android.enableR8 = false
```

★ **Note:** For a given build type, if you set **`useProguard`** to **`false`** in your app module's **`build.gradle`** file, the Android Gradle plugin uses R8 to shrink your app's code for that build type, regardless of whether you disable R8 in your project's **`gradle.properties`** file.

- `ndkCompile` **is deprecated:** You now get a build error if you try to use `ndkBuild` to compile your native libraries. You should instead use either CMake or ndk-build to Add C and C++ code to your project (/studio/projects/add-native-code).

Known issues

- The correct usage of unique package names are currently not enforced but will become more strict on later versions of the plugin. On Android Gradle plugin version 3.4.0, you can opt-in to check whether your project declares acceptable package names by adding the line below to your `gradle.properties` file.

  ```
  android.uniquePackageNames = true
  ```

  To learn more about setting a package name through the Android Gradle plugin, see Set the application ID (/studio/build/configure-app-module#set_the_application_id).

3.3.0 (January 2019)

This version of the Android plugin requires the following:

- Gradle 4.10.1 (https://docs.gradle.org/4.10.1/release-notes.html) or higher. To learn more, read the section about updating Gradle (#updating-gradle).

  ★ **Note:** When using Gradle 5.0 and higher, the default Gradle daemon memory heap size (https://docs.gradle.org/current/userguide/upgrading_version_4.html#rel5.0:default_memory_settings) decreases from 1 GB to 512 MB. This might result in a build performance regression. To override this default setting, specify the Gradle daemon heap size (https://docs.gradle.org/current/userguide/build_environment.html#sec:configuring_jvm_memory) in your project's `gradle.properties` file.

- SDK Build Tools 28.0.3 (/studio/releases/build-tools#notes) or higher.

3.3.3 (July 2020)

This minor update supports compatibility with new default settings and features for package visibility in Android 11 (/about/versions/11/privacy/package-visibility).

See the 4.0.1 release notes (#4.0.1) for details.

3.3.2 (March 2019)

This minor update supports Android Studio 3.3.2 and includes various bug fixes and performance improvements. To see a list of noteable bug fixes, read the related post on the Release Updates blog (https://androidstudio.googleblog.com/2019/03/android-studio-332-available.html).

3.3.1 (February 2019)

This minor update supports Android Studio 3.3.1 and includes various bug fixes and performance improvements.

New features

- **Improved classpath synchronization:** When resolving dependencies on your runtime and compile time classpaths, the Android Gradle plugin attempts to fix certain downstream version conflicts for dependencies that appear across multiple classpaths.

  For example, if the runtime classpath includes Library A version 2.0 and the compile classpath includes Library A version 1.0, the plugin automatically updates the dependency on the compile classpath to Library A version 2.0 to avoid errors.

  However, if the runtime classpath includes Library A version 1.0 and the compile includes Library A version 2.0, the plugin does not downgrade the dependency on the compile classpath to Library A version 1.0, and you will get an error. To learn more, see Fix conflicts between classpaths (/studio/build/dependencies#classpath_conflicts).

- **Improved incremental Java compilation when using annotation processors:** This update decreases build time by improving support for incremental Java compilation when using annotation processors.

★ **Note:** This feature is compatible with Gradle 4.10.1 and higher, except Gradle 5.1 due to Gradle issue 8194 (https://github.com/gradle/gradle/issues/8194).

- **For projects using Kapt (most Kotlin-only projects and Kotlin-Java hybrid projects):** Incremental Java compilation is enabled, even when you use data binding or the retro-lambda plugin. Annotation processing by the Kapt task is not yet incremental.
- **For projects not using Kapt (Java-only projects):** If the annotation processors you use all support incremental annotation processing (https://docs.gradle.org/4.10.1/userguide/java_plugin.html#sec:incremental_annotation_processing), incremental Java compilation is enabled by default. To monitor incremental annotation processor adoption, watch Gradle issue 5277 (https://github.com/gradle/gradle/issues/5277).

  If, however, one or more annotation processors do not support incremental builds, incremental Java compilation is not enabled. Instead, you can include the following flag in your `gradle.properties` file:

  ```
  android.enableSeparateAnnotationProcessing=true
  ```

  When you include this flag, the Android Gradle plugin executes the annotation processors in a separate task and allows the Java compilation task to run incrementally.

- **Better debug info when using obsolete API:** When the plugin detects that you're using an API that's no longer supported, it can now provide more-detailed information to help you determine where that API is being used. To see the additional info, you need to include the following in your project's `gradle.properties` file:

  ```
  android.debug.obsoleteApi=true
  ```

  You can also enable the flag by passing `-Pandroid.debug.obsoleteApi=true` from the command line.

- You can run instrumentation tests on feature modules from the command line.

Behavior changes

- **Lazy task configuration:** The plugin now uses Gradle's new task creation API (https://docs.gradle.org/current/userguide/task_configuration_avoidance.html) to avoid initializing and configuring tasks that are not required to complete the current build (or tasks not on the execution task graph). For example, if you have multiple build variants, such as "release" and "debug" build variants, and you're building the "debug" version of your app, the plugin avoids initializing and configuring tasks for the "release" version of your app.

  Calling certain older methods in the Variants API, such as `variant.getJavaCompile()`, might still force task configuration. To make sure that your build is optimized for lazy task configuration, invoke new methods that instead return a `TaskProvider` (https://docs.gradle.org/current/javadoc/org/gradle/api/tasks/TaskProvider.html) object, such as `variant.getJavaCompileProvider()`.

  If you execute custom build tasks, learn how to adapt to Gradle's new task-creation API (https://docs.gradle.org/current/userguide/task_configuration_avoidance.html#sec:old_vs_new_configuration_api_overview)
  .

- For a given build type, when setting `useProguard false`, the plugin now uses R8 instead of ProGuard to shrink and obfuscate your app's code and resources. To learn more about R8, read this blog post (https://android-developers.googleblog.com/2018/11/r8-new-code-shrinker-from-google-is.html) from the Android Developer's Blog.

- **Faster R class generation for library projects:** Previously, the Android Gradle plugin would generate an `R.java` file for each of your project's dependencies and then compile those R classes alongside your app's other classes. The plugin now generates a JAR containing your app's compiled R class directly, without first building intermediate `R.java` classes. This optimization may significantly improve build performance for projects that include many library subprojects and dependencies, and improve the indexing speed in Android Studio.

- When building an Android App Bundle (/guide/app-bundle), APKs generated from that app bundle that target Android 6.0 (API level 23) or higher now include uncompressed versions of your native libraries by default. This optimization avoids the need for the device to make a copy of the library and thus reduces the on-disk size of your app. If you'd rather disable this optimization, add the following to your `gradle.properties` file:

