Android Developers

# Android Plugin for Gradle Release Notes

The Android Studio build system is based on Gradle, and the Android plugin for Gradle 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 (#revisions).

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

- Configure Your Build (https://developer.android.google.cn/studio/build/index.html)
- Android Plugin DSL Reference (http://google.github.io/android-gradle-dsl/current/)
- 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) .

In this document

## Update the Android Plugin for Gradle

When you update Android Studio, you may receive a prompt to automatically update the Android plugin for 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.

You can specify the Android plugin for Gradle (https://developer.android.google.cn/tools/building/plugin-for-gradle.html) 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

```
buildscript {
    repositories {
        // Gradle 4.1 and higher include support for Google's Maven repo using
        // the google() method. And you need to include this repo to download
        // Android plugin 3.0.0 or higher.
        google()
        ...
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.0.1'
    }
}
```

> **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 **Tools** > **Android** > **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 plugin for Gradle. For the best performance, you should use the latest possible version of both Gradle and the Android 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+ |
| 2.3.0+ | 3.3+ |
| 3.0.0+ | 4.1+ |

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 4.1 in the `gradle-wrapper.properties` file.

```
...
distributionUrl = https\://services.gradle.org/distributions/gradle-4.1-all.zip
...
```

## 3.0.0 (October 2017)

Android plugin for Gradle 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 | ~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. To learn more, read Migrate to Android Plugin for Gradle 3.0.0 (https://developer.android.google.cn/studio/build/gradle-plugin-3-0-0-migration.html).

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 (https://developer.android.google.cn/studio/releases/build-tools.html#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 (https://developer.android.google.cn/studio/build/gradle-plugin-3-0-0-migration.html#new_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 (https://developer.android.google.cn/studio/build/multidex.html#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

- Improved incremental resource processing using AAPT2, which is now enabled by default. If you are experiencing issues while using AAPT2, please report a bug (https://developer.android.google.cn/studio/report-bugs.html). 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 (https://developer.android.google.cn/studio/build/gradle-plugin-3-0-0-migration.html#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 (https://developer.android.google.cn/topic/instant-apps/index.html) and the Android Instant Apps SDK (which you can download using the SDK manager (https://developer.android.google.cn/studio/intro/update.html#sdk-manager)). To learn more about creating Feature modules with the new plugin, read Structure of an instant app with multiple features (https://developer.android.google.cn/topic/instant-apps/getting-started/structure.html#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 (https://developer.android.google.cn/studio/write/java8-support.html).
- Added support for running tests with Android Test Orchestrator (https://developer.android.google.cn/training/testing/junit-runner.html#using-android-test-orchestrator), which allows you to run each of your app's tests within its own invocation of `Instrumentation (https://developer.android.google.cn/reference/android/app/Instrumentation.html)`. Because each test runs in its own `Instrumentation (https://developer.android.google.cn/reference/android/app/Instrumentation.html)` 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 (https://developer.android.google.cn/reference/android/app/Instrumentation.html)`, 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 (https://developer.android.google.cn/training/testing/junit-runner.html#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.

```
android {
      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:

```
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 (https://developer.android.google.cn/guide/topics/ui/look-and-feel/fonts-in-xml.html) (which is a new feature introduced in Android 8.0 (API level 26) (https://developer.android.google.cn/about/versions/oreo/index.html)).
- Support for language-specific APKs with Android Instant Apps SDK 1.1 (https://developer.android.google.cn/topic/instant-apps/release-

(https://developer.android.google.cn/topic/instant-apps/guides/config-splits.html).

- You can now change the output directory for your external native build project, as shown below:

```
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 (https://developer.android.google.cn/studio/projects/add-native-code.html#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:

