ANDROID APPLICATION DEVELOPMENT

Course Code-	**Theory Course**	**L-T-P-C**	**4-0-0-4**

Course Contents

UNIT-I	**Introduction to Android :** Overview, History,. Features of Android, Architecture of Android. Android Phones, SDK. Android Development Tools. Android Emulator. Creating Android Virtual Device. Creating your first Android Application
UNIT-II	**Activities:** Introduction, Activity Lifecycle. **Intents:** Introduction. Linking Activities using Intents. Calling built-in applications using Intents, **Fragments:** Introduction. Adding Fragments Dynamically. Lifecycle of Fragment. Interaction between fragments
UNIT-III	**Android User Interface:** Understanding the components of a screen. Display Orientation **Designing Your User Interface with Views:** Basic Views. Picker Views List View Specialized Fragment, Displaying Pictures and Menes with views
UNIT-IV	**Databases - SQLite:** Introduction, Creating. Opening and Closing Database Working with Cursors, Insert, Update, Delete, Building and Executing Queries
UNIT-V	**Messaging and E-mail:** SMS Messaging and Send Email **Developing Android Services**: Creating Services Communication between a Service and an Activity, Binding Activities to Services **Publishing Android Application**: Preparing for abusing. Deploying APK Files

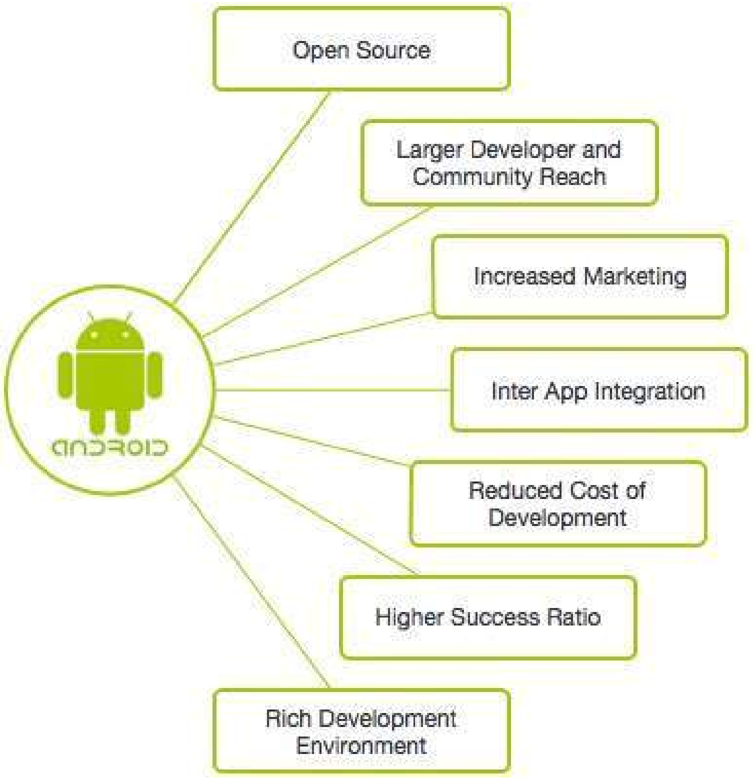
Android is an open source and Linux-based **Operating System** for mobile devices such as smartphones and tablet computers. Android was developed by the *Open Handset Alliance*, led by Google, and other companies.

Android offers a unified approach to application development for mobile devices which means developers need only develop for Android, and their applications should be able to run on different devices powered by Android.

The first beta version of the Android Software Development Kit (SDK) was released by Google in 2007 where as the first commercial version, Android 1.0, was released in September 2008.

On June 27, 2012, at the Google I/O conference, Google announced the next Android version, 4.1 **Jelly Bean**. Jelly Bean is an incremental update, with the primary aim of improving the user interface, both in terms of functionality and performance.

The source code for Android is available under free and open source software licenses. Google publishes most of the code under the Apache License version 2.0 and the rest, Linux kernel changes, under the GNU General Public License version 2.


Open Source
Larger Developer and Community Reach
Increased Marketing
Inter App Integration
Reduced Cost of Development
Higher Success Ratio
Rich Development Environment
android

Features of Android

Android is a powerful operating system competing with Apple 4GS and supports great features. Few of them are listed below −

Sr.No.	Feature & Description
1	**Beautiful UI** Android OS basic screen provides a beautiful and intuitive user interface.
2	**Connectivity** GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX.
3	**Storage** SQLite, a lightweight relational database, is used for data storage purposes.
4	**Media support** H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, AAC 5.1, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP.
5	**Messaging** SMS and MMS
6	**Web browser** Based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine supporting HTML5 and CSS3.
7	**Multi-touch** Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero.
8	**Multi-tasking** User can jump from one task to another and same time various application can run simultaneously.
9	**Resizable widgets** Widgets are resizable, so users can expand them to show more content or shrink them to save space.

10	**Multi-Language** Supports single direction and bi-directional text.
11	**GCM** Google Cloud Messaging (GCM) is a service that lets developers send short message data to their users on Android devices, without needing a proprietary sync solution.
12	**Wi-Fi Direct** A technology that lets apps discover and pair directly, over a high-bandwidth peer-to-peer connection.
13	**Android Beam** A popular NFC-based technology that lets users instantly share, just by touching two NFC-enabled phones together.

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**, **SlideME**, **Opera Mobile Store**, **Mobango**, **F-droid** and 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 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.

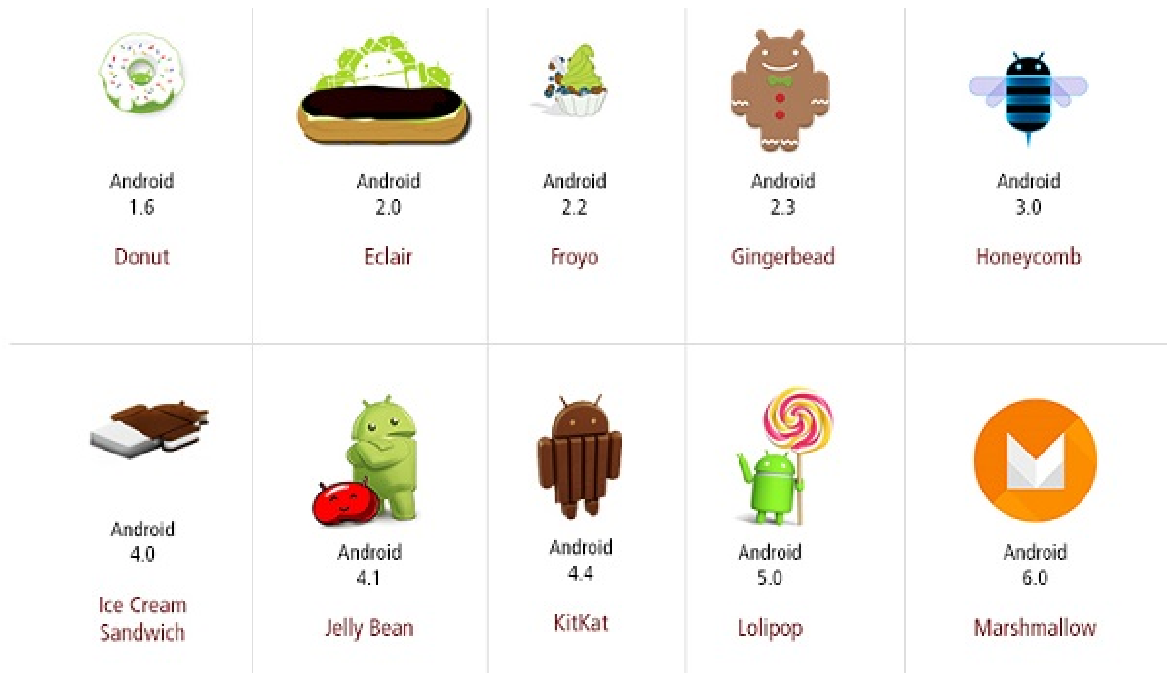
Categories of Android applications

There are many android applications in the market. The top categories are −

Music	News	Multimedia
Sports	Lifestyle	Food & Drink
Travel	Weather	Books
Business	Reference	Navigation
Social Media	Utilities	Finance

History of Android

The code names of android ranges from A to N currently, such as Aestro, Blender, Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwitch, Jelly Bean, KitKat, Lollipop and Marshmallow. Let's understand the android history in a sequence.