```
android.bundle.enableUncompressedNativeLibs = false
```

- The plugin enforces minimum versions of some third-party plugins.
- **Single-variant project sync**: Syncing your project (/studio/build#sync-files) with your build configuration is an important step in letting Android Studio understand how your project is structured. However, this process can be time-consuming for large projects. If your project uses multiple build variants, you can now optimize project syncs by limiting them to only the variant you have currently selected.

  You need to use Android Studio 3.3 or higher with Android Gradle Plugin 3.3.0 or higher to enable this optimization. When you meet these requirements, the IDE prompts you to enable this optimization when you sync your project. The optimization is also enabled by default on new projects.

  To enable this optimization manually, click **File > Settings > Experimental > Gradle** (**Android Studio > Preferences > Experimental > Gradle** on a Mac) and select the **Only sync the active variant** checkbox.

★ **Note**: This optimization fully supports projects that include Java and C++ languages, and has some support for Kotlin. When enabling the optimization for projects with Kotlin content, Gradle sync falls back to using full variants internally.

- **Automatic downloading of missing SDK packages**: This functionality has been expanded to support NDK. To learn more, read Auto-download missing packages with Gradle (/studio/intro/update#download-with-gradle).

Bug Fixes

- Android Gradle plugin 3.3.0 fixes the following issues:
  - The build process calling `android.support.v8.renderscript.RenderScript` instead of the AndroidX version, despite Jetifier being enabled
  - Clashes due to `androidx-rs.jar` including statically bundled `annotation.AnyRes`
  - When using RenderScript, you no longer have to manually set the Build Tools version in your `build.gradle` files

3.2.0 (September 2018)

This version of the Android plugin requires the following:

- Gradle 4.6 (https://docs.gradle.org/4.6/release-notes.html) or higher. To learn more, read the section about updating Gradle (#updating-gradle).
- SDK Build Tools 28.0.3 (/studio/releases/build-tools#notes) or higher.

3.2.1 (October 2018)

With this update, you no longer need to specify a version for the SDK Build Tools. The Android Gradle plugin now uses version 28.0.3 by default.

New features

- **Support for building Android App Bundles:** The app bundle is a new upload format that includes all your app's compiled code and resources while deferring APK generation and signing to the Google Play Store. You no longer have to build, sign, and manage multiple APKs, and users get smaller downloads that are optimized for their device. To learn more, read About Android App Bundles (/guide/app-bundle).
- **Support for improved incremental build speeds when using annotation processors:** The `AnnotationProcessorOptions` (https://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.AnnotationProcessorOptions.html) DSL now extends `CommandLineArgumentProvider` (https://docs.gradle.org/current/javadoc/org/gradle/process/CommandLineArgumentProvider.html) , which enables either you or the annotation processor author to annotate arguments for the processor using incremental build property type annotations (https://docs.gradle.org/current/userguide/more_about_tasks.html#sec:up_to_date_checks). Using these annotations improves the correctness and performance of incremental and cached clean builds. To learn more, read Pass arguments to annotation processors (/studio/build/dependencies#processor-arguments).
- **Migration tool for AndroidX:** When using Android Gradle plugin 3.2.0 with Android 3.2 and higher, you can migrate your project's local and Maven dependencies to use the new AndroidX libraries by selecting **Refactor > Migrate to AndroidX** from the menu bar. Using this migration tool also sets the following flags to `true` in your `gradle.properties` file:

    - **`android.useAndroidX`:** When set to `true`, the Android plugin uses the appropriate AndroidX library instead of a Support Library. When this flag is not specified, the plugin sets it to `false` by default.
    - **`android.enableJetifier`:** When set to `true`, the Android plugin automatically migrates existing third-party libraries to use AndroidX by rewriting their binaries. When this flag is not specified, the plugin sets it to `false` by default. You can set this flag to `true` only while `android.useAndroidX` is also set to `true`, otherwise you get a build error.

  To learn more, read the AndroidX overview (/topic/libraries/support-library/androidx-overview).

- **New code shrinker, R8:** R8 is a new tool for code shrinking and obfuscation that replaces ProGuard. You can start using the preview version of R8 by including the following in your project's `gradle.properties` file:

  Groovy (#groovy)Kotlin (#kotlin)

  ```
  android.enableR8 = true
  ```

Behavior changes

- Desugaring with D8 is now enabled by default.
- AAPT2 is now on Google's Maven repo. To use AAPT2, make sure that you have the `google()` dependency in your `build.gradle` file, as shown below:

  Groovy (#groovy)Kotlin (#kotlin)

  ```
  buildscript {
      repositories {
          google() // here
          jcenter()
      }
      dependencies {
          classpath 'com.android.tools.build:gradle:3.2.0'
      }
  ```

```
}
allprojects {
    repositories {
        google() // and here
        jcenter()
}
```

- Native multidex is now enabled by default. Previous versions of Android Studio enabled native multidex when deploying the debug version of an app to a device running Android API level 21 or higher. Now, whether you're deploying to a device or building an APK for release, the Android Gradle plugin enables native multidex for all modules that set `minSdkVersion=21` or higher.
- The plugin now enforces a minimum version of the protobuf plugin (0.8.6), Kotlin plugin (1.2.50), and Crashlytics plugin (1.25.4).
- The feature module plugin,`com.android.feature`, now enforces the use of only letters, digits, and underscores when specifying a module name. For example, if your feature module name includes dashes, you get a build error. This behavior matches that of the dynamic feature plugin.

Bug fixes

- JavaCompile is now cacheable in projects with data binding. (Issue #69243050 (https://issuetracker.google.com/69243050))
- Better compile avoidance for library modules with data binding. (Issue #77539932 (https://issuetracker.google.com/77539932))
- You can now re-enable configure-on-demand (https://docs.gradle.org/current/userguide/multi_project_builds.html#sec:configuration_on_demand) if you've disable it in earlier versions due to some unpredictable build errors. (Issue #77910727 (https://issuetracker.google.com/77910727))

3.1.0 (March 2018)

This version of the Android plugin requires the following:

- Gradle 4.4 (https://docs.gradle.org/current/release-notes.html) or higher. To learn more, read the section about updating Gradle (#updating-gradle).

- Build Tools 27.0.3 (/studio/releases/build-tools#notes) or higher. Keep in mind, you no longer need to specify a version for the build tools using the `android.buildToolsVersion` property—the plugin uses the minimum required version by default.

New DEX compiler, D8

By default, Android Studio now uses a new DEX compiler called D8. DEX compilation is the process of transforming `.class` bytecode into `.dex` bytecode for the Android Runtime (or Dalvik, for older versions of Android). Compared to the previous compiler, called DX, D8 compiles faster and outputs smaller DEX files, all while having the same or better app runtime performance.

D8 shouldn't change your day-to-day app development workflow. However, if you experience any issues related to the new compiler, please report a bug (/studio/report-bugs). You can temporarily disable D8 and use DX by including the following in your project's `gradle.properties` file:

