Install Android Studio

Setting up Android Studio takes just a few clicks.

While the [Android Studio download](http://developer.android.com/sdk/index.html) completes, verify which version of the JDK you have: open a command line and type javac -version. If the JDK is not available or the version is lower than 1.8, download the [Java SE Development Kit 8](http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html).

To install Android Studio on Windows, proceed as follows:

1. Launch the .exe file you downloaded.
2. Follow the setup wizard to install Android Studio and any necessary SDK tools.

On some Windows systems, the launcher script does not find where the JDK is installed. If you encounter this problem, you need to set an environment variable indicating the correct location.

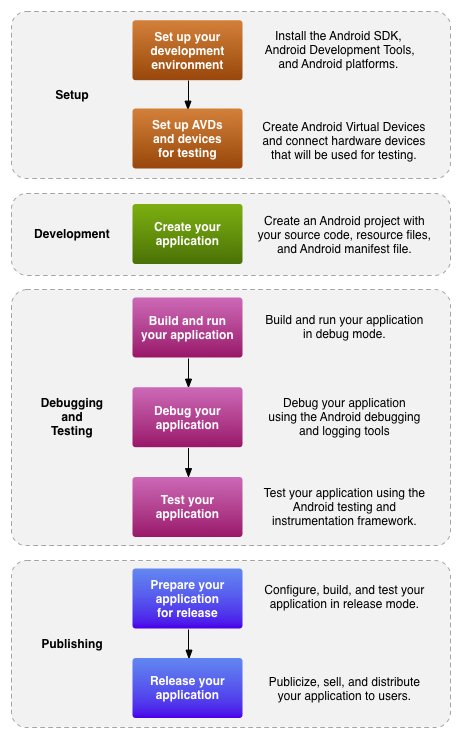
Select **Start menu > Computer > System Properties > Advanced System Properties**. Then open **Advanced tab > Environment Variables** and add a new system variable JAVA\_HOME that points to your JDK folder, for exampleC:\Program Files\Java\jdk1.8.0\_77.

As new tools and other APIs become available, Android Studio tells you with a pop-up, or you can check for updates by clicking **Help > Check for Update**.

# Developer Workflow

To develop apps for Android, you use a set of tools that are included in Android Studio. In addition to using the tools from Android Studio, you can also access most of the SDK tools from the command line. Developing with Android Studio is the preferred method because it can directly invoke the tools that you need while developing applications.

However, you may choose to develop with another IDE or a simple text editor and invoke the tools on the command line or with scripts. This is a less streamlined way to develop because you will sometimes have to call command line tools manually, but you will have access to the same number of features that you would have in Android Studio.



**Figure 1.** The development process for Android applications.

## **App Workflow**

The basic steps for developing applications (with or without Android Studio) are shown in figure 1. The development steps encompass four development phases, which include:

* **Environment Setup**

During this phase you install and set up your development environment. You also create Android Virtual Devices (AVDs) and connect hardware devices on which you can install your applications.

See [Managing Virtual Devices](http://developer.android.com/tools/devices/index.html) and [Using Hardware Devices](http://developer.android.com/tools/device.html) for more information.

* **Project Setup and Development**

During this phase you set up and develop your Android Studio project and application modules, which contain all of the source code and resource files for your application. For more information, see [Create an Android project](http://developer.android.com/tools/projects/index.html).

* **Building, Debugging and Testing**

During this phase you build your project into a debuggable .apk package(s) that you can install and run on the emulator or an Android-powered device. Android Studio uses a build system based on [Gradle](http://www.gradle.org/)that provides flexibility, customized build variants, dependency resolution, and much more. If you're using another IDE, you can build your project using Gradle and install it on a device using [adb](http://developer.android.com/tools/help/adb.html). For more information, see [Build and run your application](http://developer.android.com/tools/building/index.html).

Next, with Android Studio you debug your application using the [Android Device Monitor](http://developer.android.com/tools/help/monitor.html) and device log messages ([logcat](http://developer.android.com/tools/help/logcat.html)) along with the IntelliJ IDEA intelligent coding features. You can also use a JDWP-compliant debugger along with the debugging and logging tools that are provided with the Android SDK. For more information see [Debug your application with the SDK debugging and logging tools](http://developer.android.com/tools/debugging/index.html).

## **Android Project Files**

Android Studio project files and settings provide project-wide settings that apply across all modules in the project.

.idea

Directory for IntelliJ IDEA settings.

app

Application module directories and files.

build

This directory stores the build output for all project modules.

gradle

Contains the gradler-wrapper files.

