Projects overview

A *project* in Android Studio contains everything that defines your workspace for an app, from source code and assets to test code and build configurations.

When you start a new project, Android Studio creates the necessary structure for all your files and makes them visible in the **Project** window in Android Studio. To open the window, select **View > Tool Windows > Project**.

This page provides an overview of the key components inside your project.

Modules

A *module* is a collection of source files and build settings that let you divide your project into discrete units of functionality. Your project can have one or many modules, and one module can use another module as a dependency. You can independently build, test, and debug each module.

Additional modules are useful when creating code libraries within your own project or when you want to create different sets of code and resources for different device types, such as phones and wearables, but keep all the files scoped within the same project and share some code.

To add a new module to your project, click **File > New > New Module**.

Android Studio offers a few distinct types of modules:

Android app module

Provides a container for your app's source code, resource files, and app-level settings, such as the module-level build file and Android Manifest file. When you create a new project, the default app module is named "app."
Android Studio offers the following types of app modules:

- Phone & Tablet
- Automotive
- Wear OS
- Television

- Baseline Profile Generator
- Benchmark

Each module provides essential files and some code templates that are appropriate for the corresponding app or device type.

For more information on adding a module, read Add a module for a new device (/studio/projects/add-app-module).

Feature module

Represents a modularized feature of your app that can take advantage of *Play Feature Delivery*. For example, with feature modules, you can provide your users with certain features of your app on demand or as instant experiences through Google Play Instant (/topic/google-play-instant/overview).

Android Studio offers the following types of feature modules:

- Dynamic Feature Module
- Instant Dynamic Feature Library Module

To learn more, read about Play Feature Delivery (/studio/projects/dynamic-delivery).

Library module

Provides a container for your reusable code, which you can use as a dependency in other app modules or import into other projects. Structurally, a library module is the same as an app module, but when built, it creates a code archive file instead of an APK, so it can't be installed on a device.

In the **Create New Module** window, Android Studio offers the following types of library modules:

- **Android Library:** Contains all file types supported in an Android project except native C++ code, including Java and Kotlin source code, resources, and manifest files. The build result is an Android Archive (AAR) file that you can add as a dependency for your Android app modules.
- **Android Native Library:** Contains all file types supported in an Android project, similar to an Android Library. However, Android Native Libraries also can contain native C++ source code. The build result is an Android Archive (AAR) file that you can add as a dependency for your Android app modules.

- **Java or Kotlin Library:** Contains only Kotlin or Java source files. The build result is a Java Archive (JAR) file that you can add as a dependency for your Android app modules or otherKotlin or Java projects.

Modules are sometimes referred to as *subprojects*, because Gradle also refers to modules as projects.

When you create a library module and want to add it as a dependency to your Android app module, you must declare it as follows:

GroovyKotlin (#kotlin)
(#groovy)

```
dependencies {
    implementation project(':my-library-module')
}
```

Project files

By default, Android Studio displays your project files in the **Android** view. This view doesn't reflect the actual file hierarchy on disk. Instead, it's organized by modules and file types to simplify navigation between key source files of your project, hiding certain files or directories that are not commonly used.

Some of the structural differences between the **Android** view and the structure on disk are that the **Android** view:

- Shows all the project's build-related configuration files in a top-level **Gradle Script** group.
- Shows all manifest files for each module in a module-level group when you have different manifest files for different product flavors and build types.
- Shows all alternative resource files in a single group instead of in separate folders per resource qualifier. For example, all density versions of your launcher icon are visible side by side.

Within each Android app module, files are shown in the following groups:

manifests

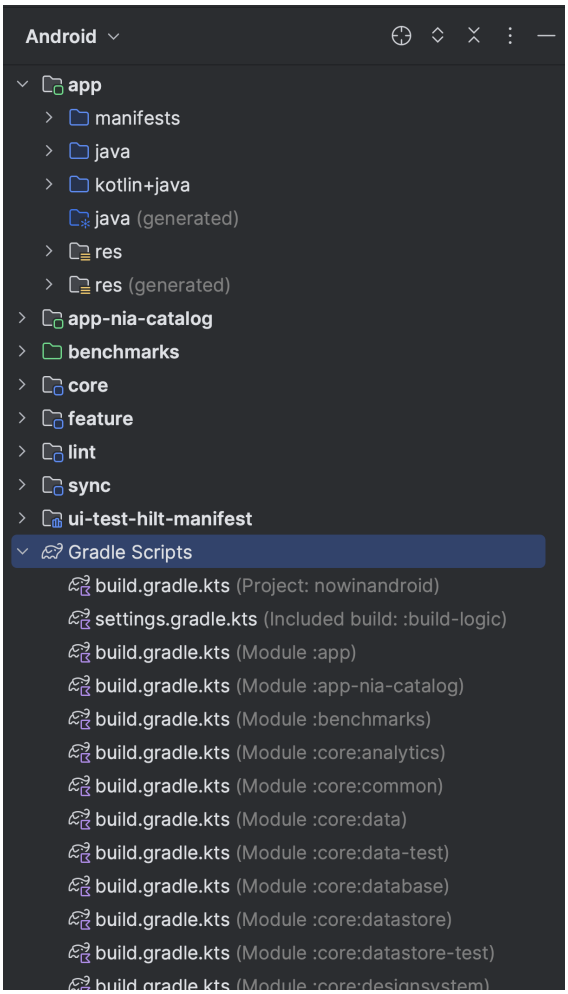
Contains the `AndroidManifest.xml` (/guide/topics/manifest/manifest-intro) file.

java

Contains the Kotlin and Java source code files, separated by package names, including JUnit test code.

res

Contains all non-code resources, such as UI strings and bitmap images, divided into corresponding subdirectories. For more information about possible resource types, see App resources overview



(/guide/topics/resources/providing-resources).