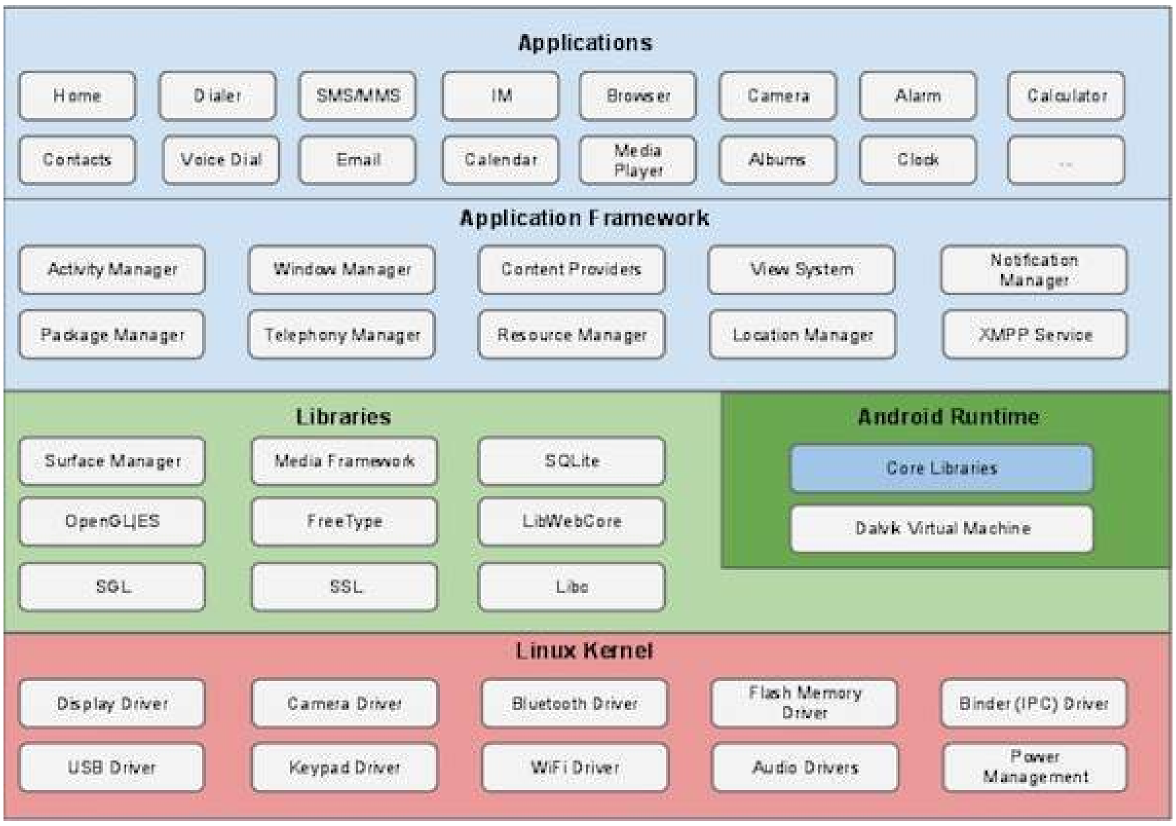
What is API level?

API Level is an integer value that uniquely identifies the framework API revision offered by a version of the Android platform.

Platform Version	**API Level**	**VERSION_CODE**	
Android 6.0	23	MARSHMALLOW	
Android 5.1	22	LOLLIPOP_MR1	
Android 5.0	21	LOLLIPOP	
Android 4.4W	20	KITKAT_WATCH	KitKat for Wearables Only
Android 4.4	19	KITKAT	
Android 4.3	18	JELLY_BEAN_MR2	

Android 4.2, 4.2.2	17	JELLY_BEAN_MR1	
Android 4.1, 4.1.1	16	JELLY_BEAN	
Android 4.0.3, 4.0.4	15	ICE_CREAM_SANDWICH_MR1	
Android 4.0, 4.0.1, 4.0.2	14	ICE_CREAM_SANDWICH	
Android 3.2	13	HONEYCOMB_MR2	
Android 3.1.x	12	HONEYCOMB_MR1	
Android 3.0.x	11	HONEYCOMB	
Android 2.3.4 Android 2.3.3	10	GINGERBREAD_MR1	
Android 2.3.2 Android 2.3.1 Android 2.3	9	GINGERBREAD	
Android 2.2.x	8	FROYO	
Android 2.1.x	7	ECLAIR_MR1	
Android 2.0.1	6	ECLAIR_0_1	
Android 2.0	5	ECLAIR	
Android 1.6	4	DONUT	
Android 1.5	3	CUPCAKE	
Android 1.1	2	BASE_1_1	
Android 1.0	1	BASE	

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

At the bottom of the layers is Linux - Linux 3.6 with approximately 115 patches. This provides a level of abstraction between the device hardware and it contains all the essential hardware drivers 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 including open-source Web browser engine WebKit, well known library libc, SQLite database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

Android Libraries

This category encompasses those Java-based libraries that are specific to Android development. Examples of libraries in this category include the application framework libraries in addition to those that facilitate user interface building, graphics drawing and database access. A summary of some key core Android libraries available to the Android developer is as follows −

- **android.app** − Provides access to the application model and is the cornerstone of all Android applications.
- **android.content** − Facilitates content access, publishing and messaging between applications and application components.
- **android.database** − Used to access data published by content providers and includes SQLite database management classes.
- **android.opengl** − A Java interface to the OpenGL ES 3D graphics rendering API.

- **android.os** − Provides applications with access to standard operating system services including messages, system services and inter-process communication.
- **android.text** − Used to render and manipulate text on a device display.
- **android.view** − The fundamental building blocks of application user interfaces.
- **android.widget** − A rich collection of pre-built user interface components such as buttons, labels, list views, layout managers, radio buttons etc.
- **android.webkit** − A set of classes intended to allow web-browsing capabilities to be built into applications.

Having covered the Java-based core libraries in the Android runtime, it is now time to turn our attention to the C/C++ based libraries contained in this layer of the Android software stack.

Android Runtime

This is the third section of the architecture and available on the second layer from the bottom. This section provides a key component called **Dalvik Virtual Machine** which is a kind of Java Virtual Machine specially designed and optimized for Android.

The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine.

The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

Application Framework

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.

The Android framework includes the following key services −

- **Activity Manager** − Controls all aspects of the application lifecycle and activity stack.
- **Content Providers** − Allows applications to publish and share data with other applications.
- **Resource Manager** − Provides access to non-code embedded resources such as strings, color settings and user interface layouts.
- **Notifications Manager** − Allows applications to display alerts and notifications to the user.
- **View System** − An extensible set of views used to create application user interfaces.

Applications

You will find all the Android application at the top layer. You will write your application to be installed on this layer only. Examples of such applications are Contacts Books, Browser, Games etc.

Application components are the essential building blocks of an Android application. These components are loosely coupled by the application manifest file *AndroidManifest.xml* that describes each component of the application and how they interact.

There are following four main components that can be used within an Android application −

Sr.No	Components & Description
1	**Activities** They dictate the UI and handle the user interaction to the smart phone screen.
2	**Services** They handle background processing associated with an application.
3	**Broadcast Receivers** They handle communication between Android OS and applications.
4	**Content Providers** They handle data and database management issues.

Activities

An activity represents a single screen with a user interface,in-short Activity performs actions on the screen. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched.

An activity is implemented as a subclass of **Activity** class as follows −

```
public class MainActivity extends Activity {
}
```

Services

A service is a component that runs in the background to perform long-running operations. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.

A service is implemented as a subclass of **Service** class as follows −

```
public class MyService extends Service {
}
```

Broadcast Receivers

Broadcast Receivers simply respond to broadcast messages from other applications or from the system. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.

A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class and each message is broadcaster as an **Intent** object.

```
public class MyReceiver  extends  BroadcastReceiver {
   public void onReceive(context,intent){}
}
```

Content Providers

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the *ContentResolver* class. The data may be stored in the file system, the database or somewhere else entirely.

A content provider is implemented as a subclass of **ContentProvider** class and must implement a standard set of APIs that enable other applications to perform transactions.

```
public class MyContentProvider extends  ContentProvider {
   public void onCreate(){}
}
```