```
android.enableD8=false
```

For projects that use Java 8 language features (/studio/write/java8-support), incremental desugaring is enabled by default. You can disable it by specifying the following in your project's `gradle.properties` file:

```
android.enableIncrementalDesugaring=false.
```

Preview users: If you're already using a preview version of D8, note that it now compiles against libraries included in the SDK build tools (/studio/releases/build-tools)—not the JDK. So, if you are accessing APIs that exist in the JDK but not in the SDK build tools libraries, you get a compile error.

Behavior changes

- When building multiple APKs that each target a different ABI, the plugin no longer generates APKs for the following ABIs by default: `mips`, `mips64`, and `armeabi`.

If you want to build APKs that target these ABIs, you must use NDK r16b or lower (/ndk/downloads/revision_history) and specify the ABIs in your `build.gradle` file, as shown below:

Groovy (#groovy)Kotlin (#kotlin)

```
splits {
    abi {
        include("armeabi", "mips", "mips64")
        ...
    }
}
```

- The Android plugin's build cache (/studio/build/build-cache) now evicts cache entries that are older than 30 days.
- Passing `"auto"` to `resConfig` (https://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.ProductFlavor.html#com.android.build.gradle.internal.dsl.ProductFlavor:resConfig(java.lang.String)) no longer automatically picks string resources to package into your APK. If you continue to use `"auto"`, the plugin packages all string resources your app and its dependencies provide. So, you should instead specify each locale that you want the plugin to package into your APK.
- Because local modules can't depend on your app's test APK, adding dependencies to your instrumented tests using the `androidTestApi` configuration, instead of `androidTestImplementation`, causes Gradle to issue the following warning:

Groovy (#groovy)Kotlin (#kotlin)

```
WARNING: Configuration 'androidTestApi' is obsolete
  and has been replaced with 'androidTestImplementation'
```

Fixes

- Fixes an issue where Android Studio doesn't properly recognize dependencies in composite builds.
- Fixes an issue where you get a project sync error when loading the Android plugin multiple times in a single build–for example, when multiple subprojects each include the Android plugin in their buildscript classpath.

3.0.0 (October 2017)

Android Gradle plugin 3.0.0 includes a variety of changes that aim to address performance issues of large projects.

For example, on a sample skeleton project (https://github.com/jmslau/perf-android-large.git) with ~130 modules and a large number of external dependencies (but no code or resources), you can experience performance improvements similar to the following:

Android plugin version + Gradle version	Android plugin 2.2.0 + Gradle 2.14.1	Android plugin 2.3.0 + Gradle 3.3	Android plugin 3.0.0 + Gradle 4.1
Configuration (e.g. running `./gradlew --help`)	~2 mins	~9 s	~2.5 s
1-line Java change (implementation change)	~2 mins 15 s	~29 s	~6.4 s

Some of these changes break existing builds. So, you should consider the effort of migrating your project before using the new plugin.

If you don't experience the performance improvements described above, please file a bug (https://issuetracker.google.com/issues/new?component=192708&template=840533) and include a trace of your build using the Gradle Profiler (https://github.com/gradle/gradle-profiler).

This version of the Android plugin requires the following:

- Gradle 4.1 (https://docs.gradle.org/current/release-notes.html) or higher. To learn more, read the section about updating Gradle (#updating-gradle).

- Build Tools 26.0.2 (/studio/releases/build-tools#notes) or higher. With this update, you no longer need to specify a version for the build tools—the plugin uses the minimum required version by default. So, you can now remove the `android.buildToolsVersion` property.

3.0.1 (November 2017)

This is a minor update to support Android Studio 3.0.1, and includes general bug fixes and performance improvements.

Optimizations

- Better parallelism for multi-module projects through a fine grained task graph.
- When making changes to dependency, Gradle performs faster builds by not re-compiling modules that do not have access to that dependency's API. You should restrict which dependencies leak their APIs to other modules by using Gradle's new dependency configurations (/studio/build/dependencies#dependency_configurations): `implementation`, `api`, `compileOnly`, and `runtimeOnly`.
- Faster incremental build speed due to per-class dexing. Each class is now compiled into separate DEX files, and only the classes that are modified are re-dexed. You should also expect improved build speeds for apps that set `minSdkVersion` to 20 or lower, and use legacy multi-dex (/studio/build/multidex#mdex-pre-l).
- Improved build speeds by optimizing certain tasks to use chached outputs. To benefit from this optimization, you need to first enable the Gradle build cache (https://docs.gradle.org/current/userguide/build_cache.html#sec:build_cache_enable).
- Improved incremental resource processing using AAPT2, which is now enabled by default. If you are experiencing issues while using AAPT2, please report a bug (/studio/report-bugs). You can also disable AAPT2 by setting `android.enableAapt2=false` in your `gradle.properties` file and restarting the Gradle daemon by running `./gradlew --stop` from the command line.

New features

- Variant-aware dependency management (/studio/build/build-variants#variant_aware). When building a certain variant of a module, the plugin now automatically matches variants of local library module dependencies to the variant of the module you are building.

- Includes a new Feature module plugin to support Android Instant Apps (/topic/instant-apps) and the Android Instant Apps SDK (which you can download using the SDK manager (/studio/intro/update#sdk-manager)). To learn more about creating Feature modules with the new plugin, read Structure of an instant app with multiple features (/topic/instant-apps/getting-started/structure#structure_of_an_instant_app_with_multiple_features).
- Built-in support for using certain Java 8 language features and Java 8 libraries. **Jack is now deprecated and no longer required**, and you should first disable Jack to use the improved Java 8 support built into the default toolchain. For more information, read Use Java 8 language features (/studio/write/java8-support).
- Added support for running tests with Android Test Orchestrator (/training/testing/junit-runner#using-android-test-orchestrator), which allows you to run each of your app's tests within its own invocation of `Instrumentation` (/reference/android/app/Instrumentation). Because each test runs in its own `Instrumentation` (/reference/android/app/Instrumentation) instance, any shared state between tests doesn't accumulate on your device's CPU or memory. And, even if one test crashes, it takes down only its own instance of `Instrumentation` (/reference/android/app/Instrumentation), so your other tests still run.
  - Added `testOptions.execution` to determine whether to use on-device test orchestration. If you want to use Android Test Orchestrator (/training/testing/junit-runner#using-android-test-orchestrator), you need to specify `ANDROID_TEST_ORCHESTRATOR`, as shown below. By default, this property is set to `HOST`, which disables on-device orchestration and is the standard method of running tests.

Groovy (#groovy)Kotlin (#kotlin)

