

Android OS

Android is a mobile operating system that is based on a modified version of Linux. It was originally developed by a startup of the same name, Android, Inc. In 2005, as part of its strategy to enter the mobile space, Google purchased Android and took over its development work (as well as its development team).

Google wanted Android to be open and free; hence, most of the Android code was released under the open-source Apache License, which means that anyone who wants to use Android can do so by downloading the full Android source code. Moreover, vendors (typically hardware manufacturers) can add their own proprietary extensions to Android and customize Android to differentiate their products from others. This simple development model makes Android very attractive and has thus piqued the interest of many vendors.

The main advantage of adopting Android is that it offers a unified approach to application development. Developers need only develop for Android, and their applications should be able to run on numerous different devices, as long as the devices are powered using Android. In the world of smartphones, applications are the most important part of the success chain.

Features of Android

As Android is open source and freely available to manufacturers for customization, there are no fixed hardware and software configurations. However, Android itself supports the following features:

Storage — Uses SQLite, a lightweight relational database, for data storage.

Connectivity — Supports GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth (includes A2DP and AVRCP), WiFi, LTE, and WiMAX.

Messaging — Supports both SMS and MMS.

Web browser — Based on the open-source WebKit, together with Chrome's V8 JavaScript engine supporting HTML5 and CSS3.

Media support — Includes support for the following media: H.263, H.264 (in 3GP or MP4 container), MPEG-4 SP, AMR, AMR-WB (in 3GP container), AAC, HE-AAC (in MP4 or 3GP container), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP

Hardware support — Accelerometer Sensor, Camera, Digital Compass, Proximity Sensor, and GPS

Multi-touch — Supports multi-touch screens

Multi-tasking — Supports multi-tasking applications

Flash support — Android 2.3 supports Flash 10.1.

Tethering — Supports sharing of Internet connections as a wired/wireless hotspot

Beautiful UI - Android OS basic screen provides a beautiful and intuitive user interface.

Resizable widgets - Widgets are resizable, so users can expand them to show more content or shrink them to save space

Multi-Language - Support single direction and bi-directional text.

GCM - Google Cloud Messaging (GCM) is a service that let developers send short message data to their users on Android devices, without needing a proprietary sync solution.

Wi-Fi Direct - A technology that let apps discover and pair directly, over a high-bandwidth peer-to-peer connection.

Android Beam - A popular NFC-based technology that let users instantly share, just by touching two NFC-enabled phones together.

Different versions of Android

Each version of Android since 1.5 has been developed with a specific codename. These codenames are chosen alphabetically and have thus far all been dessert items (or, generically, sweet/sugary foods). Some code names are associated with more than one version number, while others are limited to only a specific one. The naming typically appears to correspond to changes in the developer API levels, but this is not always true.

The following names are used for the currently existing Android releases. Note that versions 1.0 and 1.1 were not publicly named. However, Android 1.1 was internally referred to as "Petit-Four"

- Android 1.0 (API 1)
- Android 1.1 (API 2)
- Android 1.5 Cupcake (API 3)
- Android 1.6 Donut (API 4)
- Android 2.0 Eclair (API 5)
- Android 2.2 Froyo (API 8)
- Android 2.3 Gingerbread (API 9)
- Android 3.0 Honeycomb (API 11)
- Android 4.0 Ice Cream Sandwich (API 14)
- Android 4.1 Jelly Bean (API 16)
- Android 4.4 KitKat (API 19)
- Android 5.0 Lollipop (API 21)

- Android 6.0 Marshmallow (API 23)
- Android 7.0 Nougat (API 24)
- Android 8.0 Oreo (API 26)
- Android 9 Pie (API 28)
- Android 10 (API 29)

Comparison of Android and Apple's' IOS

These are the two Mobile Operating Systems used primarily in mobile technology, such as smartphones and tablets. Android, which is Linux-bases and partly open source, is more PC-like than IOS, in that its interface and basic features are generally more customizable from top to bottom. However, IOS' uniform design elements are sometimes seen as being more user-friendly. Switching from IOS to Android or vice versa will require you to buy apps again. Android is now the words most commonly used Smartphone platform and is used by many different phone manufacturers. iOS is only used on Apple devices such as the iPhone.

	Android	iOS
Source model	Open source	Closed, with open source components.
OS family	Linux	OS X, UNIX
Initial release	September 23, 2008	July 29, 2007
Customizability	A lot. Can change almost anything.	Limited unless jailbroken
Developer	Google, Open Handset Alliance	Apple Inc.
Widgets	Yes	No, except in NotificationCenter
Available language(s)	100+ Languages	34 Languages

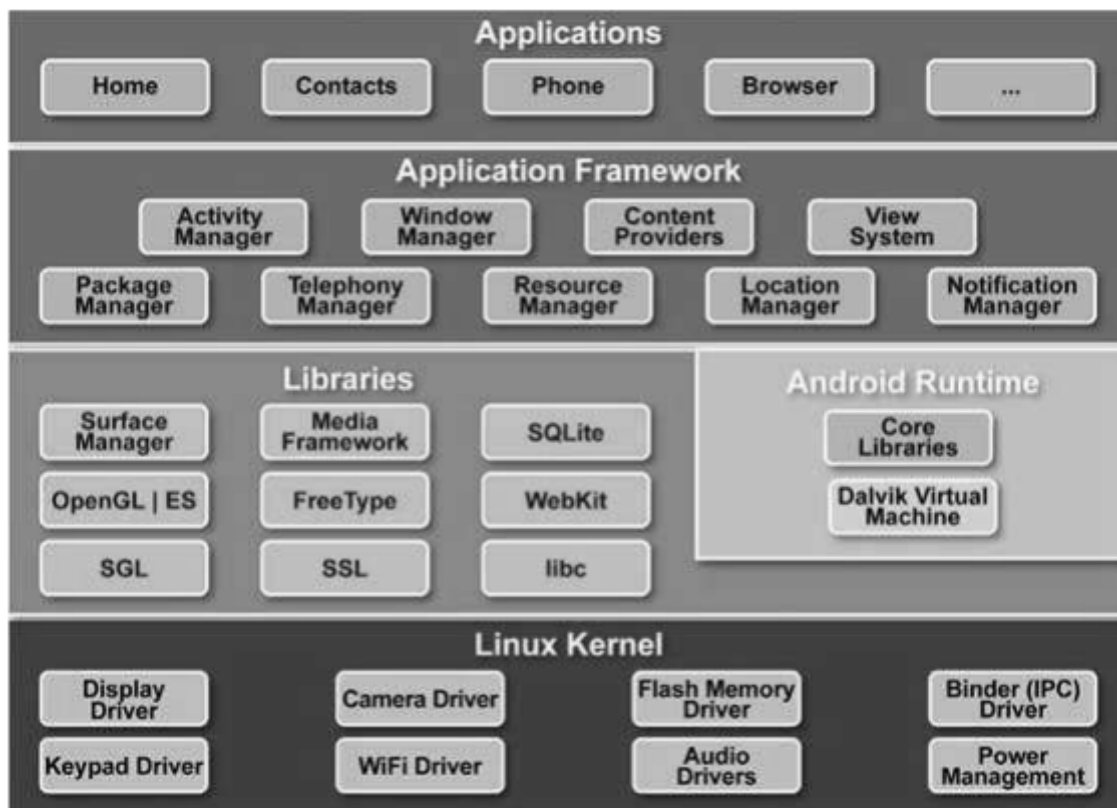
File transfer	Easier than iOS. Using USB port and Android File Transfer desktop app. Photos can be transferred via USB without apps.	More difficult. Media files can be transferred using iTunes desktop app. Photos can be transferred out via USB without apps.
Available on	Many phones and tablets. Major manufacturers are Samsung, Motorola, LG, HTC and Sony.. Nexus and Pixel line of devices is pure Android, others bundle manufacturer software.	iPod Touch, iPhone, iPad, Apple TV (2nd and 3rd generation)
Calls and messaging	Google Hangouts. 3rd party apps like Facebook Messenger, WhatsApp, Google Duo and Skype all work on Android and iOS both.	iMessage, FaceTime (with other Apple devices only). 3rd party apps like Google Hangouts, Facebook Messenger, WhatsApp, Google Duo and Skype all work on Android and iOS both.
Internet browsing	Google Chrome (or Android Browser on older versions; other browsers are available)	Mobile Safari
App store , Affordability and interface	Google Play – 1,000,000+ apps. Other app stores like Amazon and Getjar also distribute Android apps.	Apple app store – 1,000,000+ apps
Video chat	Google Duo and other 3rd party apps	FaceTime (Apple devices only) and other 3rd party apps
Voice commands	Google Now, Google Assistant	Siri
Working state	Current	Current
Maps	Google Maps	Apple Maps

Latest stable release and Updates	Android 10(September 2019)	13 (Dember 5, 2019)
Alternative app stores and side loading	Several alternative app stores other than the official Google Play Store. (e.g. Aptoide, Galaxy Apps)	Apple blocks 3rd party app stores. The phone needs to be jailbroken if you want to download apps from other stores.
Battery life and management	Many Android phone manufacturers equip their devices with large batteries with a longer life.	Apple batteries are generally not as big as the largest Android batteries. However, Apple is able to squeeze decent battery life via hardware/software optimizations.
Open source	Kernel, UI, and some standard apps	The iOS kernel is not open source but is based on the open-source Darwin OS.
File manager	Yes. (Stock Android File Manager included on devices running Android 7.1.1)	Not available
Photos & Videos backup	Apps available for automatic backup of photos and videos. Google Photos allows unlimited backup of photos. OneDrive, Amazon Photos and Dropbox are other alternatives.	Up to 5 GB of photos and videos can be automatically back up with iCloud. All other vendors like Google, Amazon, Dropbox, Flickr and Microsoft have auto-backup apps for both iOS and Android.
Security	Android software patches are available soonest to Nexus device users. Manufacturers tend to lag behind in pushing out these updates. So at any given time a vast majority of Android devices are not running updated fully patched software.	Most people will never encounter a problem with malware because they don't go outside the Play Store for apps. Apple's software updates support older iOS devices also.

Rooting, bootloaders, and jailbreaking	Access and complete control over your device is available and you can unlock the bootloader.	Complete control over your device is not available.
Cloud services	Native integration with Google cloud storage. 15GB free, \$2/mo for 100GB, 1TB for \$10. Apps available for Amazon Photos, OneDrive and Dropbox.	Native integration with iCloud. 5GB free, 50GB for \$1/mo, 200GB for \$3/mo, 1TB for \$10/mo. Apps available for Google Drive and Google Photos, Amazon Photos, OneDrive and Dropbox.
Interface	Touch Screen	Touch Screen
Supported versions	Android 5.0 & later (Android 4.4 is also supported but with patches)	iOS 8 & later
First version	Android 1.0, Alpha	iOS 1.0

Architecture of Android

Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown below in the architecture diagram.



Linux kernel — This is the kernel on which Android is based. This layer contains all the lowlevel device drivers for the various hardware components of an Android device.

At the bottom of the layers is Linux. This provides basic system functionality like process management, memory management, device management like camera, keypad, display etc. Also, the kernel handles all the things that Linux is really good at, such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

Libraries — On top of Linux kernel there is a set of libraries. These contain all the code that provides the main features of an Android OS. For example, the SQLite library provides database support so that an application can use it for data storage. The WebKit library provides functionalities for web browsing. Libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

Android runtime — At the same layer as the libraries, the Android runtime provides a set of core libraries that enable developers to write Android apps using the Java programming language. The Android runtime also includes the Dalvik virtual machine, which enables every Android application to run in its own process; with its own instance of the Dalvik virtual machine (Android applications are compiled into the Dalvik executables). Dalvik is a specialized virtual machine designed specifically for Android and optimized for battery-powered mobile devices with limited memory and CPU.