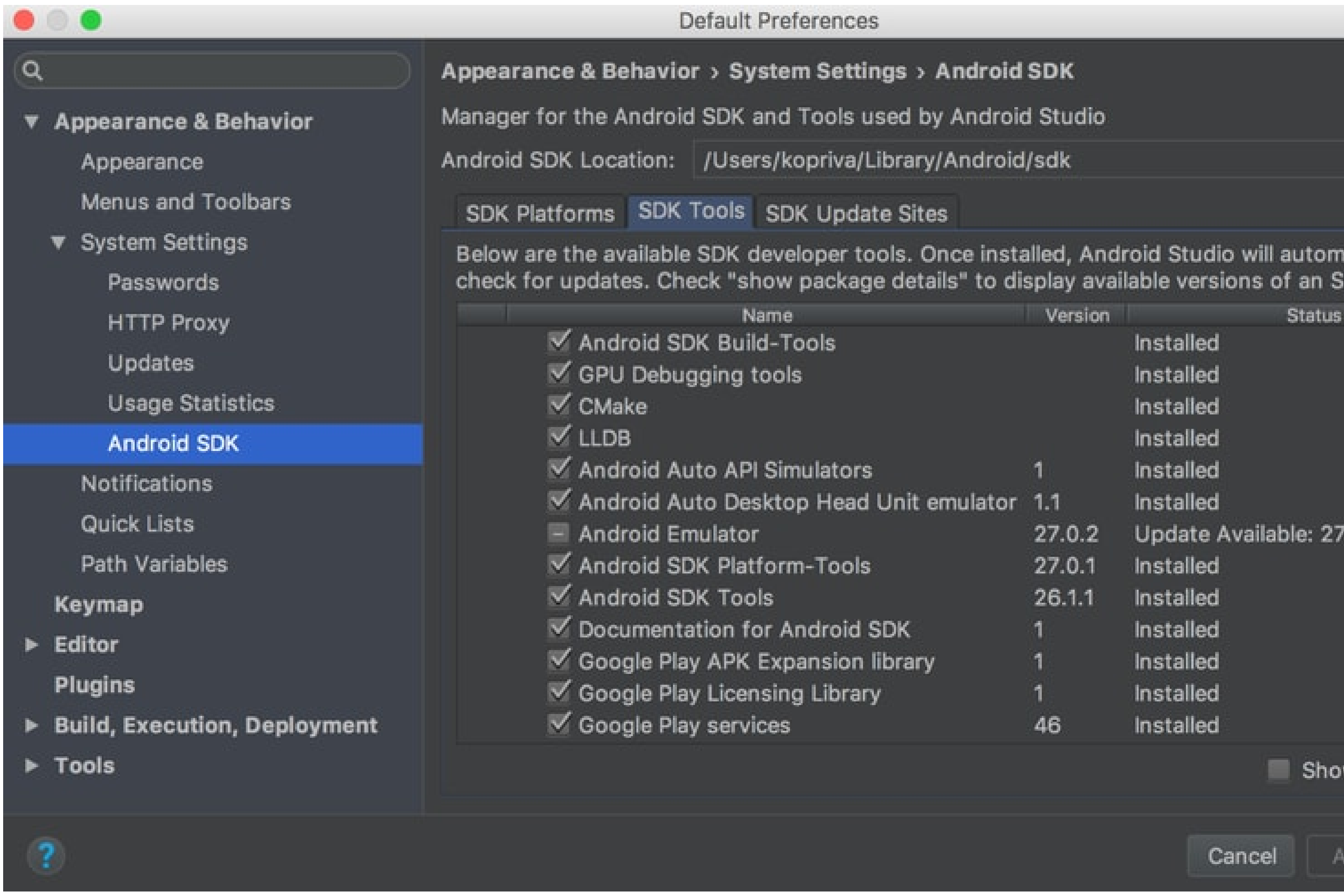
We will go through these tags in detail while covering application components in individual chapters.

Additional Components

There are additional components which will be used in the construction of above mentioned entities, their logic, and wiring between them. These components are −

S.No	Components & Description
1	**Fragments** Represents a portion of user interface in an Activity.
2	**Views** UI elements that are drawn on-screen including buttons, lists forms etc.
3	**Layouts** View hierarchies that control screen format and appearance of the views.
4	**Intents** Messages wiring components together.
5	**Resources** External elements, such as strings, constants and drawable pictures.
6	**Manifest** Configuration file for the application.

What Is the Android SDK?



Appearance & Behavior › System Settings › Android SDK

Manager for the Android SDK and Tools used by Android Studio

Android SDK Location: /Users/kopriva/Library/Android/sdk

SDK Platforms | SDK Tools | SDK Update Sites

Below are the available SDK developer tools. Once installed, Android Studio will autom
check for updates. Check "show package details" to display available versions of an S

	Name	Version	Status
✓	Android SDK Build-Tools		Installed
✓	GPU Debugging tools		Installed
✓	CMake		Installed
✓	LLDB		Installed
✓	Android Auto API Simulators	1	Installed
✓	Android Auto Desktop Head Unit emulator	1.1	Installed
−	Android Emulator	27.0.2	Update Available: 27
✓	Android SDK Platform-Tools	27.0.1	Installed
✓	Android SDK Tools	26.1.1	Installed
✓	Documentation for Android SDK	1	Installed
✓	Google Play APK Expansion library	1	Installed
✓	Google Play Licensing Library	1	Installed
✓	Google Play services	46	Installed

Sho

The Android SDK is a collection of software development tools and libraries required to develop Android applications. Every time Google releases a new version of Android or an update, a corresponding SDK is also released which developers must download and install. It is worth noting that you can also download and use the Android SDK independently of Android Studio, but typically you'll be working through Android Studio for any Android development.

The Android SDK comprises all the tools necessary to code programs from scratch and even test them. These tools provide a smooth flow of the development process from developing and debugging, through to packaging.

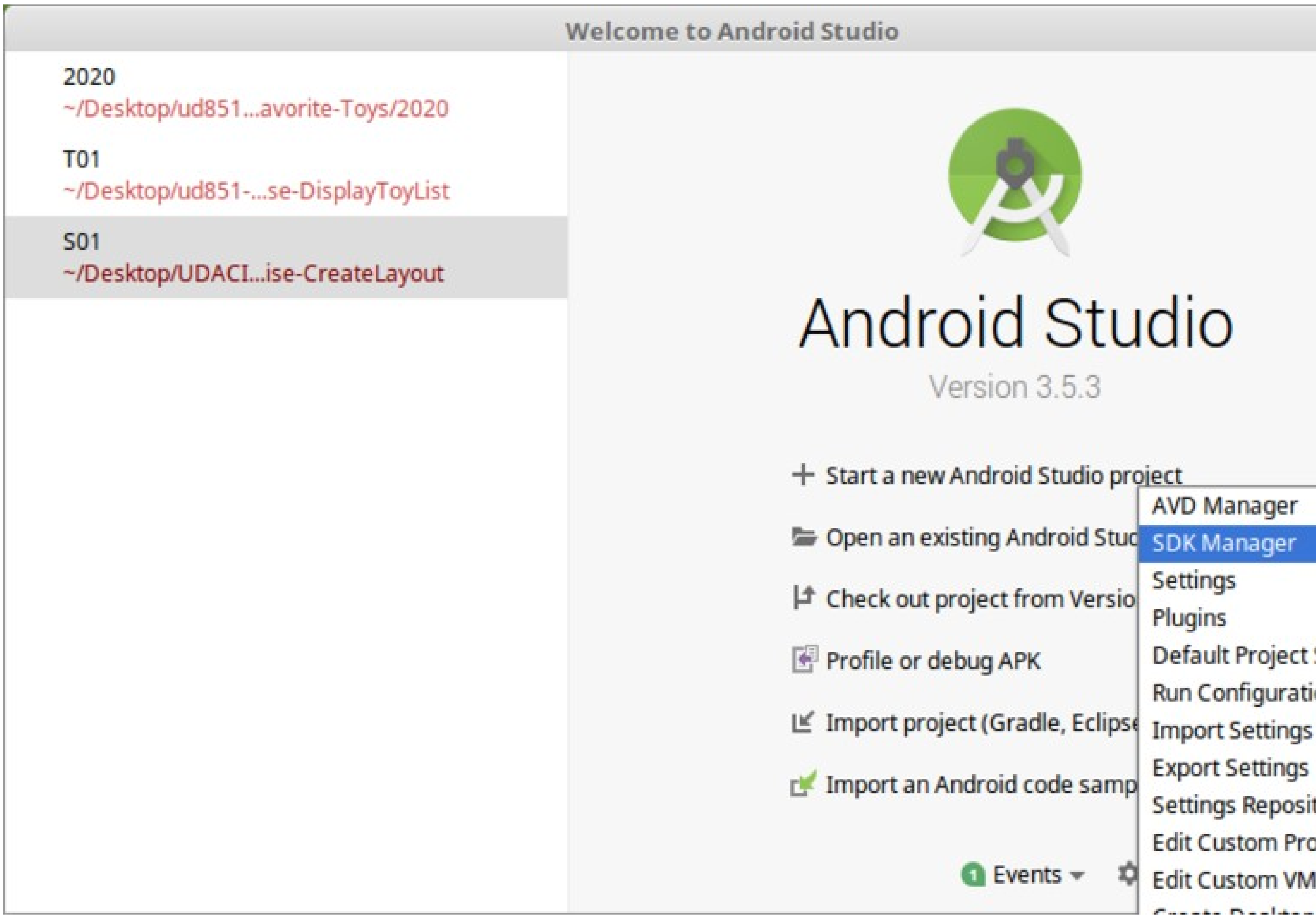
The Android SDK is compatible with Windows, macOS, and Linux, so you can develop on any of those platforms.

How to Install the Android SDK

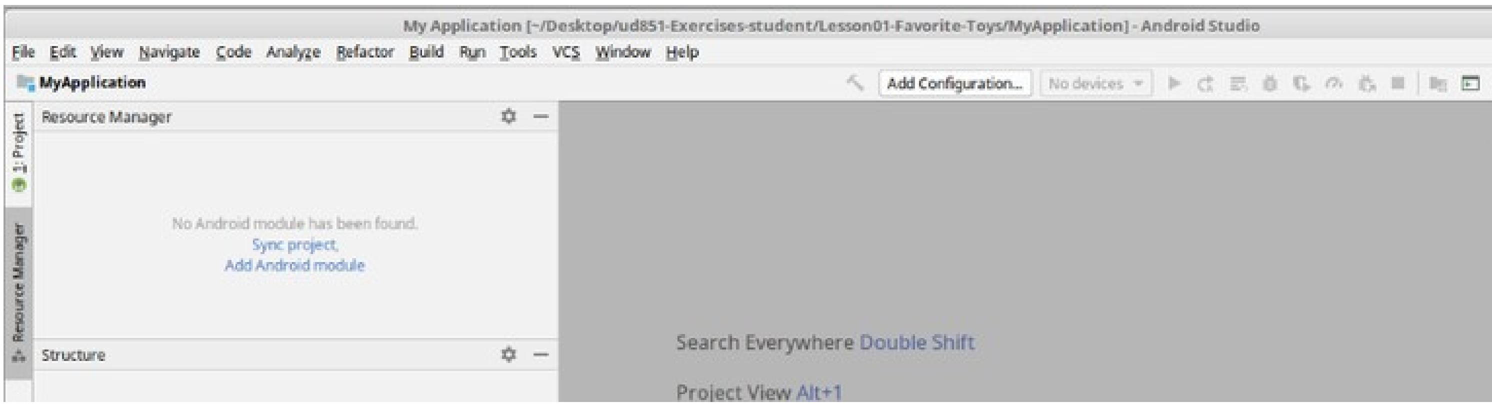
The Android SDK is optimized for Android Studio, and hence to effectively reap its benefits, you will need to install Android Studio. Having the Android SDK managed from within Android Studio is easier since support for languages like Java, Kotlin, and C++ is handled automatically. Not only that, but updates to the Android SDK are handled automatically by Android Studio.