```
android {
  testOptions {
    execution = "ANDROID_TEST_ORCHESTRATOR"
  }
}
```

- New `androidTestUtil` dependency configuration allows you to install another test helper APK before running your instrumentation tests, such as Android Test

Orchestrator:

Groovy (#groovy)Kotlin (#kotlin)

```
dependencies {
  androidTestUtil("com.android.support.test:orchestrator:1.0.0")
  ...
}
```

- Added `testOptions.unitTests.includeAndroidResources` to support unit tests that require Android resources, such as Roboelectric (http://robolectric.org/). When you set this property to `true`, the plugin performs resource, asset, and manifest merging before running your unit tests. Your tests can then inspect `com/android/tools/test_config.properties` on the classpath for the following keys:
  - `android_merged_assets`: the absolute path to the merged assets directory.

    ★ **Note:** For library modules, the merged assets do not contain the assets of dependencies (see issue #65550419 (https://issuetracker.google.com/65550419)).

  - `android_merged_manifest`: the absolute path to the merged manifest file.
  - `android_merged_resources`: the absolute path to the merged resources directory, which contains all the resources from the module and all its dependencies.
  - `android_custom_package`: the package name of the final R class. If you dynamically modify the application ID, this package name may not match the `package` attribute in the app's manifest.
- Support for fonts as resources (/guide/topics/ui/look-and-feel/fonts-in-xml) (which is a new feature introduced in Android 8.0 (API level 26) (/about/versions/oreo)).
- Support for language-specific APKs with Android Instant Apps SDK 1.1 (/topic/instant-apps/release-notes#android_instant_apps_development_sdk_v110) and higher.
- You can now change the output directory for your external native build project, as shown below:

(#kotlin)

```
android {
    ...
    externalNativeBuild {
        // For ndk-build, instead use the ndkBuild block.
        cmake {
            ...
            // Specifies a relative path for outputs from external native
            // builds. You can specify any path that's not a subdirectory
            // of your project's temporary build/ directory.
            buildStagingDirectory = "./outputs/cmake"
        }
    }
}
```

- You can now use CMake 3.7 or higher (/studio/projects/install-ndk#vanilla_cmake) when building native projects from Android Studio.
- New `lintChecks` dependency configuration allows you to build a JAR that defines custom lint rules, and package it into your AAR and APK projects. Your custom lint rules must belong to a separate project that outputs a single JAR and includes only `compileOnly` (https://docs.gradle.org/current/userguide/java_plugin.html#sec:java_plugin_and_dependency_management) dependencies. Other app and library modules can then depend on your lint project using the `lintChecks` configuration:

Groovy (#groovy)Kotlin (#kotlin)

```
dependencies {
    // This tells the Gradle plugin to build ':lint-checks' into a lint.jar
    // and package it with your module. If the module is an Android library
    // other projects that depend on it automatically use the lint checks.
    // If the module is an app, lint includes these rules when analyzing th
    lintChecks(project(":lint-checks"))
}
```

Behavior changes

- Android plugin 3.0.0 removes certain APIs, and your build will break if you use them. For example, you can no longer use the Variants API to access `outputFile()` objects or use `processManifest.manifestOutputFile()` to get the manifest file for each variant. To learn more, read API changes (/studio/known-issues#variant_api).
- You no longer need to specify a version for the build tools (so, you can now remove the `android.buildToolsVersion` property). By default, the plugin automatically uses the minimum required build tools version for the version of Android plugin you're using.
- You now enable/disable PNG crunching in the `buildTypes` block, as shown below. PNG crunching is enabled by default for all builds except debug builds because it increases build times for projects that include many PNG files. So, to improve build times for other build types, you should either disable PNG crunching or convert your images to WebP (/studio/write/convert-webp#convert_images_to_webp).

Groovy (#groovy)Kotlin (#kotlin)