Application framework — Exposes the various capabilities of the Android OS to application developers so that they can make use of them in their applications. The Application

Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

Applications — At this top layer, you will find applications that ship with the Android device (such as Phone, Contacts, Browser, etc.), as well as applications that you download and install from the Android Market. Any applications that you write are located at this layer.

Android Applications

Android applications are usually developed in the Java language using the Android Software Development Kit. Once developed, Android applications can be packaged easily and sold out either through a store such as Google Play or the Amazon Appstore. Android powers hundreds of millions of mobile devices in more than 190 countries around the world. It's the largest installed base of any mobile platform and is growing fast. Every day more than 1 million new Android devices are activated worldwide. This tutorial has been written with an aim to teach you how to develop and package Android application. We will start from environment setup for Android application programming and then drill down to look into various aspects of Android applications.

Android Application Development Environment Setup (Using Eclipse)

For Android development, you can use a Mac, a Windows PC, or a Linux machine. All the tools needed are free and can be downloaded from the Web freely. Following is the list of software's you will need before you start your Android application programming.

- Java JDK
- Android SDK
- Eclipse IDE for Java Developers
- Android Development Tools (ADT) Eclipse Plugin

JDK

JDK is the java development kit which contains the libraries, debugger, compiler, documentations for java language. For android application development we use java language so JDK is essential.

Eclipse

Eclipse is an integrated development environment (IDE) for Android Application Development. It is a multi-language software development environment featuring an extensible plug-in system. It can be used to develop various types of applications, using

languages such as Java, Ada, C, C++, COBOL, Python, etc. For Android development, you should download the Eclipse IDE for Java EE Developers.

Android SDK

The Android SDK contains a debugger, libraries, an emulator, documentation, sample code, and tutorials.

Android Development Tools (ADT)

The Android Development Tools (ADT) plug-in for Eclipse is an extension to the Eclipse IDE that supports the creation and debugging of Android applications. Using the ADT, you will be able to do the following in Eclipse:

Create new Android application projects.

Access the tools for accessing your Android emulators and devices.

Compile and debug Android applications.

Export Android applications into Android Packages (APK).

Create digital certificates for code-signing your APK

AVD

AVD stands for Android Virtual Devices. An AVD is an emulator instance that enables you to model an actual device. Each AVD consists of a hardware profile, a mapping to a system image, as well as emulated storage, such as a secure digital (SD) card. You can create as many AVDs as you want in order to test your applications with several different configurations. This testing is important to confirm the behaviour of your application when it is run on different devices with varying capabilities.

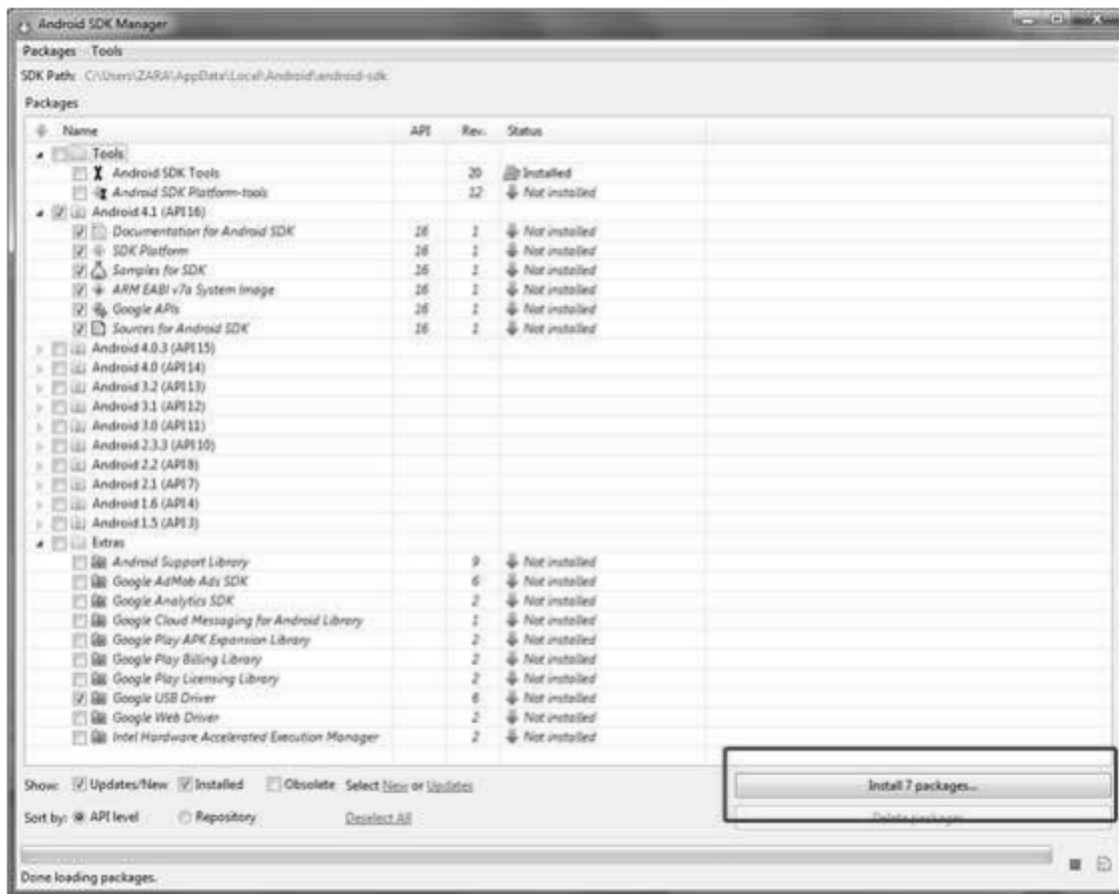
Step 1 -Setup Java Development Kit (JDK)

Download the latest version of Java JDK from Oracle's Java site: Java SE Downloads. You will find instructions for installing JDK in downloaded files, follow the given instructions to install and configure the setup. Finally, set PATH and JAVA_HOME environment variables to refer to the directory that contains java and javac. The methods to setup path and environment variable is different in different Operating System

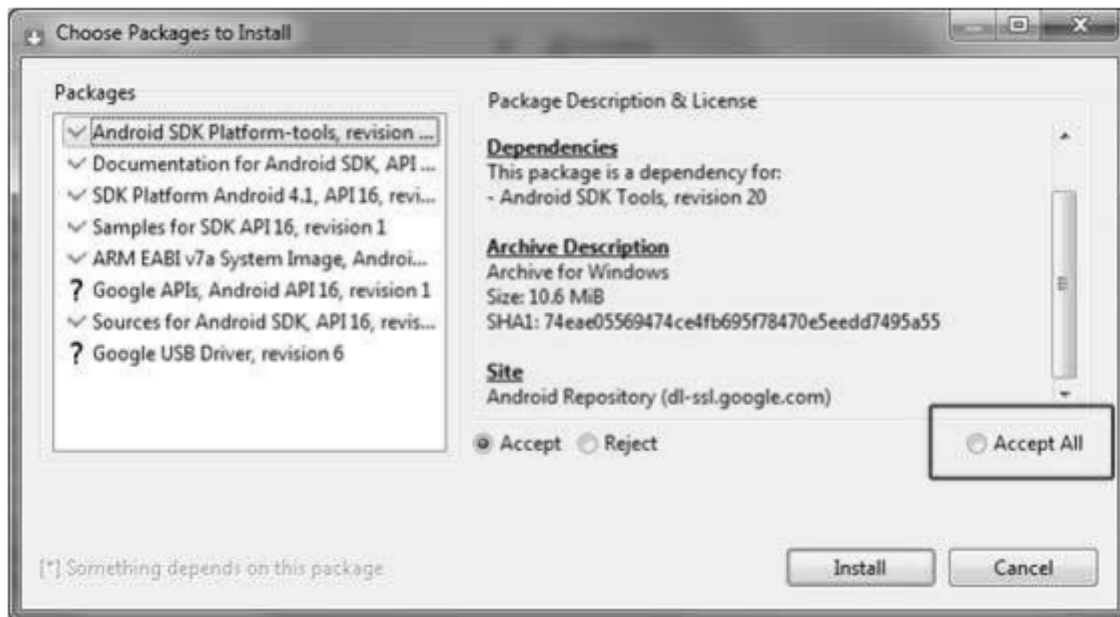
Step 2 -Setup Android SDK

Download the latest version of Android SDK from Android's official website: <http://developer.android.com/sdk/index.html>. If you are installing SDK on Windows machine, then you will find ainstaller_rXX-windows.exe, so just download and run this exe which will launch Android SDK Tool Setup wizard to guide you throughout the installation, so just follow the instructions carefully. Finally, you will have Android SDK Tools installed on your

machine. Then launch Android SDK Manager using the option All Programs > Android SDK Tools > SDK Manager, this will give you following window:



Once you launched SDK manager, it is time to install other required packages. By default it will list down total 7 packages to be installed, but we will suggest to de-select Documentation for Android SDK and Samples for SDK packages to reduce installation time. Next click the Install 7 Packages button to proceed, which will display following dialogue box:



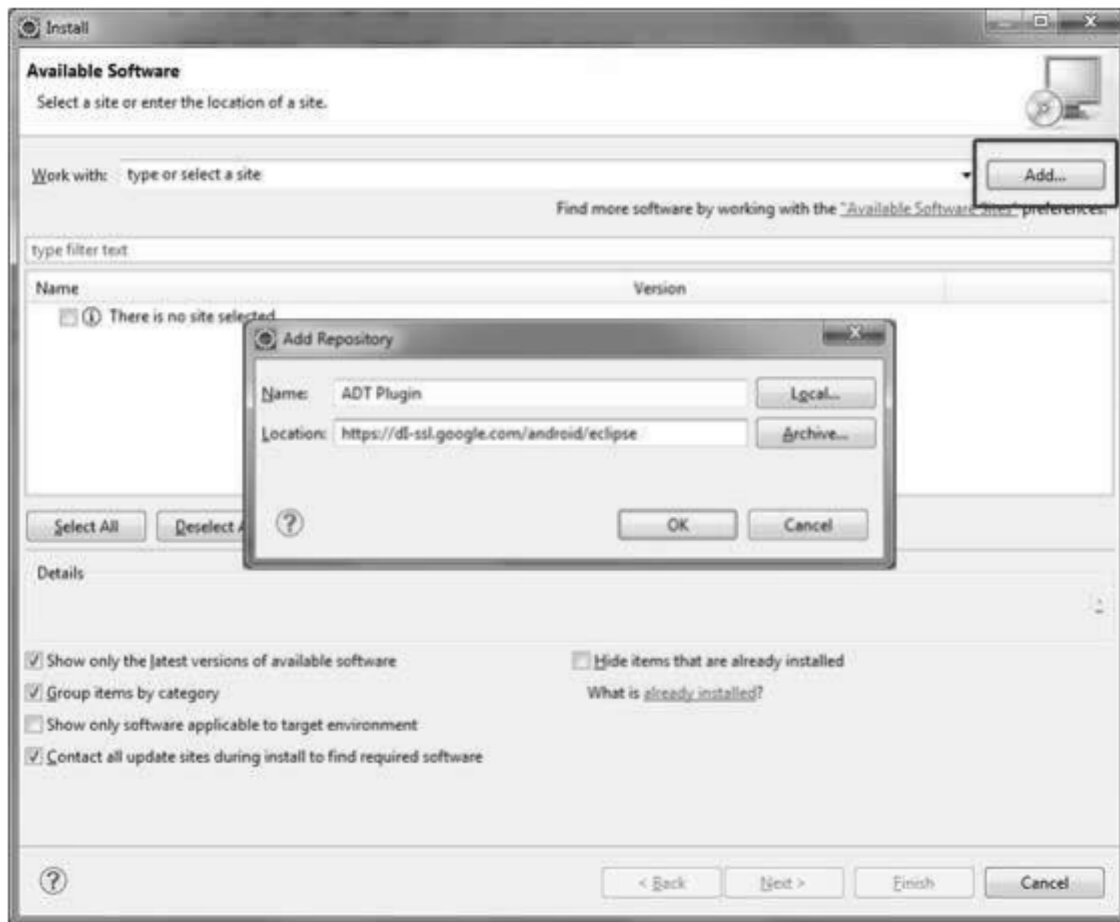
If you agree to install all the packages, select Accept All radio button and proceed by clicking Install button. Close the SDK Manager.