To install the Android SDK from within Android Studio, first start Android Studio.

From the Android Studio start page, select Configure > SDK Manager.


Welcome to Android Studio
2020
~/Desktop/ud851...avorite-Toys/2020
T01
~/Desktop/ud851-...se-DisplayToyList
S01
~/Desktop/UDACI...ise-CreateLayout
Android Studio
Version 3.5.3
+ Start a new Android Studio project
Open an existing Android Stud
Check out project from Versio
Profile or debug APK
Import project (Gradle, Eclipse
Import an Android code samp
Events
AVD Manager
SDK Manager
Settings
Plugins
Default Project
Run Configurati
Import Settings
Export Settings
Settings Reposit
Edit Custom Pro
Edit Custom VM

If you already have Android Studio open, the SDK Manager icon is found on the top right corner, as shown below.


My Application [~/Desktop/ud851-Exercises-student/Lesson01-Favorite-Toys/MyApplication] - Android Studio
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
MyApplication
Add Configuration...
No devices
1: Project
Resource Manager
Resource Manager
No Android module has been found.
Sync project,
Add Android module
Structure
Search Everywhere Double Shift
Project View Alt+1

Install the required Android SDK platform packages and developer tools. A good start is to install:

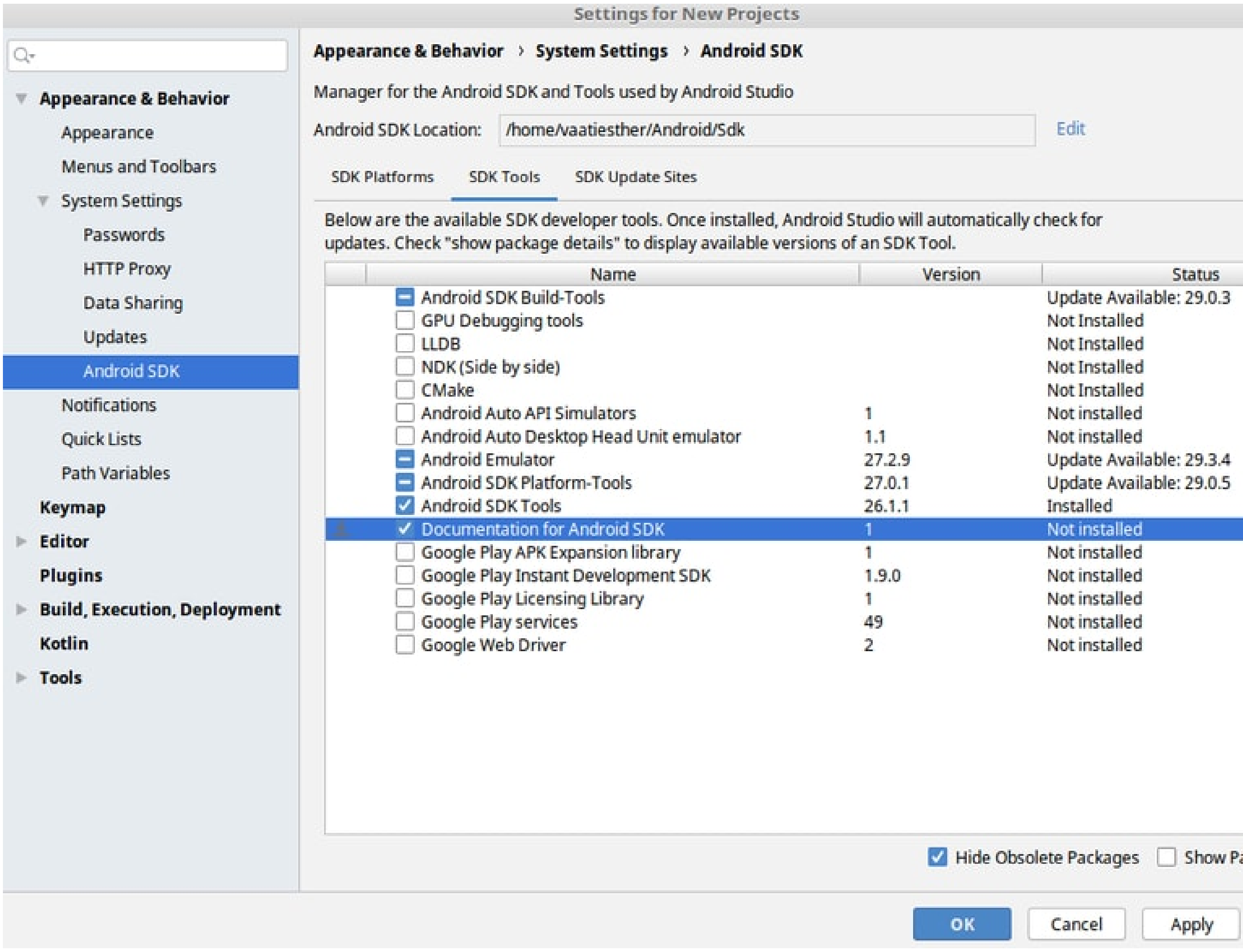
Android SDK Build-Tools

Android Emulator

Android SDK Platform-Tools

Android SDK Tools

Documentation for Android SDK



Manager for the Android SDK and Tools used by Android Studio

Android SDK Location: /home/vaatiesther/Android/Sdk

Below are the available SDK developer tools. Once installed, Android Studio will automatically check for updates. Check "show package details" to display available versions of an SDK Tool.

	Name	Version	Status
▬	Android SDK Build-Tools		Update Available: 29.0.3
☐	GPU Debugging tools		Not Installed
☐	LLDB		Not Installed
☐	NDK (Side by side)		Not Installed
☐	CMake		Not Installed
☐	Android Auto API Simulators	1	Not installed
☐	Android Auto Desktop Head Unit emulator	1.1	Not installed
▬	Android Emulator	27.2.9	Update Available: 29.3.4
▬	Android SDK Platform-Tools	27.0.1	Update Available: 29.0.5
✓	Android SDK Tools	26.1.1	Installed
✓	Documentation for Android SDK	1	Not installed
☐	Google Play APK Expansion library	1	Not installed
☐	Google Play Instant Development SDK	1.9.0	Not installed
☐	Google Play Licensing Library	1	Not installed
☐	Google Play services	49	Not installed
☐	Google Web Driver	2	Not installed

✓ Hide Obsolete Packages ☐ Show P

Click Apply, and Android Studio will install the selected tools and packages.

What are Android Emulators?

An Android emulator is a tool that creates virtual Android devices (with software and hardware) on your computer. Note that:

- It is a program (a process that runs on your computer's operating system).
- It works by mimicking the guest device's architecture (more on that in a bit).

In order to better understand what Android emulators are capable of, you should know how they work.

How do Android Emulators Work?

Complete platform-virtualization (hardware and software emulation) is possible with Quick Emulator (QEMU).

- ### Quick Emulator (QEMU)

  QEMU–shorthand for Quick EMUlator. It's an open-source and incredibly versatile tool. It can run on a large variety of host (workstation) CPUs/OSs and emulate an even larger range of guest CPUs/OSs.

  QEMU powers most Android emulators (including the one by Android Developer Studio).

  How it Works: It mimics guest device hardware. Then it translates the Application Binary Interface (ABI) of the guest device to match that of the host device. You equip this with with an OS and run it like a program on your computer.

  The translation of CPU architectures is a complex, time-consuming process which makes emulation almost painfully slow. Thankfully, it can be skipped if the guest and the host CPU architectures are the same, with the help of a hypervisor.

- ### Hypervisors

  Before 2017, Android Developer Studio's emulator had to translate Android's ARM architecture to match the Intel/AMD architectures commonly used in PCs.

  With the release of version 25.3.0, Developer Studio upgraded their emulator to support hardware-assisted virtualization.

  How it works: When the guest and the host devices have same instruction architecture (say, x86 Android system images and x86 Intel processor), QEMU skips the 'binary translation' part and runs the guest device directly on the host CPU. This is called hardware-assisted virtualization.