```
android {
  buildTypes {
    release {
      // Disables PNG crunching for the release build type.
      isCrunchPngs = false
    }
  }
}
```

- The Android plugin now automatically builds executable targets that you configure in your external CMake projects.
- You must now add annotation processors (/studio/build/dependencies#annotation_processor) to the processor classpath using the `annotationProcessor` dependency configuration.
- Using the deprecated `ndkCompile` is now more restricted. You should instead migrate to using either CMake or ndk-build to compile native code that you want to package into your APK. To learn more, read Migrate from ndkcompile (/studio/projects/add-native-code#ndkCompile).

2.3.0 (February 2017)

2.3.3 (June 2017)

This is a minor update that adds compatibility with Android Studio 2.3.3 (/studio/releases#Revisions).

2.3.2 (May 2017)

This is a minor update that adds compatibility with Android Studio 2.3.2 (/studio/releases#Revisions).

2.3.1 (April 2017)

This is a minor update to Android plugin 2.3.0 that fixes an issue where some physical Android devices did not work properly with Instant Run (/studio/run#instant-run) (see Issue #235879 (https://code.google.com/p/android/issues/detail?id=235879)).

Dependencies:

- Gradle 3.3 or higher.
- Build Tools 25.0.0 (/tools/revisions/build-tools) or higher.

New:

- Uses Gradle 3.3, which includes performance improvements and new features. For more details, see the Gradle release notes (https://docs.gradle.org/3.3/release-notes).
- **Build cache**: stores certain outputs that the Android plugin generates when building your project (such as unpackaged AARs and pre-dexed remote dependencies). Your clean builds are much faster while using the cache because the build system can simply reuse those cached files during subsequent builds, instead of recreating them. Projects using Android plugin 2.3.0 and higher use the build cache by default. To learn more, read Improve Build Speed with Build Cache (/studio/build/build-cache).
    - Includes a `cleanBuildCache` task that clears the build cache (/studio/build/build-cache#clear_the_build_cache).

    - If you are using the experimental version of build cache (included in earlier versions of the plugin), you should update your plugin (#updating-plugin) to the latest version.

Changes:

- Supports changes to Instant Run included in Android Studio 2.3 (/studio/releases).
- Configuration times for very large projects should be significantly faster.
- Fixed issues with auto-downloading for the constraint layout library (/training/constraint-layout).
- Plugin now uses ProGuard version 5.3.2 (https://www.guardsquare.com/en/proguard/manual/versions).
- Includes many fixes for reported bugs (https://code.google.com/p/android/issues/list?can=1&q=Component%3DTools++Subcomponent%3DTools-gradle%2CTools-build%2CTools-instantrun%2CTools-cpp-build+Target%3D2.3+status%3AFutureRelease%2CReleased+&sort=priority+-status&colspec=ID+Status+Priority+Owner+Summary+Stars+Reporter+Opened&cells=tiles) . Please continue to file bug reports (/studio/report-bugs) when you encounter issues.

2.2.0 (September 2016)

Dependencies:

- Gradle 2.14.1 or higher.
- Build Tools 23.0.2 (/tools/revisions/build-tools) or higher.

New:

- Uses Gradle 2.14.1, which includes performance improvements and new features, and fixes a security vulnerability that allows local privilege escalation when using the Gradle daemon. For more details, see the Gradle release notes (https://docs.gradle.org/2.14.1/release-notes).
- Using the `externalNativeBuild {}` (https://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.ExternalNativeBuild.html) DSL, Gradle now lets you link to your native sources and compile native libraries using CMake or ndk-build. After building your native libraries, Gradle packages

them into your APK. To learn more about using CMake and ndk-build with Gradle, read Add C and C++ Code to Your Project (/studio/projects/add-native-code).

- When you run a build from the command line (/studio/build/building-cmdline), Gradle now attempts to auto-download any missing SDK components or updates that your project depends on. To learn more, read Auto-download missing packages with Gradle (/studio/intro/update#download-with-gradle).
- A new experimental caching feature lets Gradle speed up build times by pre-dexing, storing, and reusing the pre-dexed versions of your libraries. To learn more about using this experimental feature, read the Build Cache (/studio/build/build-cache) guide.
- Improves build performance by adopting a new default packaging pipeline which handles zipping, signing, and zipaligning (/studio/command-line/zipalign) in one task. You can revert to using the older packaging tools by adding `android.useOldPackaging=true` to your `gradle.properties` file. While using the new packaging tool, the `zipalignDebug` task is not available. However, you can create one yourself by calling the `createZipAlignTask(String taskName, File inputFile, File outputFile)` method.
- APK signing now uses APK Signature Scheme v2 (/about/versions/nougat/android-7.0#apk_signature_v2) in addition to traditional JAR signing. All Android platforms accept the resulting APKs. Any modification to these APKs after signing invalidates their v2 signatures and prevents installation on a device. To disable this feature, add the following to your module-level `build.gradle` file:

Groovy (#groovy)Kotlin (#kotlin)

```
android {
  ...
  signingConfigs {
    create("config") {
      ...
      v2SigningEnabled = false
    }
  }
}
```

- For multidex builds, you can now use ProGuard rules to determine which classes Gradle should compile into your app's *main* DEX file. Because the Android system loads the main DEX file first when starting your app, you can prioritize certain classes at startup by compiling them into the main DEX file. After you create a ProGuard configuration file specifically for your main DEX file, pass the configuration file's path to Gradle using `buildTypes.multiDexKeepProguard` (https://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.BuildType.html#com.android.build.gradle.internal.dsl.BuildType:multiDexKeepProguard) . Using this DSL is different from using `buildTypes.proguardFiles` (https://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.BuildType.html#com.android.build.gradle.internal.dsl.BuildType:proguardFiles(java.lang.Object[])) , which provides general ProGuard rules for your app and does not specify classes for the main DEX file.
- Adds support for the `android:extractNativeLibs` flag, which can reduce the size of your app when you install it on a device. When you set this flag to `false` in the `<application>` (/guide/topics/manifest/application-element) element of your app manifest, Gradle packages uncompressed and aligned versions of your native libraries with your APK. This prevents `PackageManager` (/reference/android/content/pm/PackageManager) from copying out your native libraries from the APK to the device's file system during installation and has the added benefit of making delta updates of your app smaller.
- You can now specify `versionNameSuffix` (https://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.ProductFlavor.html#com.android.build.gradle.internal.dsl.ProductFlavor:versionNameSuffix) and `applicationIdSuffix` (https://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.ProductFlavor.html#com.android.build.gradle.internal.dsl.ProductFlavor:applicationIdSuffix) for product flavors. (Issue 59614 (http://b.android.com/59614))

Changes:

- `getDefaultProguardFile` now returns the default ProGuard files that Android plugin for Gradle provides and no longer uses the ones in the Android SDK.
- Improved Jack compiler performance and features:
    - Jack now supports Jacoco test coverage when setting `testCoverageEnabled`

(https://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.BuildType.html#com.android.build.gradle.internal.dsl.BuildType:testCoverageEnabled)
to `true`.

- Improved support for annotation processors. Annotation processors on your classpath, such as any `compile` dependencies, are automatically applied to your build. You can also specify an annotation processor in your build and pass arguments by using the `javaCompileOptions.annotationProcessorOptions {}` (https://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.AnnotationProcessorOptions.html) DSL in your module-level `build.gradle` file:

  Groovy (#groovy)Kotlin (#kotlin)