Step 3 -Setup Eclipse IDE

To install Eclipse IDE, download the latest Eclipse binaries from <http://www.eclipse.org/downloads/>. Once you have downloaded the installation, unpack the binary distribution into a convenient location. For example in C:\eclipse on windows, or /usr/local/eclipse on Linux and finally set PATH variable appropriately. Start the Eclipse IDE according the method of your Operating system.

Step 4 -Setup Android Development Tools (ADT) Plug-in

This step will help you in setting Android Development Tool plugin for Eclipse. Let's start with launching Eclipse and then, choose Help > Software Updates > Install New Software. This will display the following dialogue box.

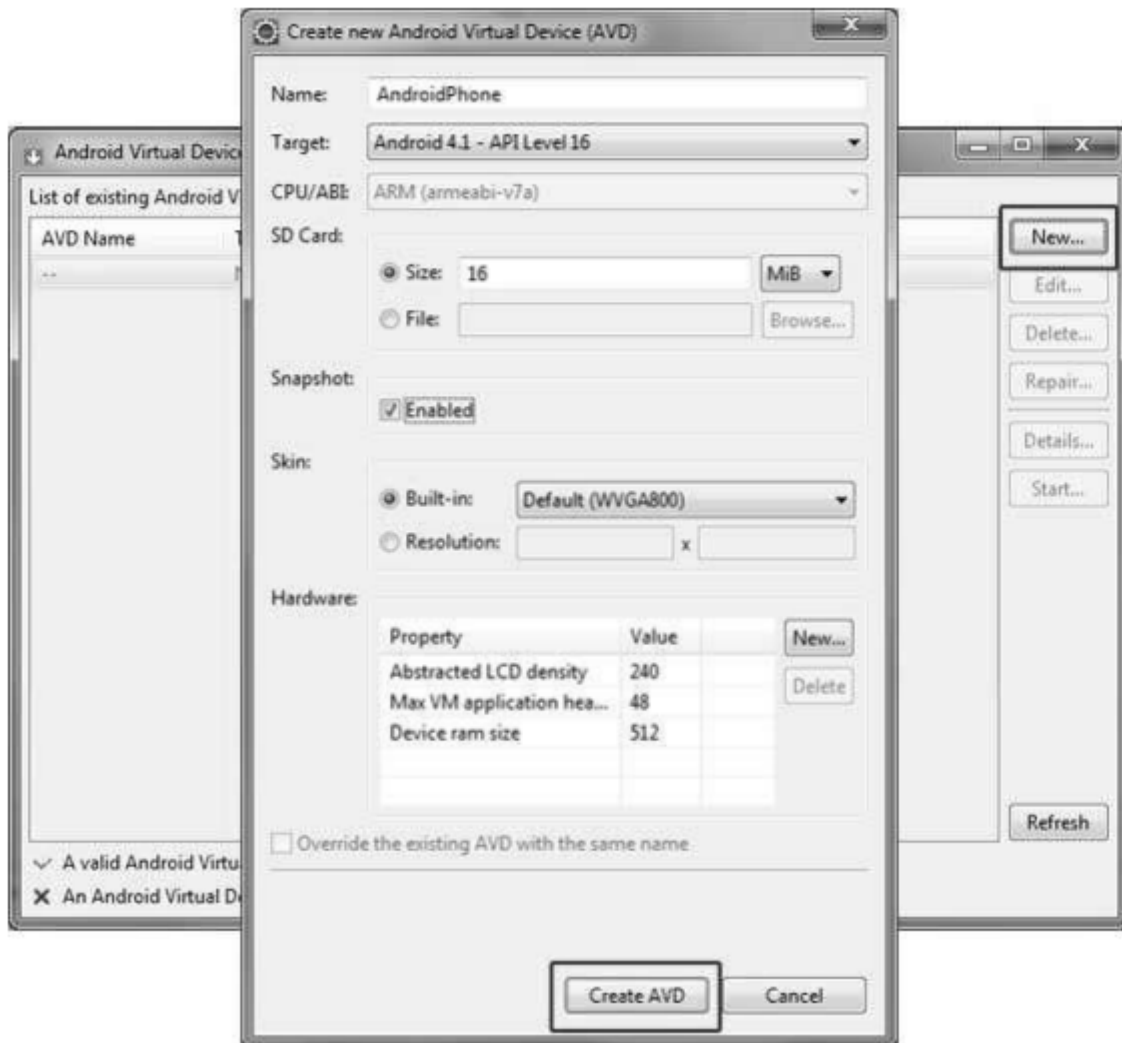


Now use Add button to add ADT Plugin as name and <https://dl-ssl.google.com/android/eclipse/> as the location. Then click OK to add this location. As soon as you will click OK button to add this location, Eclipse starts searching for the plug-in available in the given location and finally lists down the found plugins.

Now select all the listed plug-ins using Select All button and click Next button which will guide you ahead to install Android Development Tools and other required plugins.

Step 5 –Create Android Virtual Device

The next step is to create AVD to be used for testing your Android applications. To test your Android applications you will need a virtual Android device. So before we start writing our code, let us create an Android virtual device. Launch Android AVD Manager using Eclipse menu options Window > AVD Manager> which will launch Android AVD Manager. Use New button to create a new Android Virtual Device and enter the following information, before clicking Create AVD button.



If your AVD is created successfully it means your environment is ready for Android application development.

Android Application Development Environment Setup (Using Android Studio)

Android Studio is Google's officially supported IDE for developing Android apps. Android Studio is Google's officially supported IDE for developing Android apps. Based on IntelliJ IDEA, Android Studio is freely available under Apache License 2.0. The most recent stable version, 3.2.1 october 2018, includes the following features: Before you can use Android Studio to develop software, you'll need to download the most recent Java Development Kit (JDK) for your computer as well.

- A unified environment where you can develop for all Android devices.
- Support for building Android TV apps and Android Wear apps.
- Template-based wizards to create common Android designs and components.
- A rich layout editor that lets users drag-and-drop user interface components, and that offers an option to preview layouts on multiple screen configurations.
- Android-specific refactoring and quick fixes.
- Gradle-based build support.

- Lint tools to catch performance, usability, version compatibility, and other problems.
- ProGuard integration and app-signing capabilities.
- A fast and feature-rich emulator.
- Instant Run to push changes to your running app without building a new APK (Application Package Zip file).
- Built-in support for Google Cloud Platform, enabling integration with Google Cloud Messaging and App Engine.
- C++ and NDK support.
- Plugin architecture for extending Android Studio via plugins.

Google provides Android Studio for the Windows, Mac OS X, and Linux platforms. You can download this software from the Android Studio homepage. The system requirement for Android Studio.

- Microsoft Windows 7/8/10 (32-bit or 64-bit) (for linux GNOME or KDE desktop: Tested on Ubuntu 12.04, Precise Pangolin, GNU C Library (glibc) 2.11 or later)
- 2 GB RAM minimum, 8 GB RAM recommended
- 2 GB of available disk space minimum, 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution
- JDK 8
- For accelerated emulator: 64-bit operating system and Intel processor with support for Intel VT-x, Intel EM64T (Intel 64), and Execute Disable (XD) Bit functionality

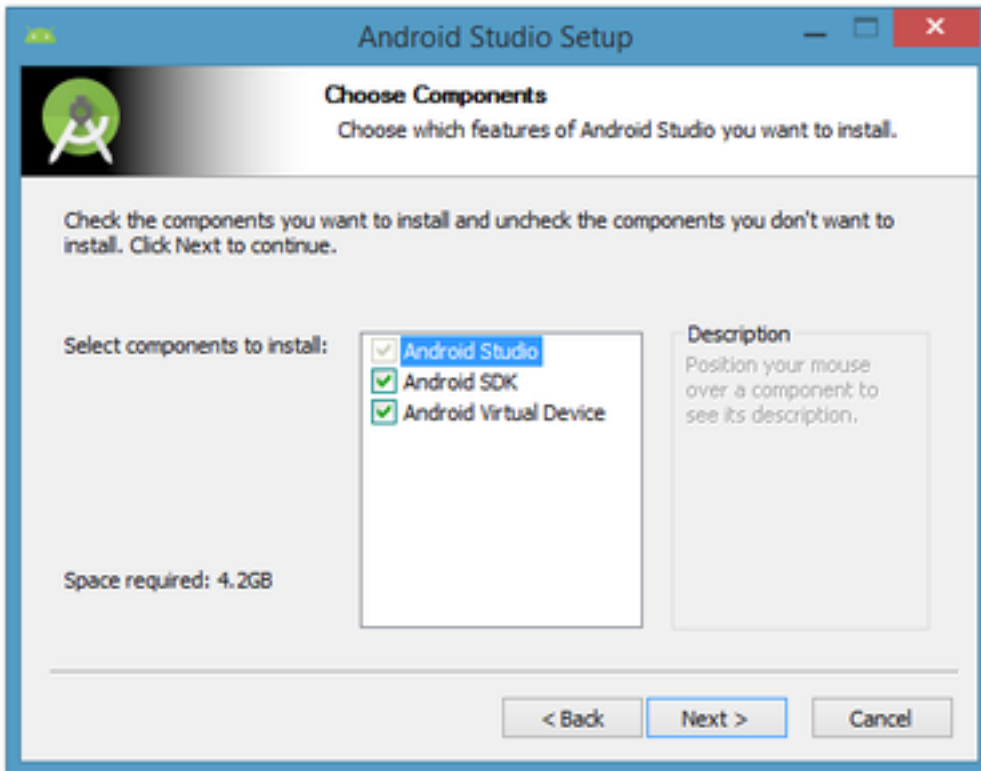
Once you've ensured your operating system is compatible with Android Studio, download the appropriate Android Studio distribution file.