  You need a hypervisor to enable this. Intel's HAXM (Hardware Acceleration Execution Manager) is a hypervisor component for Windows and macOS. There's KVM (Kernel-based Virtual Machine) for Linux.

  x86 Android ABIs (Application Binary Interface, or system images) are now available to support hardware-acceleration on most computers. If your CPU supports it, the Android emulator prompts you to enable or install the hypervisor in order to speed up virtual device performance.

  With hardware-acceleration, the Android emulator can run virtual devices at speeds similar to that of your workstation CPU.

  Now that you know how it works, here's a brief introduction to the most popular Android emulator out there.

Android Emulator (by Android Developer Studio)

Post the release of version 25.3.0, the Android Emulator is distributed separately from Android SDK tools.

To install it from your Android Developer Studio console, go to Configure -> SDK Manager -> Android SDK. Within this screen, under the 'SDK Tools' tab, you'll find the Android Emulator. Check the box and click OK to install it.

The Android emulator can be tricky to set up. Here's a brief walkthrough:

- ### Hardware Prerequisites

In order to work with Android emulator, you'll need SDK version 26.1.1 or above and a PC/laptop with 64 or 86-bit processor. If you want to work with Android 8.1 or above, you'll need a webcam that can capture 720p frames.

The emulator may prompt you to enable hardware-acceleration (to speed up virtual device performance). For this, you'll need HAXM version 7.2.0-the Intel hypervisor we mentioned above.

Windows and Linux users will need more components, depending on their processor family:

- **Intel:** Intel VT-x and Intel EM64T (Intel 64) support; Execute Disable (XD) Bit capability.
- **AMD (Linux):** AMD Virtualization (AMD-V) support; SIMD Extensions set 3.
- **AMD (Windows):** Android Studio v3.2 or above; Windows Hypervisor Platform API.

Once you're done with the installation, you'll arrive at the Android Virtual Device (AVD) Manager:

- ### Android Virtual Device Manager – Things to Note

AVD Manager lets you set up and configure your virtual Android devices. Some key points:

1. **Device Type:** Contains pre-configured, manufacturer-specific device profiles for TV, Phone, Tablet, and Wear OS. You can customize screen size and certain external capabilities–or import a profile of your own. Interestingly, the Google Play Store and the Compatibility Test Suite are only available for a very limited number of devices–ones that meet the Android Compatibility Program criteria. You cannot add this capability to a cloned or custom device profile.
2. **System Image:** Lets you equip the virtual device with a system image/ABI and an Android OS version. ABIs come in ARM, x86, x86-64 variants for different Android OS versions/APIs. Each ABI has a file size of about 900 MB. The emulator will recommend ABIs that are similar in architecture to your workstation processor, but you can pick others (with a trade-off in emulator performance). Only certain system images have Google APIs (which give you access to Google apps like Maps, Gmail, etc.).
3. **Verify Configuration:** Allows you to change previously defined hardware/software profiles and configure some startup settings for the virtual device (name, orientation etc.)

Once you click 'Finish', the emulator launches the virtual device instance with your specifications– **complete with extended controls to adjust device rotation, geolocation, camera, fingerprint sensor, network latency, battery state, and more.**

You can test your web app, a project from Android Studio or upload a native app APK from your own device.

By default, these virtual device instances get saved in the state they were in when you closed them. You can also save snapshots of the device state (includes OS, your app/project, and user data) and pick up where you left off later.

Third-Party Alternatives to Developer Studio’s Android Emulator

Before version 25.3.0, most developers learned to stay away from the slow Android emulator that came with Developer Studio. So for early-stage testing, they used third-party online Android emulators. Genymotion was a popular choice.

Today, there are third-party Android emulators like Bluestacks, Nox, MeMu, etc. geared towards gamers who want uninterrupted gameplay and better controls of PC/laptop for immersive Android games (think PubG, Battle Royale Mode, Iron Throne, et al).

Some developers still prefer these alternatives. Since they have fewer extended controls than the Developer Studio emulator, they create and launch virtual device instances a lot faster. But for all the purported benefits, there are trade-offs with third-party/online Android emulators:

- **Cryptocurrency Miners:** In July 2018, a redditor’s investigation revealed that AndY Android emulator dropped a cryptocurrency miner on users’ devices.
- **Fake app-rating farms:** Some third-party emulators periodically force users to download and rate random apps from the Play store.
- **Bloating:** Some vendors monetize their emulators via in-app ads. Multiple ad calls and heavy ad creatives cause bloating and lag.

Android Emulator: Capabilities and Limitations

The latest Android emulator by Developer Studio can closely imitate a real Android device. But it still has its limitations.

- ### Capabilities

  Data transfer is faster on a virtual device (than a physical device connected via USB). The drag-and-drop file upload lets you place .apk files from your computer to the virtual mobile device. It’s particularly great when developers need to quickly test apps under context.

  The emulator is also pretty useful when you’re working with physical sensors like the accelerometer. If you were testing a specific app feature that relies on the sensors, it’ll be easier to configure the settings through the visual, extended controls of the emulator.

- ### Limitations

  The most popular chipset for Android smartphones out there is ARM v7a. Most PCs/laptops run on Intel (x86). Recall that guest and host CPU architectures need to match for faster emulation. Basically, without a computer equipped with an ARM processor, **you’re stuck with poor emulation of most of the commercially-available Android devices.**

  The AVD Manager creates separate directories to store each virtual device’s user data, SD card data, and cache. A single virtual device can take as much as 3.5GB of your disk space. **Over time, a library of virtual devices will clam up your workstation.**

Virtual devices' performance is affected by that of your workstation. **The emulator will crash and burn if you don't have enough free disk space at launch.**

Enabling hardware-acceleration takes care of performance issues. **But setting up hardware-acceleration is a complex process that even experienced developers struggle with.** The results often lead to complete system failure.

Android emulator isn't reliable when it comes to understanding app interactions with the native device environment. For instance, you'd never know:

- Which background processes your app runs
- How front-end appears in different brightness levels
- How the app responds to a complete range of touch-gestures

Start Testing on Real Device

Testing: Android Emulators vs Real Devices

In CI/CD pipelines, developers execute tests on code before committing changes to a branch. After some quick unit testing within the IDE, functional and instrumentation testing take precedence.

You can execute some instrumented tests on Android emulator. An example would be verifying that the code correctly handles app-specific or core platform resource files (like font, animations, UI, etc.), or testing to see if external dependencies on 'sign-up via connected apps' (Facebook, Google etc.) functionality works.

App Performance testing is done to verify that interactions are smooth, that there's no jank (dropped frames), and that the app uses device resources (battery and memory) within reasonable constraints. To get accurate results, you first need an accurate measure of the CPU and graphics capabilities of the target device.

At best, the Android emulator can give you virtual devices running on near-native speed. There's no way you can expect native results with them. **On virtual devices, you can't test your Android app's performance against any benchmarks.**

Plus-and this goes without saying-there's no easy or reliable workaround that'd let you test native or web apps at scale with Android emulators.

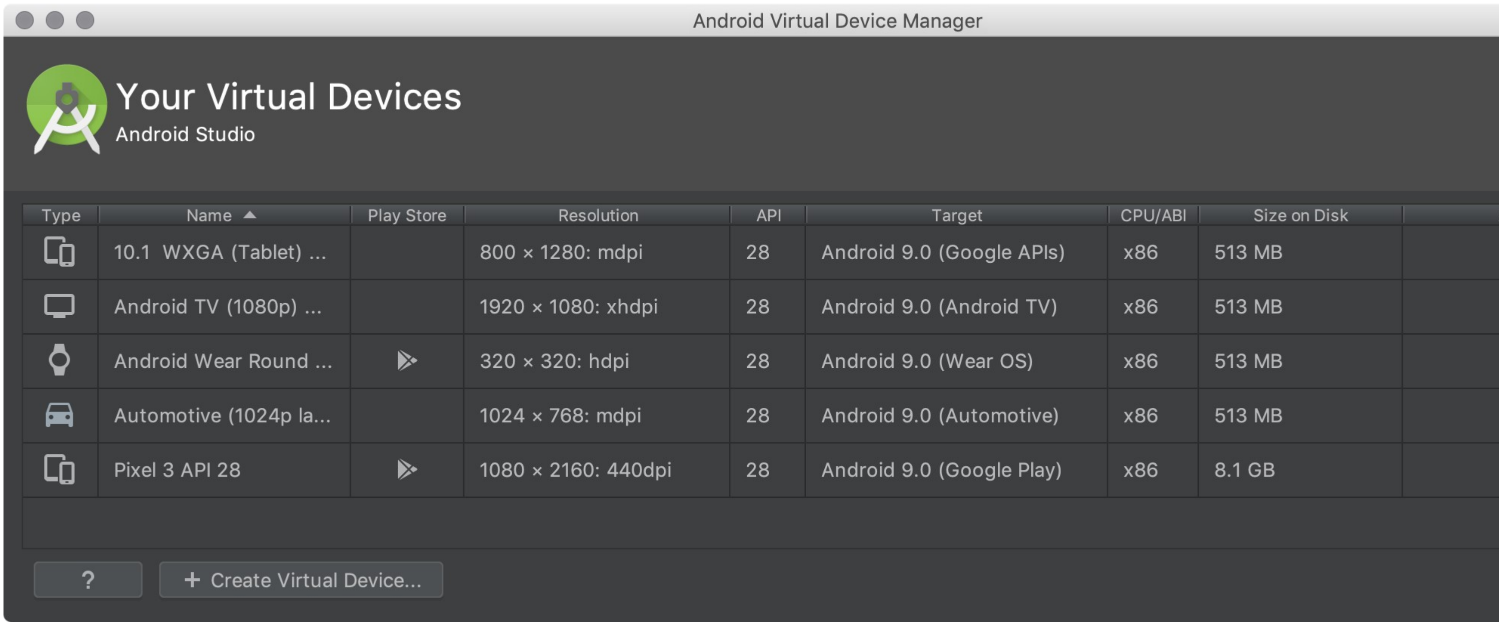
For serious, pre-release cross browser testing (on UI and all functionalities), you will need a diverse collection of real Android devices. QA engineers use testing automation frameworks like Appium or Espresso, write test scripts, and execute them on as many different Android devices as necessary to meet the benchmarked compatibility standards for a given target market.