  ```
  android {
    ...
    defaultConfig {
      ...
      javaCompileOptions {
        annotationProcessorOptions {
          className = "com.example.MyProcessor"
          // Arguments are optional.
          arguments(mapOf(foo to "bar"))
        }
      }
    }
  }
  ```

  If you want to apply an annotation processor at compile time but not include it in your APK, use the `annotationProcessor` dependency scope:

  Groovy (#groovy)Kotlin (#kotlin)

  ```
  dependencies {
      implementation("com.google.dagger:dagger:2.0")
      annotationProcessor("com.google.dagger:dagger-compiler:2
     // or use buildVariantAnnotationProcessor to target a spe
  }
  ```

For a list of parameters you can set, run the following from the command line:

```
$ java -jar /build-tools/jack.jar --help-properties
```

- By default, if the Gradle daemon's heap size is at least 1.5 GB, Jack now runs in the same process as Gradle. To adjust the daemon heap size, add the following to your `gradle.properties` file:

```
# This sets the daemon heap size to 1.5GB.
org.gradle.jvmargs=-Xmx1536M
```

2.1.0 (April 2016)

2.1.3 (August 2016)

This update requires Gradle 2.14.1 and higher. Gradle 2.14.1 includes performance improvements, new features, and an important security fix (https://docs.gradle.org/2.14/release-notes#local-privilege-escalation-when-using-the-daemon). For more details, see the Gradle release notes (https://docs.gradle.org/2.14.1/release-notes).

Dependencies:

- Gradle 2.10 or higher.
- Build Tools 23.0.2 (/tools/revisions/build-tools) or higher.

New:

- Added support for the N Developer Preview, JDK 8, and Java 8 language features (/preview/j8-jack) using the Jack toolchain. To find out more, read the N Preview guide (/about/versions/nougat).

★ **Note:** Instant Run (/tools/building/building-studio#instant-run) does not currently work with Jack and will be disabled while using the new toolchain. You only need to use Jack if you are developing for the N Preview and want to use the supported Java 8 language features.

- Added default support for incremental Java compilation to reduce compilation time during development. It does this by only recompiling portions of the source that have changed or need to be recompiled. To disable this feature, add the following code to your module-level `build.gradle` file:

  Groovy (#groovy)Kotlin (#kotlin)

  ```
  android {
    ...
    compileOptions {
      incremental = false
    }
  }
  ```

- Added support for dexing-in-process which performs dexing within the build process rather than in a separate, external VM processes. This not only makes incremental builds faster, but also speeds up full builds. The feature is enabled by default for projects that have set the Gradle daemon's maximum heap size to at least 2048 MB. You can do this by including the following in your project's `gradle.properties` file:

  ```none org.gradle.jvmargs = -Xmx2048m ```

  If you have defined a value for `javaMaxHeapSize` (https://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.DexOptions.html#com.android.build.gradle.internal.dsl.DexOptions:javaMaxHeapSize) in your module-level `build.gradle` file, you need to set `org.gradle.jvmargs` to the value of `javaMaxHeapSize` + 1024 MB. For example, if you have set `javaMaxHeapSize` to "2048m", you need to add the following to your project's `gradle.properties` file:

  ```none org.gradle.jvmargs = -Xmx3072m ```

  To disable dexing-in-process, add the following code to your module-level `build.gradle` file:

Groovy (#groovy)Kotlin (#kotlin)

```
android {
  ...
  dexOptions {
      dexInProcess = false
  }
}
```

2.0.0 (April 2016)

Dependencies:

- Gradle 2.10 or higher.
- Build Tools 21.1.1 (/tools/revisions/build-tools) or higher.

New:

- Enables Instant Run (/tools/building/building-studio#instant-run) by supporting bytecode injection, and pushing code and resource updates to a running app on the emulator or a physical device.
- Added support for incremental builds, even when the app isn't running. Full build times are improved by pushing incremental changes through the Android Debug Bridge (/tools/help/adb) to the connected device.
- Added `maxProcessCount` (https://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.DexOptions.html#com.android.build.gradle.internal.dsl.DexOptions:maxProcessCount) to control how many worker dex processes can be spawned concurrently. The following code, in the module-level `build.gradle` file, sets the maximum number of concurrent processes to 4:

Groovy (#groovy)Kotlin (#kotlin)

```
android {
  ...
  dexOptions {
    maxProcessCount = 4 // this is the default value
  }
}
```

- Added an experimental code shrinker to support pre-dexing and reduce re-dexing of dependencies, which are not supported with Proguard. This improves the build speed of your debug build variant. Because the experimental shrinker does not support optimization and obfuscation, you should enable Proguard for your release builds. To enable the experimental shrinker for your debug builds, add the following to your module-level `build.gradle` file:

Groovy (#groovy)Kotlin (#kotlin)

```
android {
  ...
  buildTypes {
    getByName("debug") {
      minifyEnabled = true
      useProguard = false
    }
    getByName("release") {
      minifyEnabled = true
      useProguard = true // this is a default setting
    }
  }
}
```

- Added logging support and improved performance for the resource shrinker. The resource shrinker now logs all of its operations into a `resources.txt` file located in the same folder as the Proguard log files.

Changed behavior:

- When `minSdkVersion` is set to 18 or higher, APK signing uses SHA256.
- DSA and ECDSA keys can now sign APK packages.

★ **Note:** The Android keystore (/training/articles/keystore) provider no longer supports DSA keys on Android 6.0

(/about/versions/marshmallow/android-6.0-changes#behavior-keystore) (API level 23) and higher.

Fixed issues:

- Fixed an issue that caused duplicate AAR dependencies in both the test and main build configurations.

Older releases

Android plugin for Gradle, revision 1.5.0 *(November 2015)*

Dependencies:

- Gradle 2.2.1 or higher.
- Build Tools 21.1.1 or higher.

General Notes:

- Integrated the Data Binding plugin into the Android plugin for Gradle. To enable it, add the following code to each per-project `build.gradle` file that uses the plugin:

  Groovy (#groovy)Kotlin (#kotlin)