Installing Android Studio on Windows

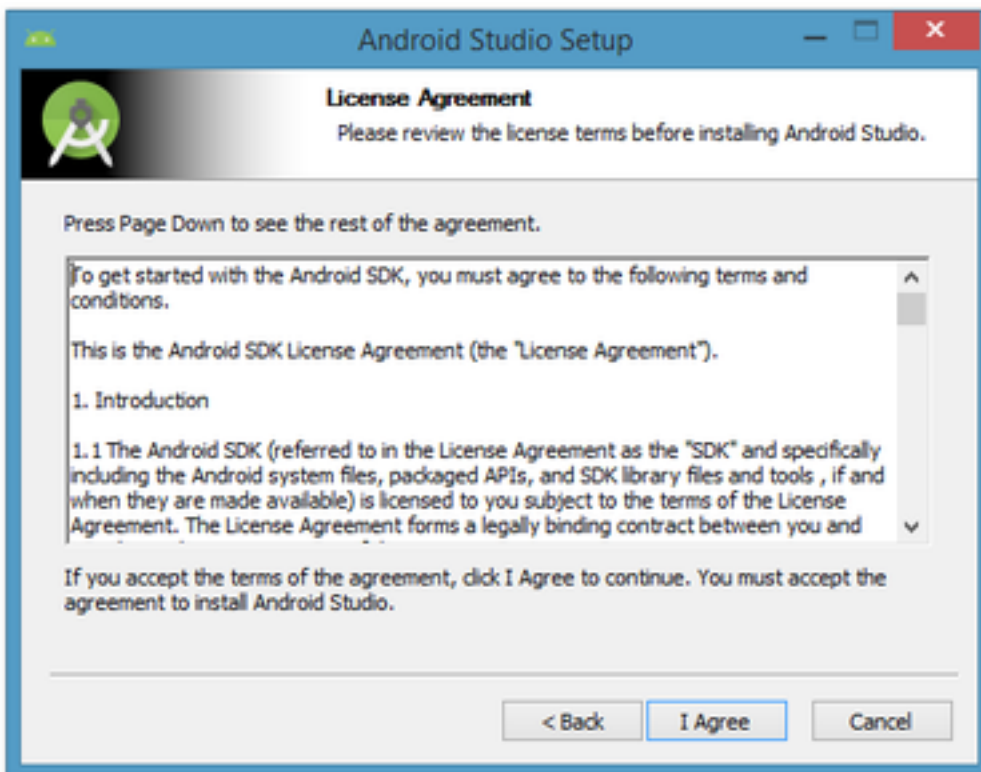
Open the downloaded Installation file to start the installation process. The installer responded by presenting the Android Studio Setup dialog box



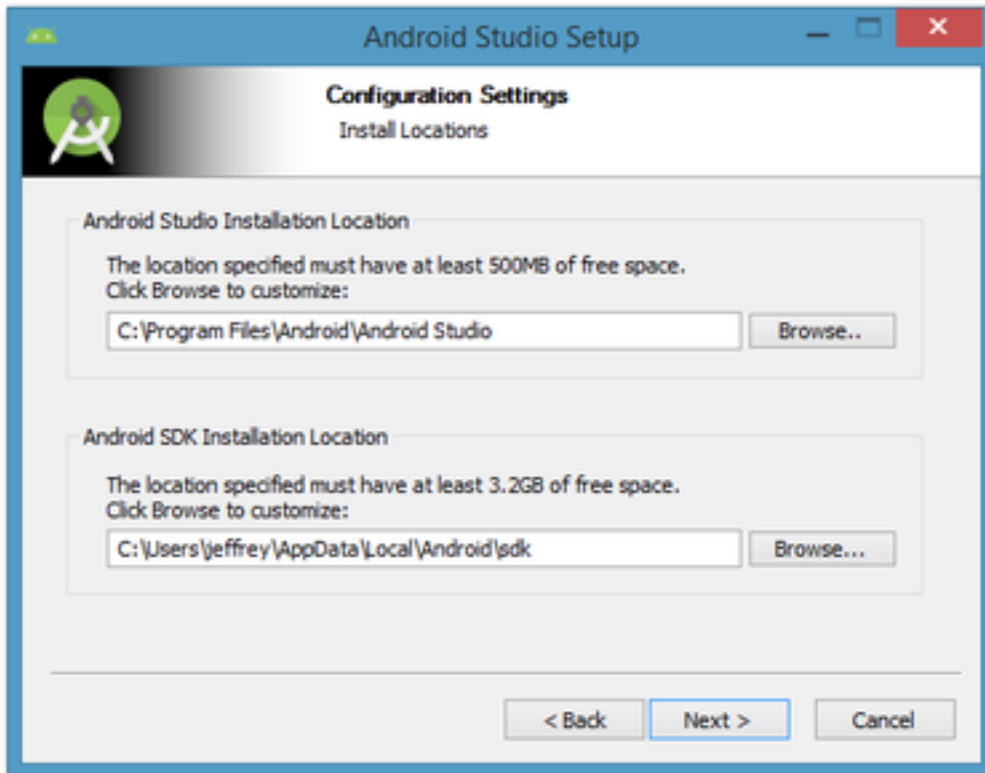
Clicking Next button to the following dialog box, which gives you the option to decline installing the Android SDK (included with the installer) and an Android Virtual Device (AVD).



You may choose to keep the default settings. After clicking Next, you'll be taken to the license agreement dialog box. Accept the license to continue the installation.

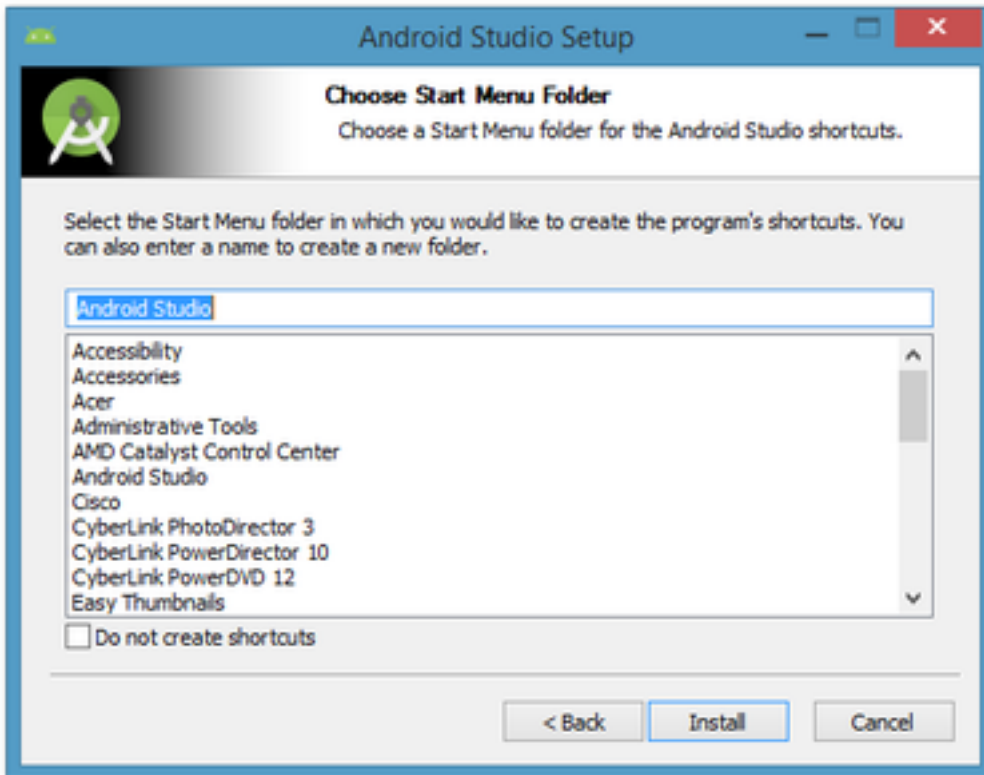


The next dialog box invites you to change the installation locations for Android Studio and the Android SDK.



Change the location or accept the default locations and click Next.

The installer defaults to creating a shortcut for launching this program, or you can choose to decline. Then click the Install button to begin installation.



The resulting dialog box shows the progress of installing Android Studio and the Android SDK. Clicking the Show Details button will let you view detailed information about the installation progress.

The dialog box will inform you when installation has finished.



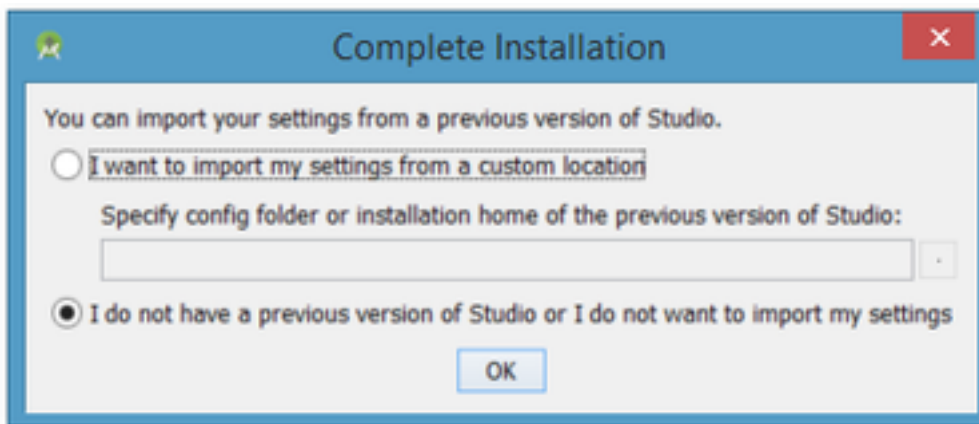
To complete your installation, leave the Start Android Studio box checked and click Finish.

Running Android Studio

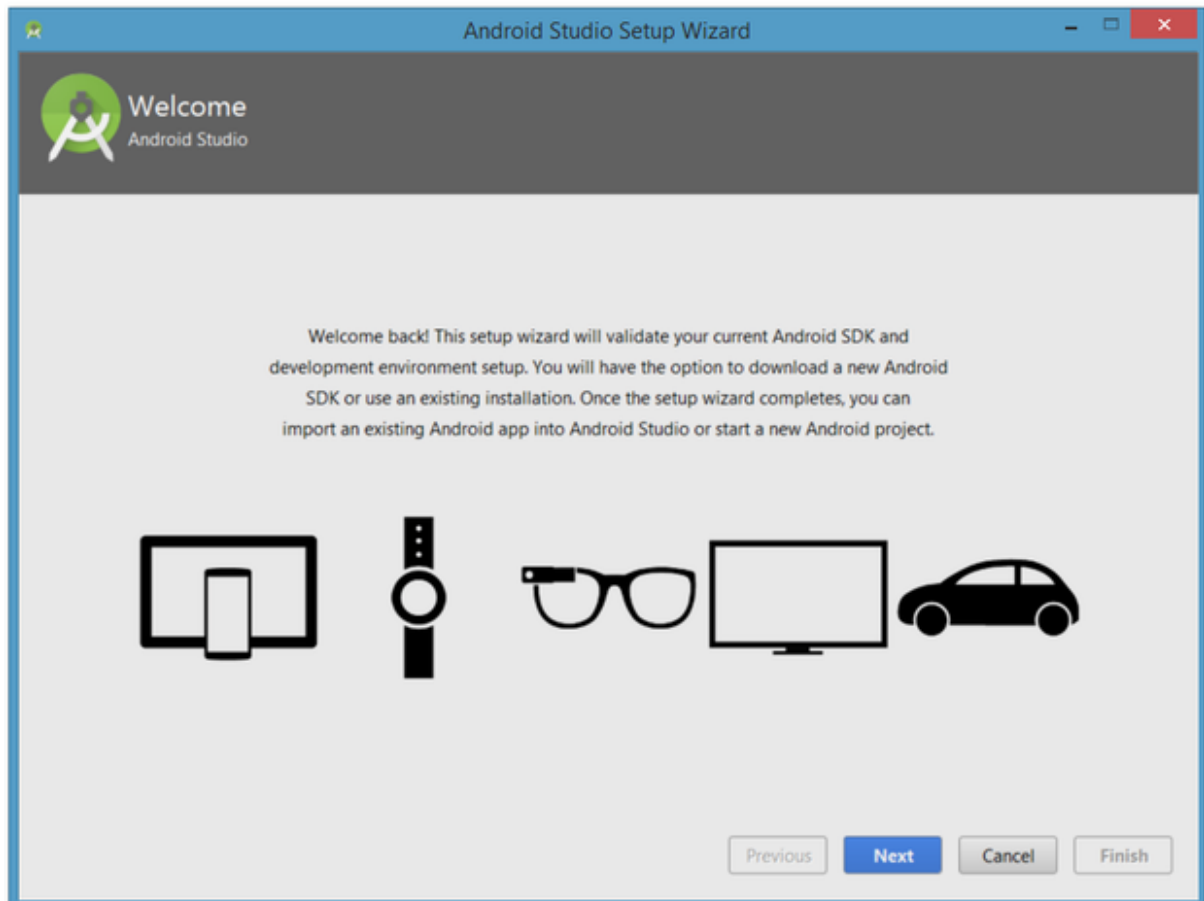
Android Studio presents a splash screen when it starts running:



On your first run, you'll be asked to respond to several configuration-oriented dialog boxes. The first dialog box focuses on importing settings from any previously installed version of Android Studio.