Create and manage virtual devices

An Android Virtual Device (AVD) is a configuration that defines the characteristics of an Android phone, tablet, Wear OS, Android TV, or Automotive OS device that you want to simulate in the Android Emulator. 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** in the toolbar.



Android Virtual Device Manager

Your Virtual Devices
Android Studio

Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk
	10.1 WXGA (Tablet) ...		800 × 1280: mdpi	28	Android 9.0 (Google APIs)	x86	513 MB
	Android TV (1080p) ...		1920 × 1080: xhdpi	28	Android 9.0 (Android TV)	x86	513 MB
	Android Wear Round ...	▶	320 × 320: hdpi	28	Android 9.0 (Wear OS)	x86	513 MB
	Automotive (1024p la...		1024 × 768: mdpi	28	Android 9.0 (Automotive)	x86	513 MB
	Pixel 3 API 28	▶	1080 × 2160: 440dpi	28	Android 9.0 (Google Play)	x86	8.1 GB

About AVDs

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

We recommend that you create an AVD for each system image that your app could potentially support based on the `<uses-sdk>` setting in your manifest.

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.

Notice that only some hardware profiles are indicated to include **Play Store**. This indicates that these profiles are fully CTS compliant and may use system images that include the Play Store app.

System images

A system image labeled with **Google APIs** includes access to Google Play services. A system image labeled with the Google Play logo in the **Play Store** column includes the Google Play Store app *and* access to Google Play services, including a **Google Play** tab in the **Extended controls** dialog that provides a convenient button for updating Google Play services on the device.

To ensure app security and a consistent experience with physical devices, system images with the Google Play Store included are signed with a release key, which means that you cannot get elevated privileges (root) with these images. If you require elevated privileges (root) to aid with your app troubleshooting, you can use the Android Open Source Project (AOSP) system images that do not include Google apps or 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. If needed, you can use the AVD Manager to wipe user data, so the device has the same data as if it were new.

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

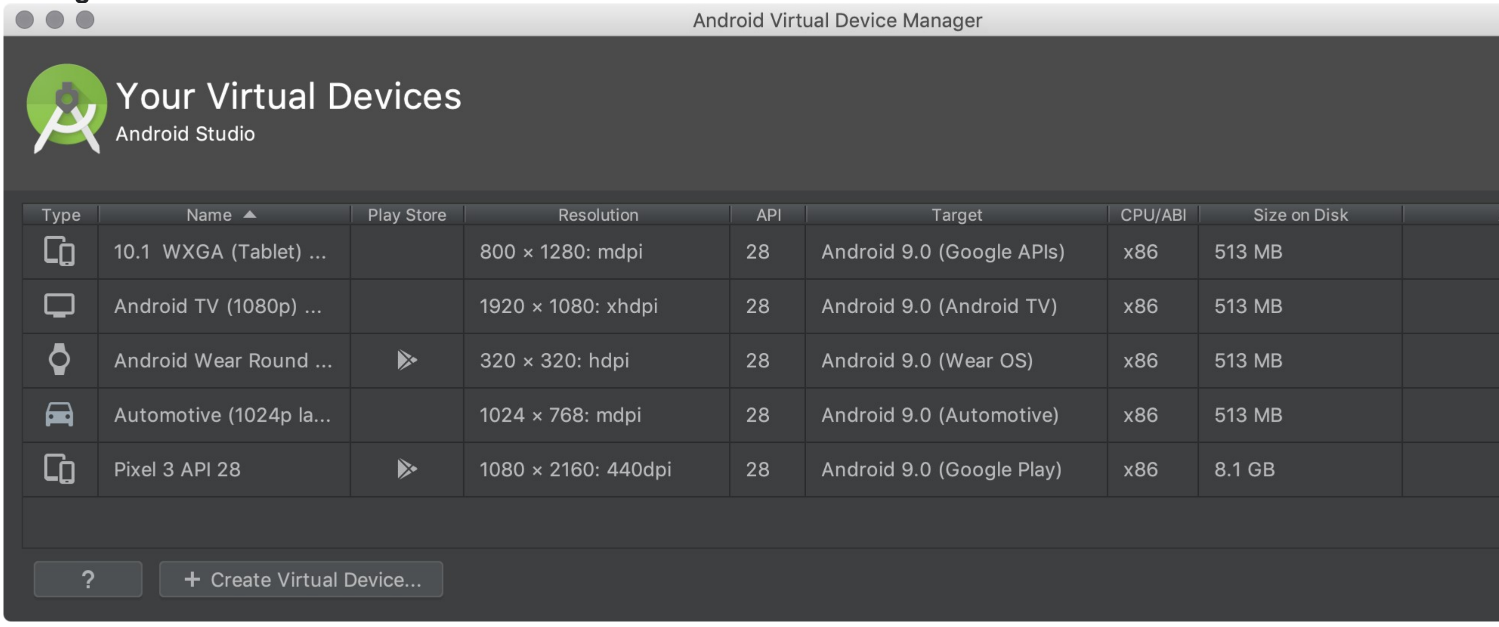
Be sure your AVD definition includes the device features your app depends on. See Hardware Profile Properties and AVD Properties for lists of features you can define in your AVDs.

Create an AVD

Tip: If you want to launch your app into an emulator, instead run your app from Android Studio and then in the **Select Deployment Target** dialog that appears, click **Create New Virtual Device**.

To create a new AVD:

1. Open the AVD Manager by clicking **Tools > AVD Manager**.



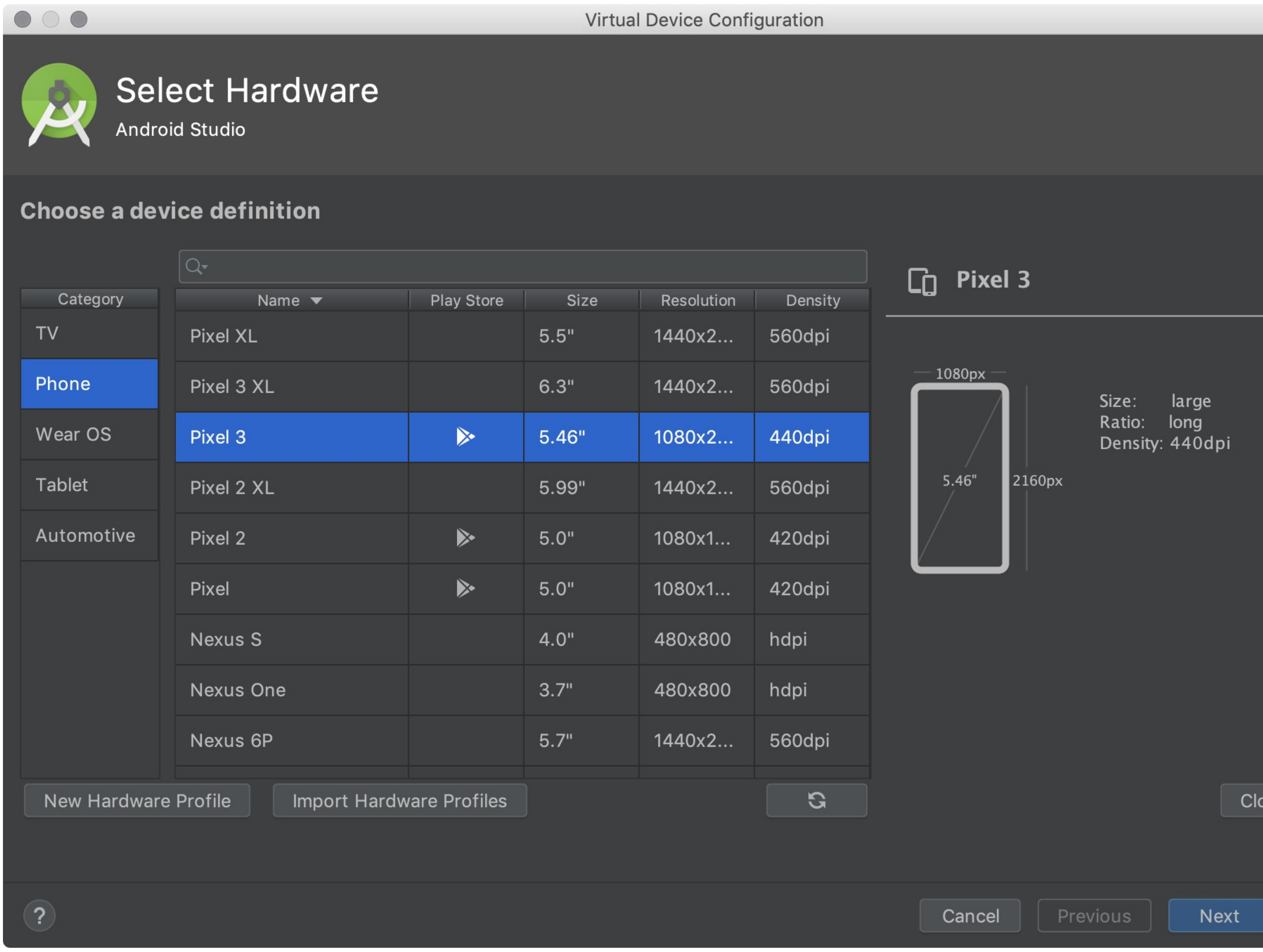
Android Virtual Device Manager

Your Virtual Devices
Android Studio

Type	Name ▲	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk
	10.1 WXGA (Tablet) ...		800 × 1280: mdpi	28	Android 9.0 (Google APIs)	x86	513 MB
	Android TV (1080p) ...		1920 × 1080: xhdpi	28	Android 9.0 (Android TV)	x86	513 MB
	Android Wear Round ...	▶	320 × 320: hdpi	28	Android 9.0 (Wear OS)	x86	513 MB
	Automotive (1024p la...		1024 × 768: mdpi	28	Android 9.0 (Automotive)	x86	513 MB
	Pixel 3 API 28	▶	1080 × 2160: 440dpi	28	Android 9.0 (Google Play)	x86	8.1 GB

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

The **Select Hardware** page appears.


Virtual Device Configuration

Select Hardware

Android Studio

Choose a device definition

Category
TV
Phone
Wear OS
Tablet
Automotive

Name	Play Store	Size	Resolution	Density
Pixel XL		5.5"	1440x2...	560dpi
Pixel 3 XL		6.3"	1440x2...	560dpi
Pixel 3	▶	5.46"	1080x2...	440dpi
Pixel 2 XL		5.99"	1440x2...	560dpi
Pixel 2	▶	5.0"	1080x1...	420dpi
Pixel	▶	5.0"	1080x1...	420dpi
Nexus S		4.0"	480x800	hdpi
Nexus One		3.7"	480x800	hdpi
Nexus 6P		5.7"	1440x2...	560dpi

Pixel 3

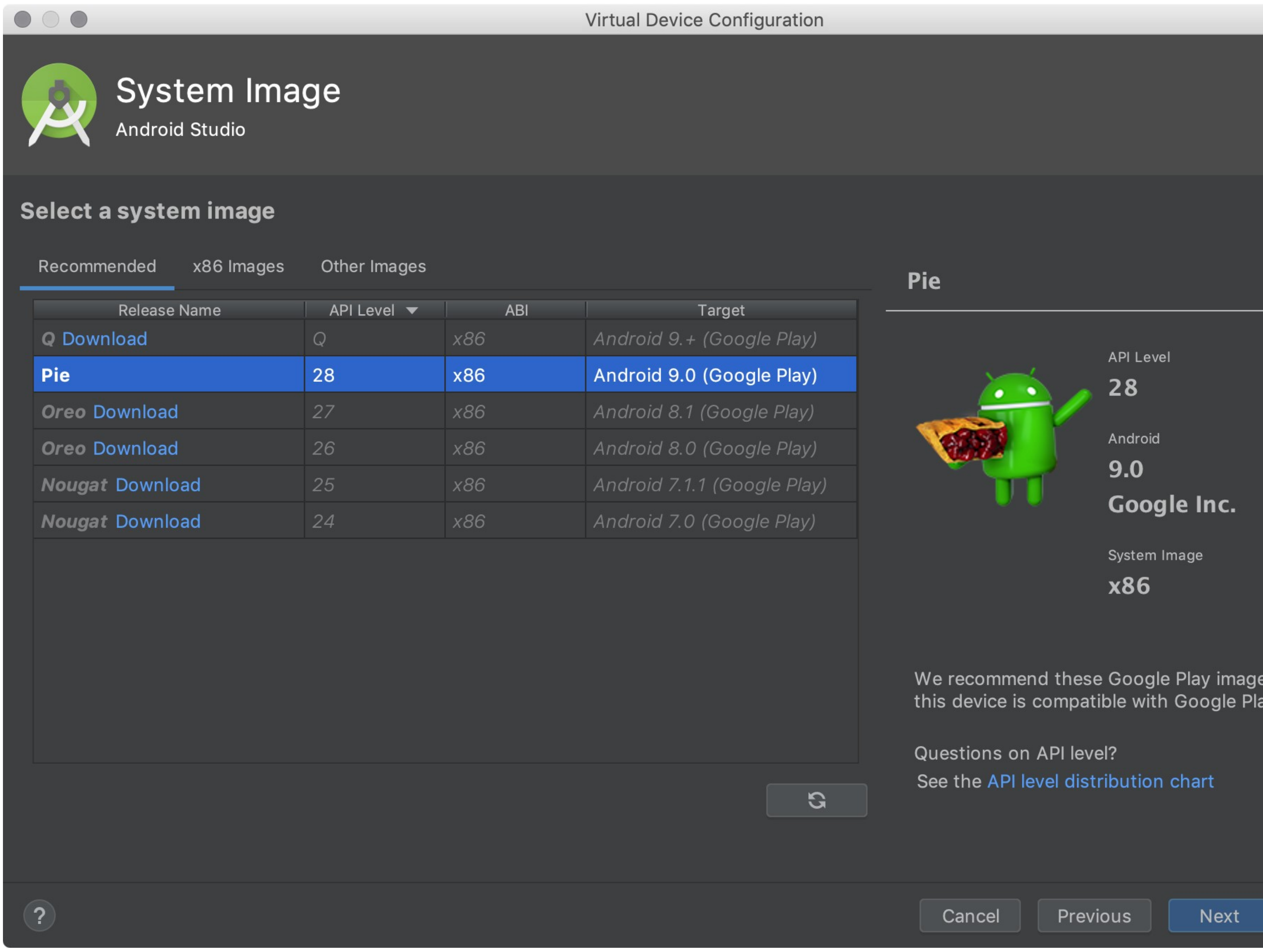
Size: large
Ratio: long
Density: 440dpi

Notice that only some hardware profiles are indicated to include **Play Store**. This indicates that these profiles are fully CTS compliant and may use system images that include the Play Store app.

3. 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.


Virtual Device Configuration

System Image

Android Studio

Select a system image

Release Name	API Level	ABI	Target
Q Download	Q	x86	Android 9.+ (Google Play)
Pie	28	x86	Android 9.0 (Google Play)
Oreo Download	27	x86	Android 8.1 (Google Play)
Oreo Download	26	x86	Android 8.0 (Google Play)
Nougat Download	25	x86	Android 7.1.1 (Google Play)
Nougat Download	24	x86	Android 7.0 (Google Play)

Pie

API Level
28

Android
9.0
Google Inc.