  ```
  android {
      dataBinding {
          enabled = true
      }
  }
  ```

- Added a new Transform API (https://google.github.io/android-gradle-dsl/javadoc/1.5/) to allow third-party plugins to manipulate compiled `.class` files before they're converted to `.dex` files. The Transform API simplifies injecting custom class manipulations while offering more flexibility regarding what you can manipulate. To insert a transform into a build, create a new class implementing one of the `Transform` interfaces, and register it with `android.registerTransform(theTransform)` or `android.registerTransform(theTransform, dependencies)`. There's no need to wire tasks together. Note the following about the Transform API:

    - A transform can apply to one or more of the following: the current project, subprojects, and external libraries.
    - A transform must be registered globally, which applies them to all variants.
    - Internal code processing, through the Java Code Coverage Library (JaCoCo), ProGuard, and MultiDex, now uses the Transform API. However, the Java Android Compiler Kit (Jack) doesn't use this API: only the `javac/dx` code path does.
    - Gradle executes the transforms in this order: JaCoCo, third-party plugins, ProGuard. The execution order for third-party plugins matches the order in which the transforms are added by the third party plugins; third-party plugin developers can't control the execution order of the transforms through an API.

- Deprecated the `dex` getter from the `ApplicationVariant` class. You can't access the `Dex` task through the variant API anymore because it's now accomplished through a transform. There's currently no replacement for controlling the dex process.
- Fixed incremental support for assets.
- Improved MultiDex support by making it available for test projects, and tests now automatically have the `com.android.support:multidex-instrumentation` dependency.
- Added the ability to properly fail a Gradle build and report the underlying error cause when the Gradle build invokes asynchronous tasks and there's a failure in the worker process.
- Added support for configuring a specific Application Binary Interface (ABI) in variants that contain multiple ABIs.
- Added support for a comma-separated list of device serial numbers for the `ANDROID_SERIAL` environment variable when installing or running tests.
- Fixed an installation failure on devices running Android 5.0 (API level 20) and higher when the APK name contains a space.
- Fixed various issues related to the Android Asset Packaging Tool (AAPT) error output.

- Added JaCoCo incremental instrumentation support for faster incremental builds. The Android plugin for Gradle now invokes the JaCoCo instrumenter directly. To force a newer version of the JaCoCo instrumenter, you need to add it as a build script dependency.
- Fixed JaCoCo support so it ignores files that aren't classes.
- Added vector drawable support for generating PNGs at build time for backward-compatibility. Android plugin for Gradle generates PNGs for every vector drawable found in a resource directory that doesn't specify an API version or specifies an `android:minSdkVersion` attribute of 20 or lower in the `<uses-sdk>` element in the app manifest. You can set PNG densities by using the `generatedDensities` property in the `defaultConfig` or `productFlavor` sections of a `build.gradle` file.
- Added sharing of the mockable `android.jar`, which the plugin generates only once and uses for unit testing. Multiple modules, such as `app` and `lib`, now share it. Delete `$rootDir/build` to regenerate it.
- Changed the processing of Java resources to occur before the obfuscation tasks instead of during the packaging of the APK. This change allows the obfuscation tasks to have a chance to adapt the Java resources following packages obfuscation.
- Fixed an issue with using Java Native Interface (JNI) code in the experimental library plugin.
- Added the ability to set the platform version separately from the `android:compileSdkVersion` attribute in the experimental library plugin.

Android plugin for Gradle, revision 1.3.1 *(August 2015)*

Dependencies:

- Gradle 2.2.1 or higher.
- Build Tools 21.1.1 or higher.

General Notes:

- Fixed the ZipAlign (/tools/help/zipalign) task to properly consume the output of the previous task when using a customized filename.
- Fixed Renderscript (/guide/topics/renderscript/compute) packaging with the NDK (/tools/sdk/ndk).

- Maintained support for the `createDebugCoverageReport` build task.
- Fixed support for customized use of the `archiveBaseName` property in the `build.gradle` build> file.
- Fixed the `Invalid ResourceType` lint (/tools/help/lint) warning caused by parameter method annotation lookup when running lint (/tools/help/lint) outside of Android Studio.

Android plugin for Gradle, revision 1.3.0 *(July 2015)*

Dependencies:

- Gradle 2.2.1 or higher.
- Build Tools 21.1.1 or higher.

General Notes:

- Added support for the `com.android.build.threadPoolSize` property to control the `Android` task thread pool size from the `gradle.properties` file or the command line. The following example sets this property to 4. ```none -Pcom.android.build.threadPoolSize=4 ```
- Set the default build behavior to exclude `LICENSE` and `LICENSE.txt` files from APKs. To include these files in an APK, remove these files from the `packagingOptions.excludes` property in the `build.gradle` file. For example:

  Groovy (#groovy)Kotlin (#kotlin)

  ```
  android {
        packagingOptions.excludes.clear()
      }
  ```

- Added the `sourceSets` task to inspect the set of all available source sets.
- Enhanced unit test support to recognize multi-flavor and build variant (/tools/building/configuring-gradle#workBuildVariants) source folders. For example, to test an app with multi-flavors `flavor1` and `flavorA` with the `Debug` build type, the test source sets are:
  - test

    - testFlavor1
    - testFlavorA
    - testFlavor1FlavorA
    - testFlavor1FlavorADebug

  Android tests already recognized multi-flavor source folders.

- Improved unit test support to:
    - Run `javac` on main and test sources, even if the `useJack` property is set to `true` in your build file.
    - Correctly recognize dependencies for each build type.
- Added support for specifying instrumentation test-runner arguments from the command line. For example:

```
./gradlew connectedCheck \
   -Pandroid.testInstrumentationRunnerArguments.size=medium \
   -Pandroid.testInstrumentationRunnerArguments.class=TestA,TestB
```

- Added support for arbitrary additional Android Asset Packaging Tool (AAPT) parameters in the `build.gradle` file. For example:

Groovy (#groovy)Kotlin (#kotlin)