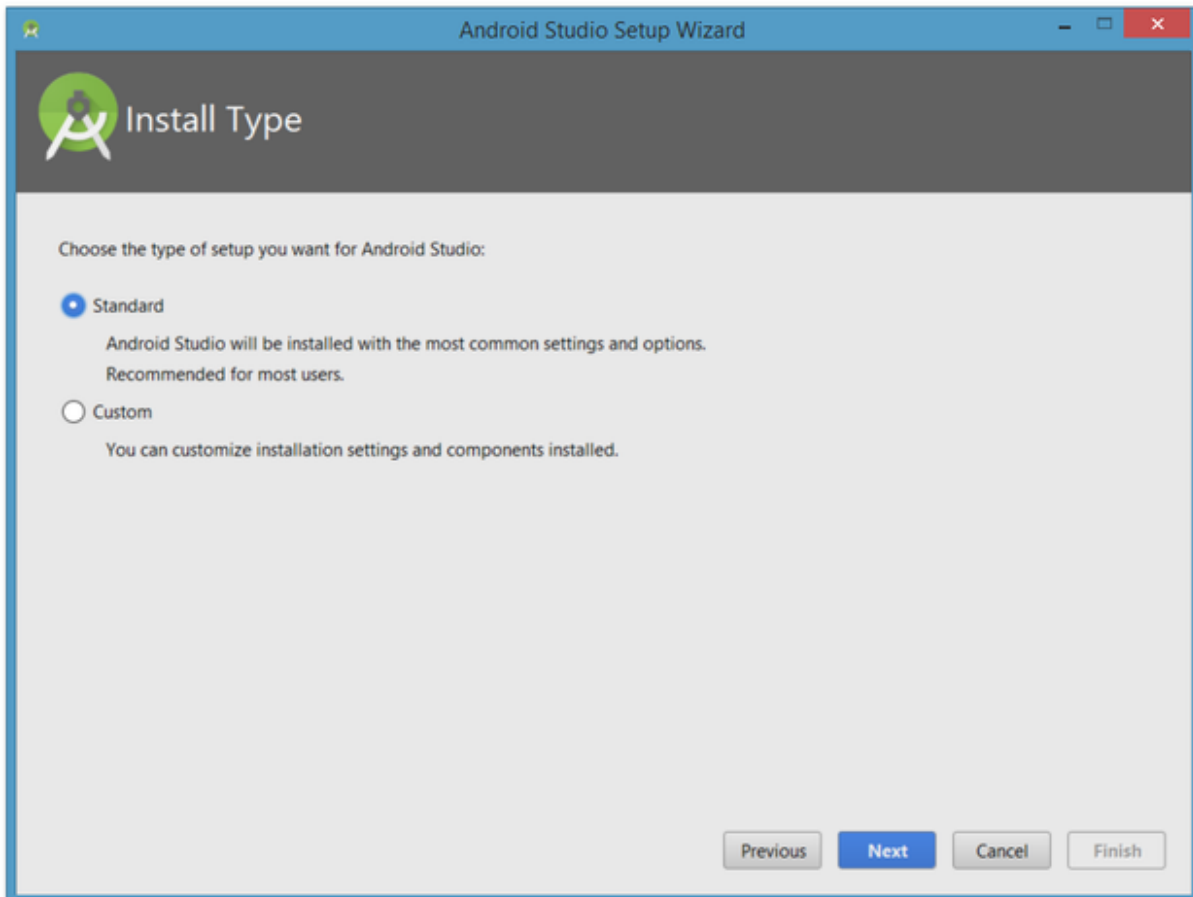


If you don't have a previously installed version, you can just keep the default setting and click OK. Android Studio will respond with a slightly enhanced version of the splash screen, followed by the Android Studio Setup Wizard dialog box:

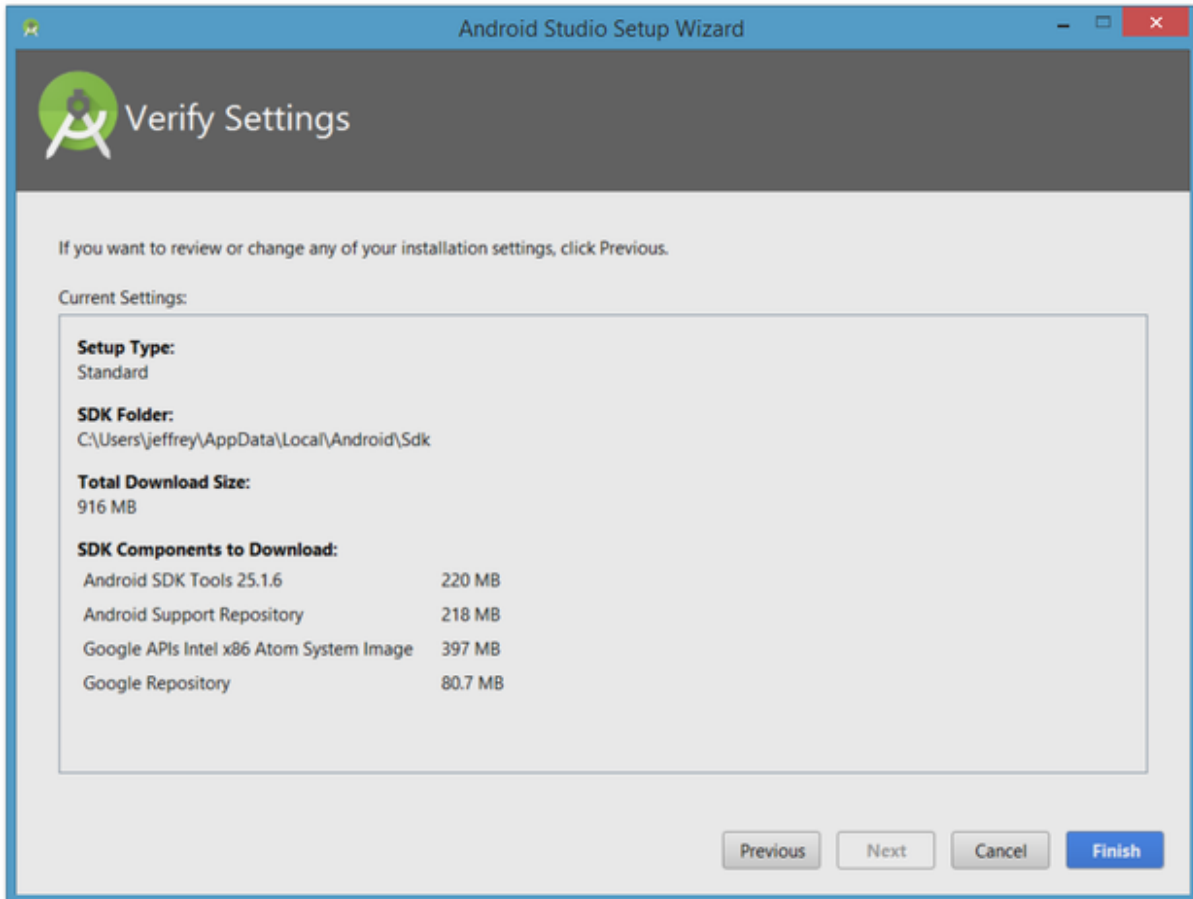


Validate your Android SDK and development environment setup

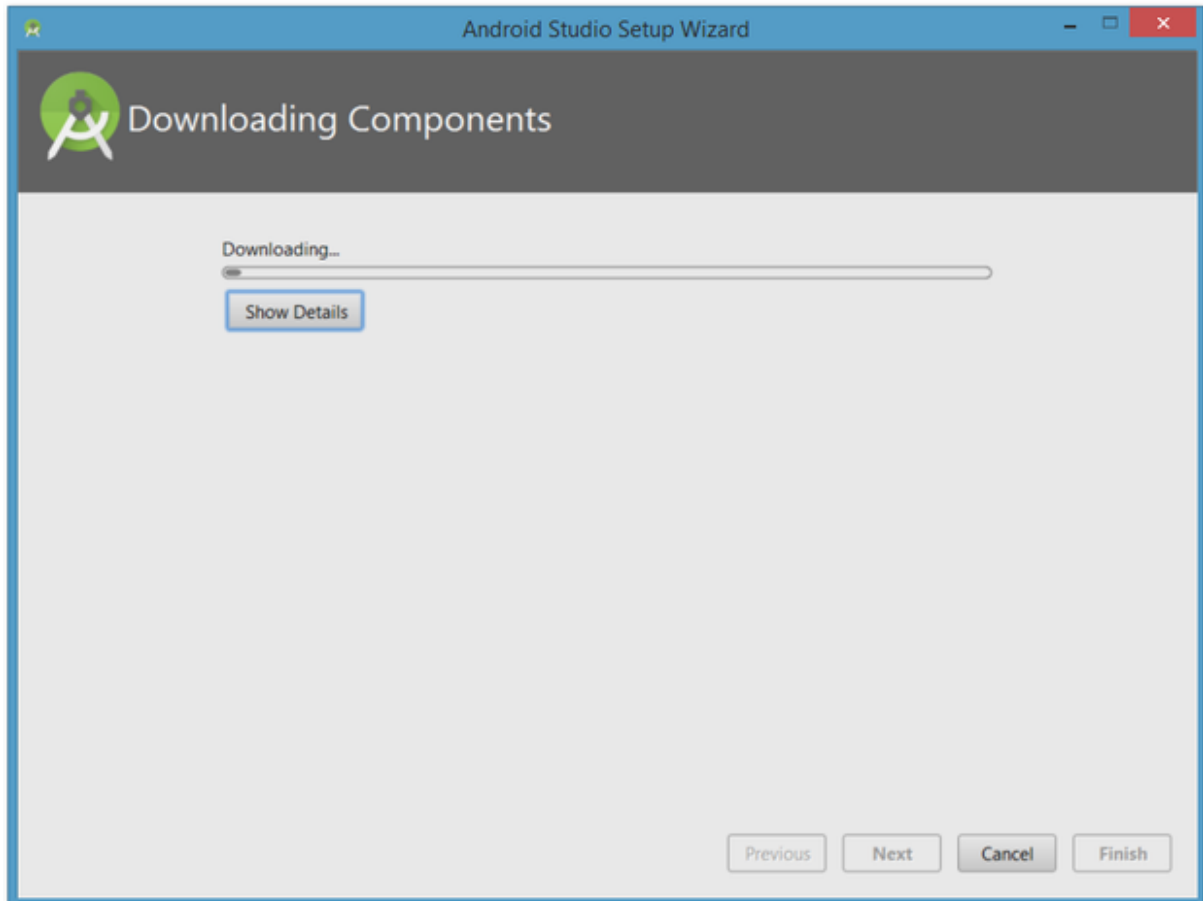
When you click Next, the setup wizard invites you to select an installation type for your SDK components. For now I recommend you keep the default standard setting.