.gitignore

Specifies the untracked files that Git should ignore.

build.gradle

Customizable properties for the build system. You can edit this file to specify the default build settings used by the application modules and also set the location of your keystore and key alias so that the build tools can sign your application when building in release mode. This file is integral to the project, so maintain it in a source revision control system.

gradle.properties

Project-wide Gradle settings.

gradlew

Gradlestartup script for Unix.

gradlew.bat

Gradlestartup script for Windows.

local.properties

Customizable computer-specific properties for the build system, such as the path to the SDK installation. Because the content of the file is specific to the local installation of the SDK, the local.properties should not be maintained in a source revision control system.

.iml

Module file created by the IntelliJ IDEA to store module information.

settings.gradle

Specifies the sub-projects to build.

## **Android Application Modules**

Android Application Modules are the modules that eventually get built into the .apk files based on your build settings. They contain things such as application source code and resource files. Most code and resource files are generated for you by default, while others should be created if required. The following directories and files comprise an Android application module:

build/

Contains build folders for the specified build variants. Stored in the main application module.

libs/

Contains private libraries. Stored in the main application module.

src/

Contains your stub Activity file, which is stored at src/main/java/<namespace.appname>/<ActivityName>.java. All other source code files (such as .java or .aidl files) go here as well.

androidTest/

Contains the instrumentation tests. For more information, see the [Android Test documentation](http://developer.android.com/tools/testing/index.html).

main/java/com.<project>.<app>

Contains Java code source for the app activities.

main/jni/

Contains native code using the Java Native Interface (JNI). For more information, see the [Android NDK documentation](http://developer.android.com/tools/sdk/ndk/index.html).

Contains the Java files generated by Android Studio, such as your R.java file and interfaces created from AIDL files.

This is empty. You can use it to store raw asset files. Files that you save here are compiled into an .apk file as-is, and the original filename is preserved. 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](http://developer.android.com/reference/android/content/res/AssetManager.html). For example, this is a good location for textures and game data.

Contains application resources, such as drawable files, layout files, and string values in the following directories. See[Application Resources](http://developer.android.com/guide/topics/resources/index.html) for more information.

anim/

For XML files that are compiled into animation objects. See the [Animation](http://developer.android.com/guide/topics/resources/animation-resource.html) resource type.

color/

For XML files that describe colors. See the [Color Values](http://developer.android.com/guide/topics/resources/color-list-resource.html) resource type.

drawable/

For bitmap files (PNG, JPEG, or GIF), 9-Patch image files, and XML files that describe Drawable shapes or Drawable objects that contain multiple states (normal, pressed, or focused). See the [Drawable](http://developer.android.com/guide/topics/resources/drawable-resource.html) resource type.

mipmap/

For app launcher icons. The Android system retains the resources in this folder (and density-specific folders such as mipmap-xxxhdpi) regardless of the screen resolution of the device where your app is installed. This behavior allows launcher apps to pick the best resolution icon for your app to display on the home screen. For more information about using the mipmap folders, see [Managing Launcher Icons as mipmap Resources](http://developer.android.com/tools/projects/index.html#mipmap).

layout/

XML files that are compiled into screen layouts (or part of a screen). See the [Layout](http://developer.android.com/guide/topics/resources/layout-resource.html) resource type.

menu/

For XML files that define application menus. See the [Menus](http://developer.android.com/guide/topics/resources/menu-resource.html) resource type.

raw/

For arbitrary raw asset files. Saving asset files here is essentially the same as saving them in the assets/directory. The only difference is how you access them. These files are processed by aapt and must be referenced from the application using a resource identifier in the R class. For example, this is a good place for media, such as MP3 or Ogg files.

values/

For XML files that define resources by XML element type. Unlike other resources in the res/ directory, resources written to XML files in this folder are not referenced by the file name. Instead, the XML element type controls how the resources defined within the XML files are placed into the R class.

xml/

For miscellaneous XML files that configure application components. For example, an XML file that defines a[PreferenceScreen](http://developer.android.com/reference/android/preference/PreferenceScreen.html), [AppWidgetProviderInfo](http://developer.android.com/reference/android/appwidget/AppWidgetProviderInfo.html), or [Searchability Metadata](http://developer.android.com/reference/android/app/SearchManager.html#SearchabilityMetadata). See [Application Resources](http://developer.android.com/guide/topics/resources/index.html) for more information about configuring these application components.

The control file that describes the nature of the application and each of its components. For instance, it describes: certain qualities about the activities, services, intent receivers, and content providers; what permissions are requested; what external libraries are needed; what device features are required, what API Levels are supported or required; and others. See the [AndroidManifest.xml](http://developer.android.com/guide/topics/manifest/manifest-intro.html) documentation for more information