```
android {
    aaptOptions {
      additionalParameters += listOf("--custom_option", "value")
    }
}
```

- Added support for a test APK module (/tools/studio/studio-features#test-module) as a separate test module, using the `targetProjectPath` and `targetVariant` properties to set the APK path and target variant.

★ **Note:** A test APK module does not support product flavors and can only target a single variant. Also, Jacoco is not supported yet.

- Added resource name validation before merging resources.
- When building an AAR (Android ARchive) package for library modules, do not provide an automatic `@{applicationId}` placeholder in the manifest merger (/tools/building/manifest-merge) settings. Instead, use a different placeholder, such as `@{libApplicationId}` and provide a value for it if you want to include application Ids in the archive library.

Android plugin for Gradle, revision 1.2.0 *(April 2015)*

Dependencies:

- Gradle 2.2.1 or higher.
- Build Tools 21.1.1 or higher.

General Notes:

- Enhanced support for running unit tests with Gradle.
  - Added support to include Java-style resources in the classpath when running unit tests directly from Gradle.
  - Added unit test dependency support for Android Archive (AAR) artifacts.
  - Added support for the `unitTestVariants` property so unit test variants can be manipulated using the `build.gradle` file.
  - Added the `unitTest.all` code block under `testOptions` to configure customized tasks for unit test. The following sample code shows how to add unit test configuration settings using this new option:

Groovy (#groovy)Kotlin (#kotlin)

```
android {
  testOptions {
    unitTest.all {
      jvmArgs += listOf("-XX:MaxPermSize=256m") // Or any ot
    }
  }
}
```

- Fixed the handling of enums and public instance fields in the packaging of the `mockable-android.jar` file.
- Fixed library project task dependencies so test classes recompile after changes.

- Added the `testProguardFile` property to apply ProGuard (/tools/help/proguard) files when minifying a test APK.
- Added the `timeOut` property to the `adbOptions` code block for setting the maximum recording time for Android Debug Bridge (/tools/help/adb) screen recording.
- Added support for 280 dpi resources.
- Improved performance during project evaluation.

Android plugin for Gradle, revision 1.1.3 *(March 2015)*

Dependencies:

- Gradle 2.2.1 or higher.
- Build Tools 21.1.1 or higher.

General Notes:

- Fixed issue with duplicated dependencies on a test app that triggered a ProGuard failure.
- Fixed Comparator implementation which did not comply with the JDK Comparator contract and generated a JDK 7 error.

Android plugin for Gradle, revision 1.1.2 *(February 2015)*

Dependencies:

- Gradle 2.2.1 or higher.
- Build Tools 21.1.1 or higher.

General Notes:

- Normalized path when creating a mockable JAR for unit testing.
- Fixed the `archivesBaseName` setting in the `build.gradle` file.

- Fixed the unresolved placeholder failure in manifest merger when building a library test application.

Android plugin for Gradle, revision 1.1.1 *(February 2015)*

Dependencies:

- Gradle 2.2.1 or higher.
- Build Tools 21.1.1 or higher.

General Notes:

- Modified build variants so only variants that package a Wear (/training/wearables/apps) app trigger Wear-specific build tasks.
- Changed dependency related issues to fail at build time rather than at debug time. This behavior allows you to run diagnostic tasks (such as 'dependencies') to help resolve the conflict.
- Fixed the `android.getBootClasspath()` method to return a value.

Android plugin for Gradle, revision 1.1.0 *(February 2015)*

Dependencies:

- Gradle 2.2.1 or higher.
- Build Tools 21.1.1 or higher.

General Notes:

- Added new unit test support
  - Enabled unit tests (/training/activity-testing/activity-unit-testing) to run on the local JVM against a special version of the `android.jar` file that is compatible with popular mocking frameworks, for example Mockito.
  - Added new test tasks `testDebug`, `testRelease`, and `testMyFlavorDebug` when using product flavors.
  - Added new source folders recognized as unit tests: `src/test/java/`, `src/testDebug/java/`, `src/testMyFlavor/java/`.
  - Added new configurations in the `build.gradle` file for declaring test-only dependencies, for example, `testCompile 'junit:junit:4.11'`, `testMyFlavorCompile 'some:library:1.0'`.

> ★ **Note:** Test-only dependencies are not currently compatible with Jack (Java Android Compiler Kit).

    - Added the `android.testOptions.unitTests.returnDefaultValues` option to control the behaviour of the mockable android.jar.

- Replaced `Test` in test task names with `AndroidTest`. For example, the `assembleDebugTest` task is now `assembleDebugAndroidTest` task. Unit test tasks still have `UnitTest` in the task name, for example `assembleDebugUnitTest`.
- Modified ProGuard (/tools/help/proguard) configuration files to no longer apply to the test APK. If minification is enabled, ProGuard processes the test APK and applies only the mapping file that is generated when minifying the main APK.
- Updated dependency management
    - Fixed issues using `provided` and `package` scopes.

> ★ **Note:** These scopes are incompatible with AAR (Android ARchive) packages and will cause a build with AAR packages to fail.

    - Modified dependency resolution to compare the dependencies of an app under test and the test app. If an artifact with the same version is found for both apps, it's not included with the test app and is packaged only with the app under test. If an artifact with a different version is found for both apps, the build fails.

- Added support for `anyDpi` resource qualifier (/guide/topics/resources/providing-resources) in resource merger.
- Improved evaluation and IDE sync speeds for projects with a large number of Android modules (/studio/projects).

Android plugin for Gradle, revision 1.0.1 *(January 2015)*

Dependencies:

- Gradle 2.2.1 up to 2.3.x.

> ★ **Note:** This version of the Android plugin for Gradle is not compatible with Gradle 2.4 and higher.

- Build Tools 21.1.1 or higher.

General Notes:

- Fixed issue with Gradle build failure when accessing the `extractReleaseAnnotations` module. (Issue 81638 (http://b.android.com/81638)).
- Fixed issue with `Disable` passing the `--no-optimize` setting to the Dalvik Executable (dex) bytecode. (Issue 82662 (http://b.android.com/82662)).
- Fixed manifest merger issues when importing libraries with a `targetSdkVersion` less than 16.
- Fixed density ordering issue when using Android Studio with JDK 8.

Android plugin for Gradle, revision 1.0.0 *(December 2014)*

Dependencies:

- Gradle 2.2.1 up to 2.3.x.

★ **Note:** This version of the Android plugin for Gradle is not compatible with Gradle 2.4 and higher.

- Build Tools 21.1.1 or higher.

General Notes:

- Initial plugin release.

Previous
← SDK build tools (/studio/releases/build-tools)

Next
Android Gradle plugin roadmap (/studio/releases/gradle-plugin-roadmap) →