Click Next and verify your settings, then click Finish to continue.



The wizard will download and unzip various components. Click Show Details if you want to see more information about the archives being downloaded and their contents.



Finally, click Finish to complete the wizard. You should see the Welcome to Android Studio dialog box:



Features of Android Studio

Android Studio is Android's official IDE. It is purpose-built for Android to accelerate your development and help you build the highest-quality apps for every Android device. Based on IntelliJ IDEA, Android Studio provides the fastest possible turnaround on your coding and run ning workflow.

Instant Run

Android Studio's Instant Run feature pushes code and resource changes to your running app. It intelligently understands the changes and often delivers them without restarting your app or rebuilding your APK, so you can see the effects immediately.

Intelligent code editor

The code editor helps you write better code, work faster, and be more productive by offering advanced code completion, refactoring, and code analysis. As you type, Android Studio provides suggestions in a dropdown list. Simply press Tab to insert the code.

Fast and feature-rich emulator

The Android Emulator installs and starts your apps faster than a real device and allows you to prototype and test your app on various Android device configurations: phones, tablets, Android Wear, and Android TV devices. You can also simulate a variety of hardware features such as GPS location, network latency, motion sensors, and multi-touch input.

Code templates and sample apps

Android Studio includes project and code templates that make it easy to add well-established patterns such as a navigation drawer and view pager. You can start with a code template or even right-click an API in the editor and select *Find Sample Code* to search for examples. Moreover, you can import fully functional apps from GitHub, right from the Create Project screen.

Lintelligence

Android Studio provides a robust static analysis framework and includes over 280 different lint checks across the entirety of your app. Additionally, it provides several quick fixes that help you address issues in various categories, such as performance, security, and correctness, with a single click.

Testing tools and frameworks

Android Studio provides extensive tools to help you test your Android apps with JUnit 4 and functional UI test frameworks. With Espresso Test Recorder, you can generate UI test code by recording your interactions with the app on a device or emulator. You can run your tests on a device, an emulator, a continuous integration environment, or in Firebase Test Lab

Configure builds without limits

Android Studio's project structure and Gradle-based builds provide the flexibility you need to generate APKs for all device types.

Robust and flexible build system

Android Studio offers build automation, dependency management, and customizable build configurations. You can configure your project to include local and hosted libraries, and define build variants that include different code and resources, and apply different code shrinking and app signing configurations.

Designed for teams

Android Studio integrates with version control tools, such as GitHub and Subversion, so you can keep your team in sync with project and build changes. The open source Gradle build system allows you to tailor the build to your environment and run on a continuous integration server such as Jenkins.

Optimized for all Android devices

Android Studio provides a unified environment where you can build apps for Android phones, tablets, Android Wear, Android TV, and Android Auto. Structured code modules allow you to divide your project into units of functionality that you can independently build, test, and debug.

Create rich and connected apps

Android Studio knows not all code is written in Java and not all code runs on the user's device.

C++ and NDK support

Android Studio fully supports editing C/C++ project files so you can quickly build JNI components in your app. The IDE provides syntax highlighting and refactoring for C/C++, and an LLDB-based debugger that allows you to simultaneously debug your Java and C/C++ code. The build tools can also execute your CMake and ndk-build scripts without any modification and then add the shared objects to your APK.

Firebase and Cloud integration

The Firebase Assistant helps you connect your app to Firebase and add services such as Analytics, Authentication, Notifications and more with step-by-step procedures right inside Android Studio. Built-in tools for Google Cloud Platform also help you integrate your Android app with services such as Google Cloud Endpoints and project modules specially-designed for Google App Engine.

Eliminate tiresome tasks

Android Studio provides GUI tools that simplify the less interesting parts of app development.

Layout Editor

When working with XML layout files, Android Studio provides a drag-and-drop visual editor that makes it easier than ever to create a new layout. The Layout Editor was built in unison with the ConstraintLayout API, so you can quickly build a layout that adapts to different screen sizes by dragging views into place and then adding layout constraints with just a few clicks.

APK Analyzer

You can use the APK Analyzer to easily inspect the contents of your APK. It reveals the size of each component so you can identify ways to reduce the overall APK size. It also allows you preview packaged assets, inspect the DEX files to troubleshoot multidex issues, and compare the differences between two APKs.

Vector Asset Studio

Android Studio makes it easy to create a new image asset for every density size. With Vector Asset Studio, you can select from Google-provided material design icons or import an SVG or PSD file. Vector Asset Studio can also generate bitmap files for each screen density to support older versions of Android that don't support the Android vector drawable format.

Translations Editor

The Translations Editor gives you a single view of all of your translated resources, making it easy to change or add translations, and to find missing translations without opening each version of the strings.xml file. It even provides a link to order translation services.

Android Studio is a blessing in Android development field earlier the work is done on Eclipse. Android is an Open Handset Alliance which anyone can learn and give tremendous development to the IT Industries. Android Studio is a platform, which helps to build advanced and fully developed

applications with latest features. Android is always getting upgraded with the latest technologies with very frequently new versions. Learning **professional android app development course** will boost your career in this vast developing area. Android is a fastest growing and developing language at present.

Create "Hello World" app

In this task, you will implement the "Hello World" app to verify that Android studio is correctly installed and learn the basics of developing with Android Studio.

Create the "Hello World" app

1. Launch Android Studio if it is not already opened.
2. In the main **Welcome to Android Studio** window, click "Start a new Android Studio project".
3. In the **New Project** window, give your application an **Application Name**, such as "Hello World".
4. Verify the Project location, or choose a different directory for storing your project.
5. Choose a unique **Company Domain**.
 - Apps published to the Google Play Store must have a unique package name. Since domains are unique, prepending your app's name with your or your company's domain name is going to result in a unique package name.
 - If you are not planning to publish your app, you can accept the default example domain.
6. Verify that the default **Project location** is where you want to store your Hello World app and other Android Studio projects, or change it to your preferred directory. Click Next.
7. On the **Target Android Devices** screen, "Phone and Tablet" should be selected. And select the API and is set as the **Minimum SDK**.
8. Click **Next**.
9. If your project requires additional components for your chosen target SDK, Android Studio will install them automatically. Click **Next**.
10. **Customize the Activity** window. Every app needs at least one activity. An activity represents a single screen with a user interface and Android Studio provides templates to help you get started. For the Hello World project, choose the simplest template, the "Empty Activity" project template is the simplest template available.
11. It is a common practice to call your main activity MainActivity. This is not a requirement.
12. Make sure the **Generate Layout file** box is checked.
13. Make sure the **Backwards Compatibility (App Compat)** box is checked.
14. Leave the **Layout Name** as activity_main. It is customary to name layouts after the activity they belong to. Accept the defaults and click **Finish**.

After these steps, Android Studio:

- Creates a folder for your Android Studio Projects.
- Builds your project with Gradle. Android Studio uses Gradle as it's build system.
- Opens the code editor with your project.
- Displays a tip of the day.

- Android Studio offers many keyboard shortcuts, and reading the tips is a great way to learn them over time.

Create a virtual device (emulator)

The Android Virtual Device (AVD) manager is used to create a virtual device or emulator that simulates the configuration for a particular type of Android device.

Using the AVD Manager, you define the hardware characteristics of a device and its API level, and save it as a virtual device configuration.

When you start the Android emulator, it reads a specified configuration and creates an emulated device that behaves exactly like a physical version of that device, but it resides on your computer. With virtual devices, you can test your apps on different devices (tablets, phones) with different API levels to make sure it looks good and works for most users. You do not need to depend on having a physical device available for app development.

Create a virtual device

In order to run an emulator on your computer, you have to create a configuration that describes the virtual device.


1. In Android Studio, select Tools > Android > AVD Manager, or click the AVD Manager icon



in the toolbar.

2. Click the **+Create Virtual Device**.... If you have created a virtual device before, the window shows all of your existing devices and the button is at the bottom.
3. The Select Hardware screen appears showing a list of pre configured hardware devices. For each device, the table shows its diagonal display size (Size), screen resolution in pixels (Resolution), and pixel density (Density).
4. For the Nexus 5 device, the pixel density is xxhdpi, which means your app uses the launcher icons in the xxhdpi folder of the mipmap folder. Likewise, your app will use layouts and drawables from folders defined for that density as well.
5. Choose the Nexus 5 hardware device and click **Next**.
6. On the **System Image** screen, from the **Recommended** tab, choose which version of the Android system to run on the virtual device. You can select the latest system image.
7. There are many more versions available than shown in the **Recommended** tab. Look at the **x86 Images** and **Other Images** tabs to see them.
8. If a **Download** link is visible next to a system image version, it is not installed yet, and you need to download it. If necessary, click the link to start the download, and click **Finish** when it's done.
9. On **System Image** screen, choose a system image and click **Next**.
10. Verify your configuration, and click **Finish**.

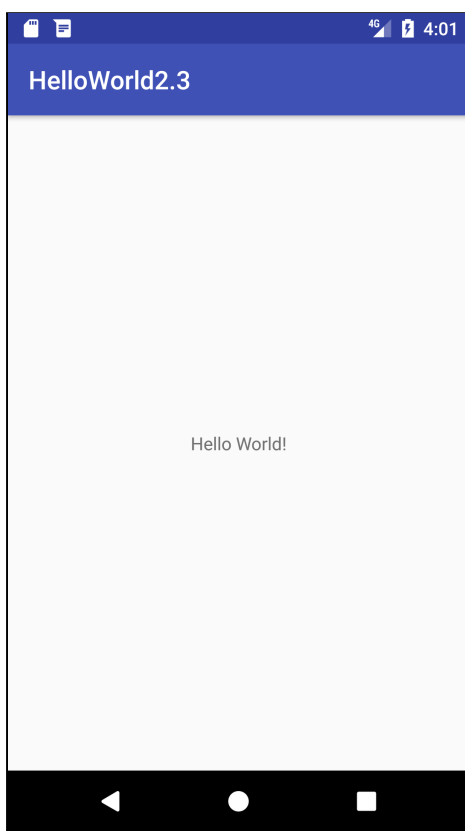
Run your app on an emulator

1. In Android Studio, select **Run > Run app** or click the **Run icon**  in the toolbar.
2. In the **Select Deployment Target** window, under **Available Emulators**, select **Nexus 5 API 23** and click **OK**.

The emulator starts and boots just like a physical device. Depending on the speed of your computer, this may take a while. Your app builds, and once the emulator is ready, Android Studio will upload the app to the emulator and run it.

You should see the Hello World app as shown in the following screenshot.

The project structure



In the Project > Android view of your previous task, there are three top-level folders below your **app** folder: **manifests**, **java**, and **res**.

1. The **manifests** folder.

This folder contains **AndroidManifest.xml**. This file describes all of the components of your Android app and is read by the Android run-time system when your program is executed.