Note: Different module types have different group structures. For example, a native module also contains a **`cpp/`** folder inside the **`main`** group, and a Kotlin or Java library doesn't contain **`manifests`** or **`res`** groups.

The Project view

To see the actual file structure of the project, including all files hidden from the **Android** view, select **Project** from the menu at the top of the **Project** window.

When you select the **Project** view, you can see a lot more files and directories, including the following:

`module-name/`

`build/`

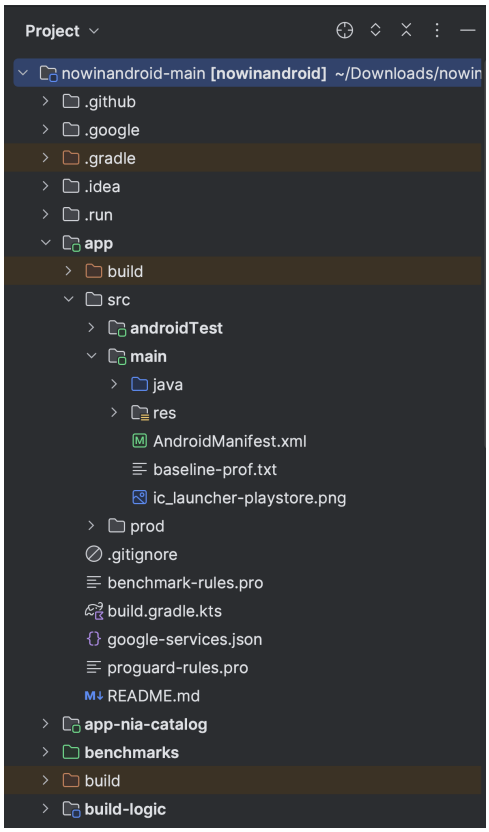
Contains build outputs.

`libs/`

Contains private libraries.

`src/`

Contains all code and resource files for the module in the following subdirectories:



`androidTest/`

Contains code for instrumentation tests that run on an Android device. For more information, see Test in Android Studio (/studio/test/test-in-android-studio).

`cpp/`

Contains native C or C++ code using the Java Native Interface (JNI). For more information, see the Android NDK documentation (/ndk).

`main/`

Contains the "main" source set files: the Android code and resources shared by all build variants (files for other build variants reside in sibling directories, such as `src/debug/` for the debug build type):

`AndroidManifest.xml`

Describes the nature of the application and each of its components. For more information, see the app manifest (/guide/topics/manifest/manifest-intro) overview.

`java/`

Contains Kotlin or Java code sources, or both, if your app has both Kotlin and Java source code.

`kotlin/`

Contains only Kotlin code sources.

`res/`

Contains application resources, such as drawable files and UI string files. For more information, see the app resources (/guide/topics/resources) overview.

`assets/`

Contains files to be compiled into an APK file as-is. For example, this is a good location for textures and game data. You can navigate this directory in the same way as a typical file system, using URIs and read files as a stream of bytes using the `AssetManager` (/reference/android/content/res/AssetManager).

`test/`

Contains code for local tests that run on your host JVM.

`build.gradle` **or** `build.gradle.kts` **(module)**

This defines the module-specific build configurations. `build.gradle` is the correct filename if you're using Groovy as your build script language, and it's `build.gradle.kts` if you're using Kotlin script.

`build.gradle` **or** `build.gradle.kts` **(project)**

This defines your build configuration that applies to all modules. `build.gradle` is the correct filename if you're using Groovy as your build script language, and it's `build.gradle.kts` if you're using Kotlin script. This file is integral to the project, so maintain it in revision control with all other source code.

For information about other build files, see Configure your build (/studio/build).

Note: To change the default view from the **Android** view to the **Project** view, select **Help > Edit Custom Properties** and add `studio.projectview=true`.

Project structure settings

To change various settings for your Android Studio project, open the **Project Structure** dialog by clicking **File > Project Structure**. It contains the following sections:

- **Project:** Sets the version for Gradle and the Android Gradle plugin (/studio/build#build-files) and the repository location name.

- **SDK Location:** Sets the location of the JDK, Android SDK, and Android NDK that your project uses.
- **Variables:** Lets you edit variables that are used within your build scripts.
- **Modules:** Lets you edit module-specific build configurations, including the target and minimum SDK, the app signature, and library dependencies. Each module's settings page is divided into the following tabs:
  - **Properties:** Specifies the versions of the SDK and build tools to use to compile the module.
  - **Signing:** Specifies the certificate to use to sign your app (/tools/publishing/app-signing#sign-auto).
- **Dependencies:** Lists the library, file, and module dependencies for this module. You can add, modify, and delete dependencies from this pane. For more information about module dependencies, see Configure build variants (/tools/building/configuring-gradle#declareDeps).

Preview: If you're using a version catalog, be aware of the issue that dependencies added through the **Project Structure** dialog are added to the module's `build.gradle` file, not the catalog. To learn more about Gradle Version Catalogs support in Android Studio, see the preview release note (/studio/preview/features#gradle-version-catalogs).

- **Build Variants:** Lets you configure different flavors and build types for your project.
  - **Flavors:** Lets you create multiple build *flavors*, where each flavor specifies a set of configuration settings, such as the module's minimum and target SDK version and the version code and version name (/studio/publish/versioning#versioningsettings.html).

    For example, you might define one flavor that has a minimum SDK of 21 and a target SDK of 29, and another flavor that has a minimum SDK of 24 and a target SDK of 33.
  - **Build Types:** Lets you create and modify build configurations, as described in Configure build variants (/tools/building/configuring-gradle). By default, every module has *debug* and *release* build types, and you can define more as needed.

Last updated 2023-07-24 UTC.