```
dependencies {
    // This tells the Gradle plugin to build ':lint-checks' into a lint.jar file
    // 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 the app.
    lintChecks project(':lint-checks')
}
```

- 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 (https://developer.android.google.cn/studio/build/gradle-plugin-3-0-0-migration.html#variant_api) in the migration guide.

- 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 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 (https://developer.android.google.cn/studio/write/convert-webp.html#convert_images_to_webp).

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

- The Android plugin now automatically builds executable targets that you configure in your external CMake projects.

- You must now add annotation processors to the processor classpath using the `annotationProcessor` dependency configuration (https://developer.android.google.cn/studio/build/gradle-plugin-3-0-0-migration.html#annotationProcessor_config).

- 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 (https://developer.android.google.cn/studio/projects/add-native-code.html#ndkCompile).

To learn more about issues that are still being addressed, see the Known issues (https://developer.android.google.cn/studio/build/gradle-plugin-3-0-0-migration.html#known_issues).

## 2.3.0 (February 2017)

**2.3.3 (June 2017)**

This is a minor update that adds compatibility with Android Studio 2.3.3 (https://developer.android.google.cn/studio/releases/index.html#Revisions).

**2.3.2 (May 2017)**

This is a minor update that adds compatibility with Android Studio 2.3.2 (https://developer.android.google.cn/studio/releases/index.html#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 (https://developer.android.google.cn/studio/run/index.html#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 (https://developer.android.google.cn/tools/revisions/build-tools.html) or higher.

New:

- Uses Gradle 3.3, which includes performance improvements and new features. For more details, see the Gradle 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 (https://developer.android.google.cn/studio/build/build-cache.html).
    - Includes a `cleanBuildCache` task that clears the build cache (https://developer.android.google.cn/studio/build/build-cache.html#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 (https://developer.android.google.cn/studio/releases/index.html).
- Configuration times for very large projects should be significantly faster.
- Fixed issues with auto-downloading for the constraint layout library (https://developer.android.google.cn/training/constraint-layout/index.html).
- 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 (https://developer.android.google.cn/studio/report-bugs.html) when you encounter issues.

## 2.2.0 (September 2016)

Dependencies:

- Build Tools 23.0.2 (https://developer.android.google.cn/tools/revisions/build-tools.html) 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 {}` (http://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 (https://developer.android.google.cn/studio/projects/add-native-code.html).
- When you run a build from the command line (https://developer.android.google.cn/studio/build/building-cmdline.html), 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 (https://developer.android.google.cn/studio/intro/update.html#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 (https://developer.android.google.cn/studio/build/build-cache.html) guide.
- Improves build performance by adopting a new default packaging pipeline which handles zipping, signing, and zipaligning (https://developer.android.google.cn/studio/command-line/zipalign.html) 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 (https://developer.android.google.cn/about/versions/nougat/android-7.0.html#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:

```
android {
```

This site uses cookies to store your preferences for site-specific language and display options.

OK

```
      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` (http://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` (http://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>` (https://developer.android.google.cn/guide/topics/manifest/application-element.html) element of your app manifest, Gradle packages uncompressed and aligned versions of your native libraries with your APK. This prevents `PackageManager` (https://developer.android.google.cn/reference/android/content/pm/PackageManager.html) 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` (http://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` (http://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

- Improved Jack compiler performance and features:
  - Jack now supports Jacoco test coverage when setting `testCoverageEnabled (http://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:

```
android {
  ...
  defaultConfig {
    ...
    javaCompileOptions {
      annotationProcessorOptions {
        className 'com.example.MyProcessor'
        // Arguments are optional.
        arguments = [ foo : 'bar' ]
      }
    }
  }
}
```

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

```
dependencies {
    compile 'com.google.dagger:dagger:2.0'
    annotationProcessor 'com.google.dagger:dagger-compiler:2.0'
   // or use buildVariantAnnotationProcessor to target a specific build variant
}
```

- You can set additional flags for Jack using `jackOptions.additionalParameters()` (https://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.JackOptions.html#com.android.build.gradle.internal.dsl.JackOptions:additionalParameters). The following code snippet sets the `jack.incremental` parameter to `true`:

  ```
  android {
    defaultConfig {
      ...
      jackOptions {
        enabled true
        additionalParameters("jack.incremental" : true)
      }
    }
  }
  ```

  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

This site uses cookies to store your preferences for site-specific language and display options.

OK

Dependencies:

- Gradle 2.10 or higher.
- Build Tools 23.0.2 (https://developer.android.google.cn/tools/revisions/build-tools.html) or higher.

New:

- Added support for the N Developer Preview, JDK 8, and Java 8 language features (https://developer.android.google.cn/preview/j8-jack.html) using the Jack toolchain. To find out more, read the N Preview guide (https://developer.android.google.cn/preview/overview.html).

  > **Note:** Instant Run (https://developer.android.google.cn/tools/building/building-studio.html#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:

```
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:

```
org.gradle.jvmargs = -Xmx2048m
```

If you have defined a value for `javaMaxHeapSize` (http://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:

```
org.gradle.jvmargs = -Xmx3072m
```

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

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

## 2.0.0 (April 2016)

Dependencies:

- Gradle 2.10 or higher.
- Build Tools 21.1.1 (https://developer.android.google.cn/tools/revisions/build-tools.html) or higher.

New:

- Enables Instant Run (https://developer.android.google.cn/tools/building/building-studio.html#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

- Added `maxProcessCount` (http://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 slave 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:

```
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:

```
android {
  ...
  buildTypes {
    debug {
      minifyEnabled true
      useProguard false
    }
    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 (https://developer.android.google.cn/training/articles/keystore.html) provider no longer supports DSA keys on Android 6.0 (https://developer.android.google.cn/about/versions/marshmallow/android-6.0-changes.html#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)*
- Android plugin for Gradle, revision 1.3.1 (#) *(August 2015)*
- Android plugin for Gradle, revision 1.3.0 (#) *(July 2015)*
- Android plugin for Gradle, revision 1.2.0 (#) *(April 2015)*
- Android plugin for Gradle, revision 1.1.3 (#) *(March 2015)*
- Android plugin for Gradle, revision 1.1.2 (#) *(February 2015)*
- Android plugin for Gradle, revision 1.1.1 (#) *(February 2015)*

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

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

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