2. The **java** folder. All your Java language files are organized in this folder. The **java** folder contains three subfolders:

- **com.example.hello.helloworld (or the domain name you have specified)**: All the files for a package are in a folder named after the package. For your Hello World application, there is one package and it only contains MainActivity.java

- **com.example.hello.helloworld(androidTest)**: This folder is for your instrumented tests, and starts out with a skeleton test file.
 - **com.example.hello.helloworld(test)**: This folder is for your unit tests and starts out with an automatically created skeleton unit test file.
3. The **res** folder. This folder contains all the resources for your app, including images, layout files, strings, icons, and styling. It includes these subfolders:
 - **drawable**. Store all your app's images in this folder.
 - **layout**. Every activity has at least one layout file that describes the UI in XML. For Hello World, this folder contains activity_main.xml.
 - **mipmap**. Store your launcher icons in this folder. There is a sub-folder for each supported screen density. Android uses the screen density, that is, the number of pixels per inch to determine the required image resolution. Android groups all actual

screen densities into generalized densities, such as medium (mdpi), high (hdpi), or extra-extra-extra-high (xxxhdpi). The `ic_launcher.png` folder contains the default launcher icons for all the densities supported by your app.

- **values.** Instead of hardcoding values like strings, dimensions, and colors in your XML and Java files, it is best practice to define them in their respective values file. This makes it easier to change and be consistent across your app.
4. The **values** subfolder within the `res` folder. It includes these subfolders:
- **colors.xml.** Shows the default colors for your chosen theme, and you can add your own colors or change them based on your app's requirements.
 - **dimens.xml.** Store the sizes of views and objects for different resolutions.
 - **strings.xml.** Create resources for all your strings. This makes it easy to translate them to other languages.
 - **styles.xml.** All the styles for your app and theme go here. Styles help give your app a consistent look for all UI elements.

Android Virtual Device (AVD)

An Android Virtual Device (AVD) is a configuration that defines the characteristics of an Android phone, tablet, Wear OS, or Android TV device that you want to simulate in the Android Emulator. The Android Emulator simulates Android devices on your computer so that you can test your application on a variety of devices and Android API levels without needing to have each physical device. The emulator provides almost all of the capabilities of a real Android device. You can simulate incoming phone calls and text messages, specify the location of the device, simulate different network speeds, simulate rotation and other hardware sensors, access the Google Play Store, and much more. Testing your app on the emulator is in some ways faster and easier than doing so on a physical device.

Each instance of the Android Emulator uses an Android virtual device (AVD) to specify the Android version and hardware characteristics of the simulated device. To effectively test your app, you should create an AVD that models each device on which your app is designed to run. To create and manage AVDs, use the AVD Manager.

Each AVD functions as an independent device, with its own private storage for user data, SD card, and so on. By default, the emulator stores the user data, SD card data, and cache in a directory specific to that AVD. When you launch the emulator, it loads the user data and SD card data from the AVD directory.

The AVD Manager is an interface you can launch from Android Studio that helps you create and manage AVDs.

To open the AVD Manager, do one of the following:

Select **Tools > AVD Manager**.

Click **AVD Manager** icon in the toolbar.



About AVDs

An AVD contains a hardware profile, system image, storage area, skin, and other properties.

Hardware profile

The hardware profile defines the characteristics of a device as shipped from the factory. The AVD Manager comes preloaded with certain hardware profiles, such as Pixel devices, and you can define or customize the hardware profiles as needed.

System images

A system image, that the emulator uses to simulate the operating system. labeled with Google APIs includes access to Google Play services.

Storage area

The AVD has a dedicated storage area on your development machine. It stores the device user data, such as installed apps and settings, as well as an emulated SD card.

Skin

An emulator skin specifies the appearance of a device. The AVD Manager provides some predefined skins. You can also define your own, or use skins provided by third parties.

AVD and app features

Be sure your AVD definition includes the device features your app depends on. See Hardware Profile Properties and AVD Properties for lists of features you can define in your AVDs.

You can use the emulator manually through its graphical user interface and programmatically through the command line and the emulator console.

Hardware and software requirements

The Android Emulator has additional requirements beyond the basic system requirements for Android Studio:

- SDK Tools 26.1.1 or higher

- 64-bit processor

- Windows: CPU with UG (unrestricted guest) support

- HAXM 6.2.1 or later (HAXM 7.2.0 or later recommended)

The use of hardware acceleration has additional requirements on Windows and Linux:

Intel processor on Windows or Linux: Intel processor with support for Intel VT-x, Intel EM64T (Intel 64), and Execute Disable (XD) Bit functionality

AMD processor on Linux: AMD processor with support for AMD Virtualization (AMD-V) and Supplemental Streaming SIMD Extensions 3 (SSSE3)

AMD processor on Windows: Android Studio 3.2 or higher and Windows 10 April 2018 release or higher for Windows Hypervisor Platform (WHPX) functionality

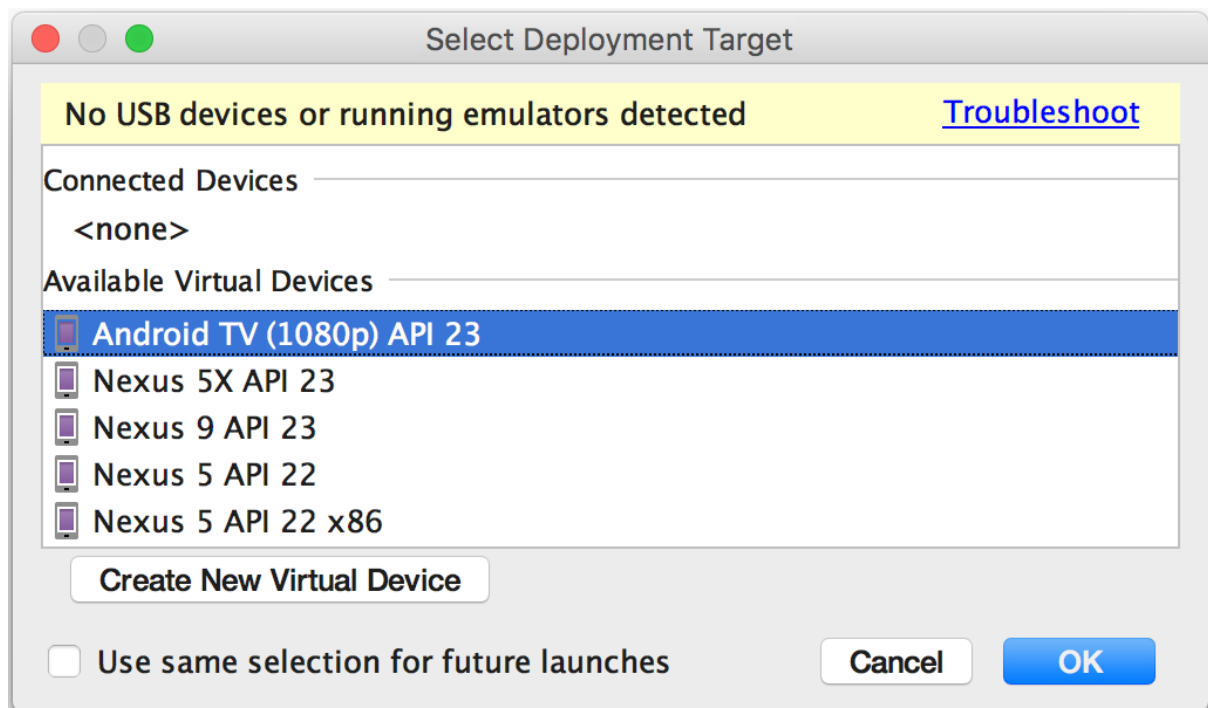
Run an app on the Android Emulator

You can run an app from an Android Studio project, or you can run an app that's been installed on the Android Emulator as you would run any app on a device.

To start the Android Emulator and run an app in your project:

Open an Android Studio project and click Run .

The Select Deployment Target dialog appears.



Select Deployment Target dialog

If you receive an error or warning message at the top of the dialog, click the link to correct the problem or get more information.

Some errors you must fix before you can continue, such as certain Hardware Accelerated Execution Manager (Intel HAXM) errors.

Launch the Android Emulator without first running an app

To start the emulator:

Open the AVD Manager.

Double-click an AVD, or click Run .

The Android Emulator appears.

While the emulator is running, you can run Android Studio projects and choose the emulator as the target device. You can also drag one or more APKs onto the emulator to install them, and then run them.

Create and manage virtual devices

To open the AVD Manager, do one of the following:

Select Tools > AVD Manager.

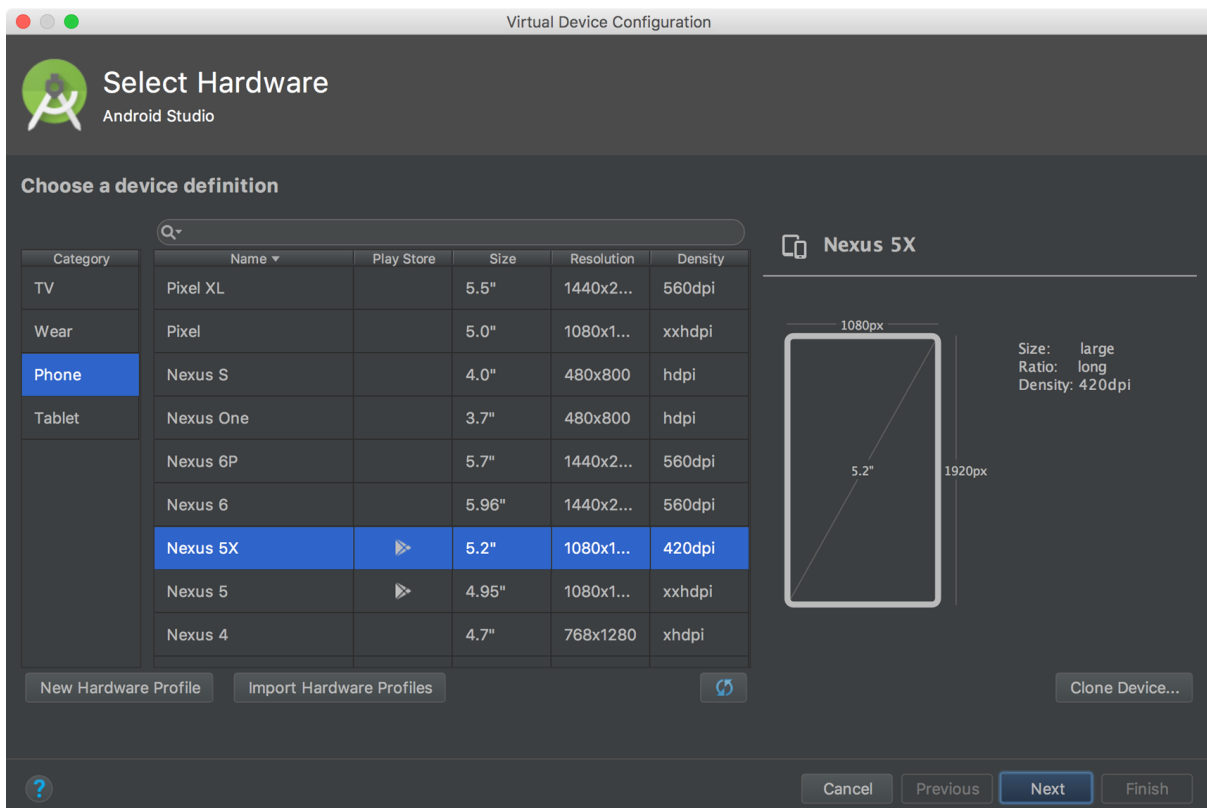
Click AVD Manager icon in the toolbar.

Create an AVD

Click Create Virtual Device, at the bottom of the AVD Manager dialog.



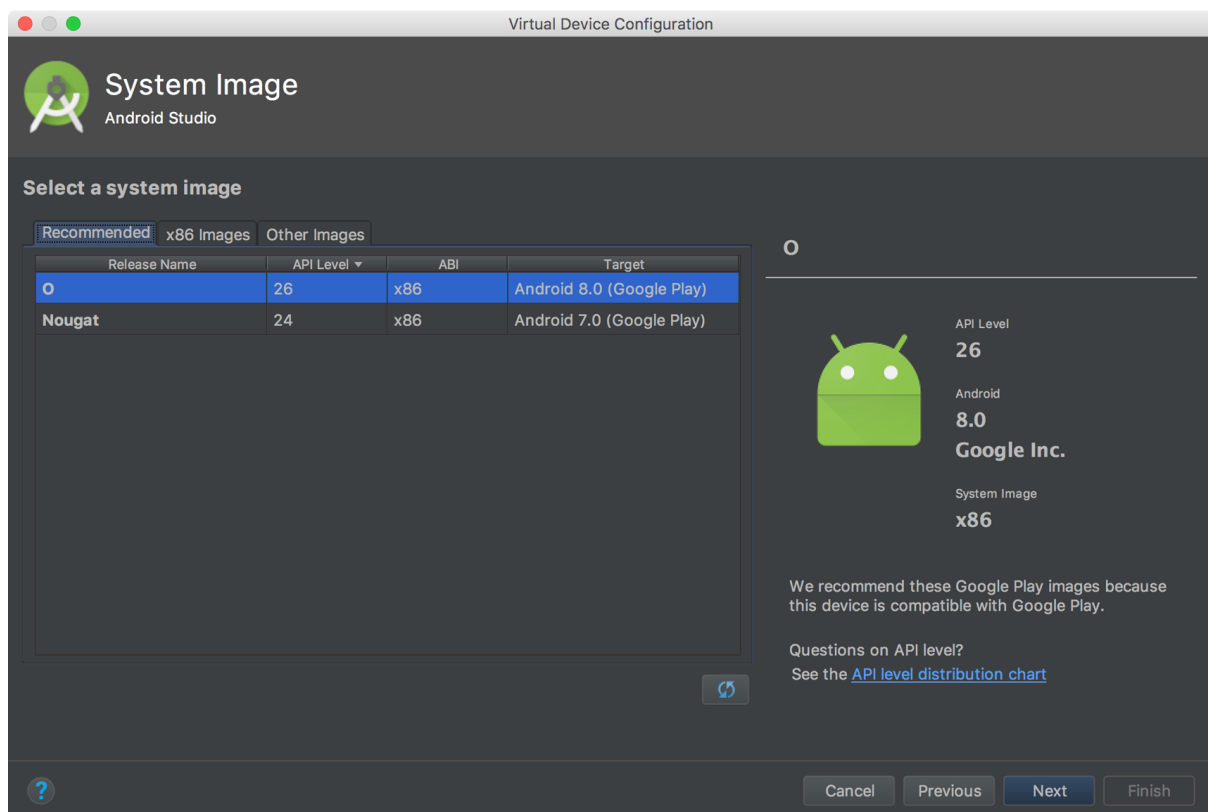
The Select Hardware page appears.



Select a hardware profile, and then click Next.

If you don't see the hardware profile you want, you can create or import a hardware profile.

The System Image page appears.



Select the system image for a particular API level, and then click Next.

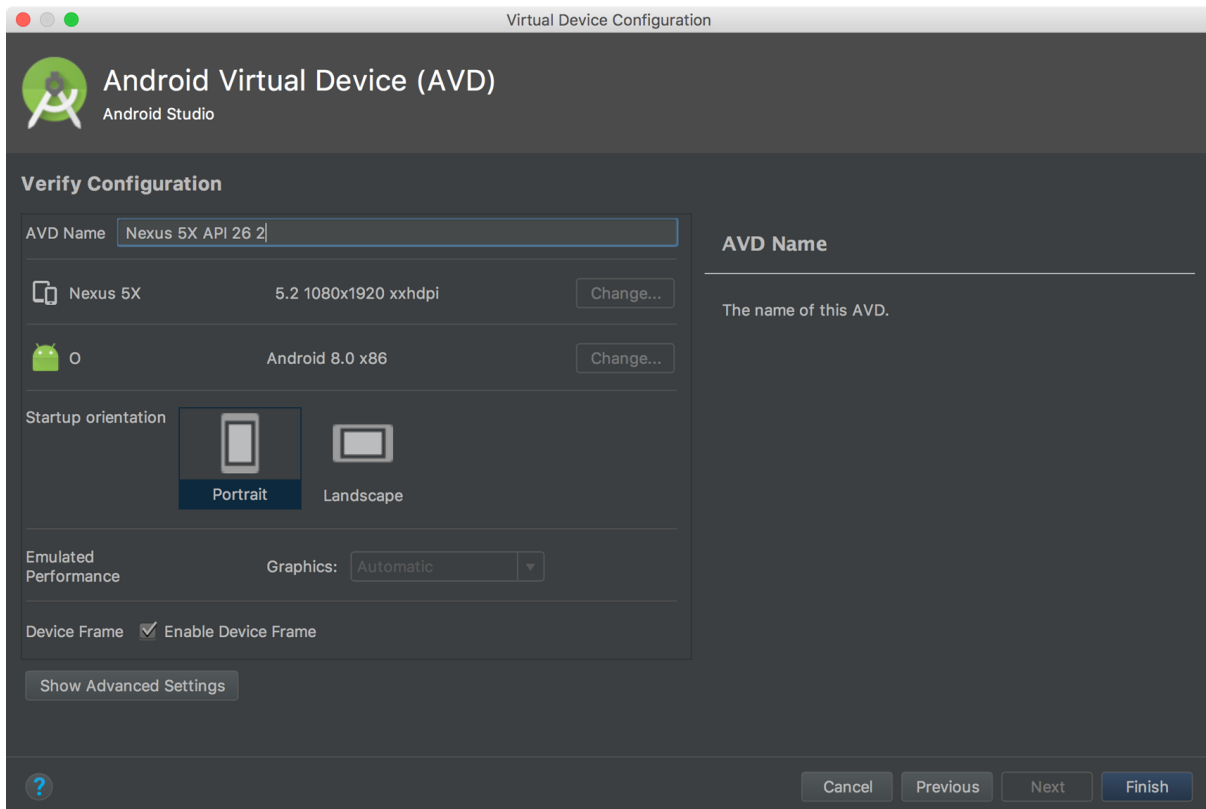
The Recommended tab lists recommended system images. The other tabs include a more complete list. The right pane describes the selected system image. x86 images run the fastest in the emulator.

If you see Download next to the system image, you need to click it to download the system image. You must be connected to the internet to download it.

The API level of the target device is important, because your app won't be able to run on a system image with an API level that's less than that required by your app, as specified in the minSdkVersion attribute of the app manifest file.

If your app declares a <uses-library> element in the manifest file, the app requires a system image in which that external library is present. If you want to run your app on an emulator, create an AVD that includes the required library. To do so, you might need to use an add-on component for the AVD platform; for example, the Google APIs add-on contains the Google Maps library.

The Verify Configuration page appears.



Change AVD properties as needed, and then click Finish.

Click Show Advanced Settings to show more settings, such as the skin.

The new AVD appears in the Your Virtual Devices page or the Select Deployment Target dialog

Limitations

The Android Emulator doesn't include virtual hardware for the following:

1. Bluetooth
2. NFC
3. SD card insert/eject
4. Device-attached headphones
5. USB

The watch emulator for Wear OS doesn't provide the Overview (Recent Apps) button, D-pad, and fingerprint sensor

App Manifest Overview

Every app project must have an AndroidManifest.xml file at the root of the project source set. The manifest file describes essential information about your app to the Android build tools, the Android operating system, and Google Play.

Among many other things, the manifest file is required to declare the following:

The app's package name, which usually matches your code's namespace. The Android build tools use this to determine the location of code entities when building your project.

The components of the app, which include all activities, services, broadcast receivers, and content providers. Each component must define basic properties such as the name of its Java

class. It can also declare capabilities such as which device configurations it can handle, and intent filters that describe how the component can be started.

The permissions that the app needs in order to access protected parts of the system or other apps.

The hardware and software features the app requires, which affects which devices can install the app from Google Play.

If you're using Android Studio to build your app, the manifest file is created for you, and most of the essential manifest elements are added as you build your app

The XML below is a simple example AndroidManifest.xml that declares two activities for the app.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="1"
    android:versionName="1.0"
    package="com.example.myapp">
    <uses-sdk android:minSdkVersion="15" android:targetSdkVersion="26" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <!-- This name is resolved to com.example.myapp.MainActivity
        based upon the package attribute -->
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".DisplayMessageActivity"
            android:parentActivityName=".MainActivity" />
    </application>
</manifest>
```

The following sections describe how some of the most important characteristics of your app are reflected in the manifest file.

The manifest file's root element requires an attribute for your app's package name.

For example, the following snippet shows the root <manifest> element with the package name "com.example.myapp":

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapplication"
    android:versionCode="1"
    android:versionName="1.0" >
    ...
</manifest>

```

As such, the name in the manifest's package attribute should always match your project's base package name where you keep your activities and other app code.

App components

For each app component that you create in your app, you must declare a corresponding XML element in the manifest file:

- <activity> for each subclass of Activity.
- <service> for each subclass of Service.
- <receiver> for each subclass of BroadcastReceiver.
- <provider> for each subclass of ContentProvider.

If you subclass any of these components without declaring it in the manifest file, the system cannot start it.

The name of your subclass must be specified with the name attribute, using the full package designation. For example

```

<manifest ... >
    <application ... >
        <activity android:name="com.example.myapplication.MainActivity" ... >
        </activity>
    </application>
</manifest>

```

Intent filters

App activities, services, and broadcast receivers are activated by intents. An intent is a message defined by an Intent object that describes an action to perform, including the data to be acted upon, the category of component that should perform the action, and other instructions.

When an app issues an intent to the system, the system locates an app component that can handle the intent based on intent filter declarations in each app's manifest file. The system launches an instance of the matching component and passes the Intent object to that component. An app component can have any number of intent filters (defined with the <intent-filter> element), each one describing a different capability of that component.

Icons and labels

A number of manifest elements have icon and label attributes for displaying a small icon and a text label, respectively, to users for the corresponding app component.

In every case, the icon and label that are set in a parent element become the default icon and label value for all child elements.

Permissions

Android apps must request permission to access sensitive user data (such as contacts and SMS) or certain system features (such as the camera and internet access). Each permission is identified by a unique label. For example, an app that needs to send SMS messages must have the following line in the manifest:

```
<manifest ... >  
    <uses-permission android:name="android.permission.SEND_SMS"/>  
    ...  
</manifest>
```

Beginning with Android 6.0 (API level 23), the user can approve or reject some app permissions at runtime. But no matter which Android version your app supports, you must declare all permission requests with a `<uses-permission>` element in the manifest

Device compatibility

The manifest file is also where you can declare what types of hardware or software features your app requires, and thus, which types of devices your app is compatible with.

The following are just a couple of the most common tags.

```
<uses-feature>
```

The `<uses-feature>` element allows you to declare hardware and software features your app needs.

```
<uses-sdk>
```

Each successive platform version often adds new APIs not available in the previous version. To indicate the minimum version with which your app is compatible, your manifest must include the `<uses-sdk>` tag and its `minSdkVersion` attribute.