System Image
x86

We recommend these Google Play image
this device is compatible with Google Pla

Questions on API level?
See the API level distribution chart

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

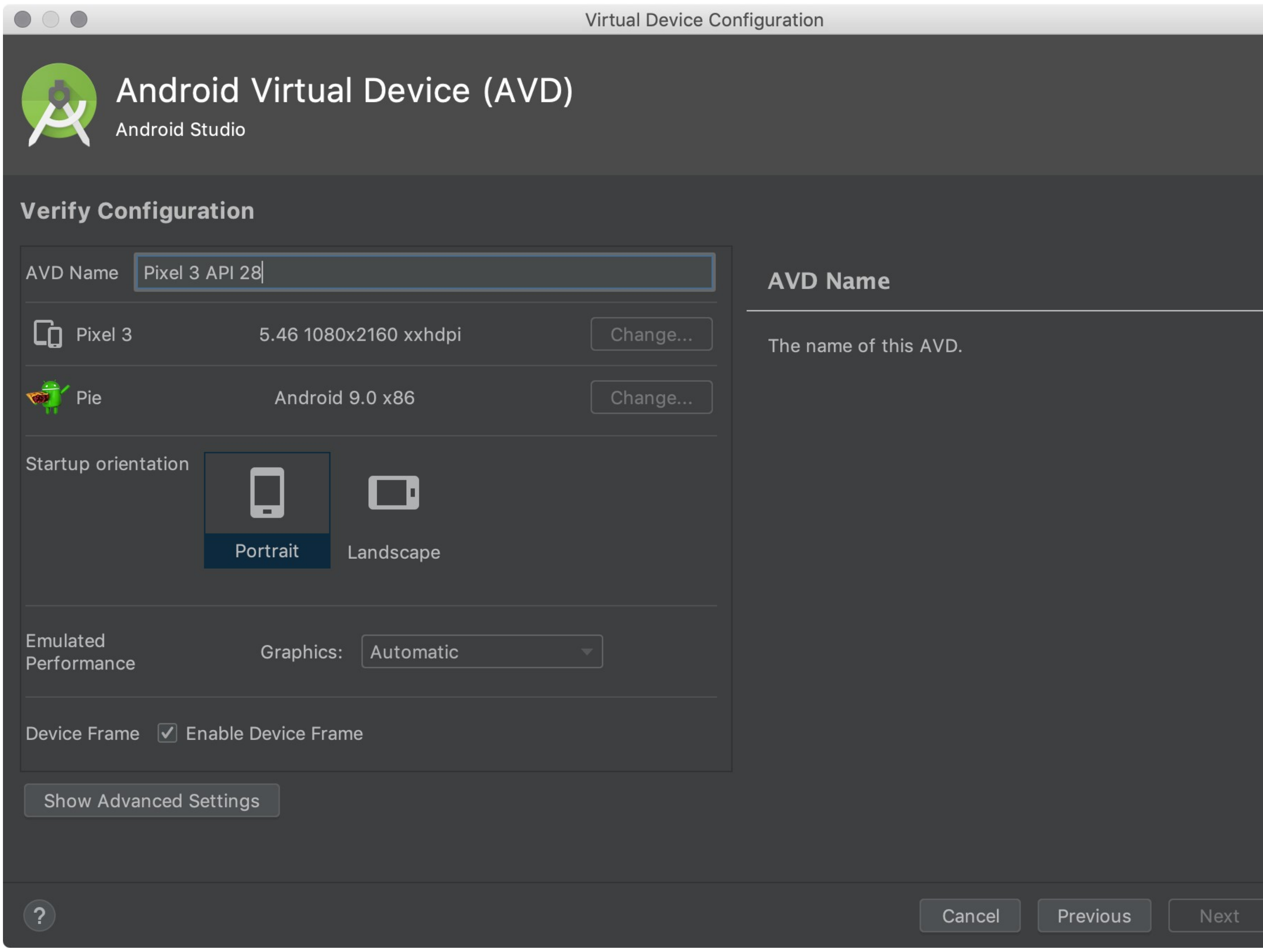
The **Recommended** tab lists recommended system images. The other tabs include a more complete list. The right pane describes the selected system image. x86 images run the fastest in the emulator.

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. For more information about the relationship between system API level and `minSdkVersion`, see [Versioning Your Apps](#).

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.


Virtual Device Configuration

Android Virtual Device (AVD)
Android Studio

Verify Configuration

AVD Name: Pixel 3 API 28

Pixel 3 5.46 1080x2160 xxhdpi

Pie Android 9.0 x86

Startup orientation: Portrait

Emulated Performance Graphics: Automatic

Device Frame ✓ Enable Device Frame

AVD Name

The name of this AVD.

5. 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.

To create an AVD starting with a copy:

1. From the **Your Virtual Devices** page of the AVD Manager, right-click an AVD and select **Duplicate**.

Or click Menu ▼ and select **Duplicate**.

The **Verify Configuration** page appears.

2. Click **Change** or **Previous** if you need to make changes on the **System Image** and **Select Hardware** pages.

3. Make your changes, and then click **Finish**.

The AVD appears in the **Your Virtual Devices** page.

Create a hardware profile

The AVD Manager provides predefined hardware profiles for common devices so you can easily add them to your AVD definitions. If you need to define a different device, you can create a new hardware profile. You can define a new hardware profile from the beginning, or copy a hardware profile as a start. The preloaded hardware profiles aren't editable.

To create a new hardware profile from the beginning:

1. In the **Select Hardware** page, click **New Hardware Profile**.
2. In the **Configure Hardware Profile** page, change the hardware profile properties as needed.
3. Click **Finish**.

Your new hardware profile appears in the **Select Hardware** page. You can optionally create an AVD that uses the hardware profile by clicking **Next**. Or, click **Cancel** to return to the **Your Virtual Devices** page or **Select Deployment Target** dialog.

To create a hardware profile starting with a copy:

1. In the **Select Hardware** page, select a hardware profile and click **Clone Device**.

Or right-click a hardware profile and select **Clone**.

2. In the **Configure Hardware Profile** page, change the hardware profile properties as needed.
3. Click **Finish**.

Your new hardware profile appears in the **Select Hardware** page. You can optionally create an AVD that uses the hardware profile by clicking **Next**. Or, click **Cancel** to return to the **Your Virtual Devices** page or **Select Deployment Target** dialog.

Edit existing AVDs

From the **Your Virtual Devices** page, you can perform the following operations on an existing AVD:

- To edit an AVD, click **Edit this AVD** and make your changes.
- To delete an AVD, right-click an AVD and select **Delete**. Or click Menu ▼ and select **Delete**.
- To show the associated AVD `.ini` and `.img` files on disk, right-click an AVD and select **Show on Disk**. Or click Menu ▼ and select **Show on Disk**.
- To view AVD configuration details that you can include in any bug reports to the Android Studio team, right-click an AVD and select **View Details**. Or click Menu ▼ and select **View Details**.

Edit existing hardware profiles

From the **Select Hardware** page, you can perform the following operations on an existing hardware profile:

- To edit a hardware profile, select it and click **Edit Device**. Or right-click a hardware profile and select **Edit**. Next, make your changes.
- To delete a hardware profile, right-click it and select **Delete**.

You can't edit or delete the predefined hardware profiles.

Run and stop an emulator, and clear data

From the **Your Virtual Devices** page, you can perform the following operations on an emulator:

- To run an emulator that uses an AVD, double-click the AVD. Or click **Launch** ▶.
- To stop a running emulator, right-click an AVD and select **Stop**. Or click Menu ▼ and select **Stop**.

- To clear the data for an emulator, and return it to the same state as when it was first defined, right-click an AVD and select **Wipe Data**. Or click Menu ▼ and select **Wipe Data**.

Import and export hardware profiles

From the **Select Hardware** page, you can import and export hardware profiles:

- To import a hardware profile, click **Import Hardware Profiles** and select the XML file containing the definition on your computer.
- To export a hardware profile, right-click it and select **Export**. Specify the location where you want to store the XML file containing the definition.

Hardware profile properties

You can specify the following properties of hardware profiles in the **Configure Hardware Profile** page. AVD configuration properties override hardware profile properties, and emulator properties that you set while the emulator is running override them both.

The predefined hardware profiles included with the AVD Manager aren't editable. However, you can copy them and edit the copies.

Hardware Profile Property	**Description**
Device Name	Name of the hardware profile. The name can contain uppercase or lowercase letters, numbers from 0 to 9, periods (.), underscores (_), parentheses (()), and spaces. The name of the file storing the hardware profile is derived from the hardware profile name.
Device Type	Select one of the following: Phone/Tablet Wear OS Android TV Chrome OS Device Android Automotive
Screen Size	The physical size of the screen, in inches, measured at the diagonal. If the size is larger than your computer screen, it's reduced in size at launch.
Screen Resolution	Type a width and height in pixels to specify the total number of pixels on the simulated screen.
Round	Select this option if the device has a round screen, such as some Wear OS devices.
Memory: RAM	Type a RAM size for the device and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte).
Input: Has Hardware Buttons (Back/Home/Menu)	Select this option if your device has hardware navigation buttons. Deselect it if these buttons are implemented in software only. If you select this option, the buttons won't appear on the screen. You can use the emulator side panel to "press" the buttons, in either case.
Input: Has Hardware	Select this option if your device has a hardware keyboard. Deselect it if it doesn't. If you select

Keyboard	this option, a keyboard won't appear on the screen. You can use your computer keyboard to send keystrokes to the emulator, in either case.
Navigation Style	Select one of the following: None - No hardware controls. Navigation is through the software. D-pad - Directional Pad support. Trackball Wheel These options are for actual hardware controls on the device itself. However, the events sent to the device by an external controller are the same.
Supported Device States	Select one or both options: Portrait - Oriented taller than wide. Landscape - Oriented wider than tall. If you select both, you can switch between orientations in the emulator. You must select at least one option to continue.
Cameras	To enable the camera, select one or both options: Back-Facing Camera - The lens faces away from the user. Front-Facing Camera - The lens faces toward the user. Later, you can use a webcam or a photo provided by the emulator to simulate taking a photo with the camera.
Sensors: Accelerometer	Select if the device has hardware that helps the device determine its orientation.
Sensors: Gyroscope	Select if the device has hardware that detects rotation or twist. In combination with an accelerometer, it can provide smoother orientation detection and support a six-axis orientation system.
Sensors: GPS	Select if the device has hardware that supports the Global Positioning System (GPS) satellite-based navigation system.
Sensors: Proximity Sensor	Select if the device has hardware that detects if the device is close to your face during a phone call to disable input from the screen.
Default Skin	Select a skin that controls what the device looks like when displayed in the emulator. Remember that specifying a screen size that's too big for the resolution can mean that the screen is cut off, so you can't see the whole screen. See Create an emulator skin for more information.

AVD properties

You can specify the following properties for AVD configurations in the **Verify Configuration** page. The AVD configuration specifies the interaction between the development computer and the emulator, as well as properties you want to override in the hardware profile.

AVD configuration properties override hardware profile properties. Emulator properties that you set while the emulator is running override them both.

AVD Property	Description
AVD Name	Name of the AVD. The name can contain uppercase or lowercase letters, numbers from 0 to 9, periods (.), underscores (_), parentheses (()), dashes (-), and spaces. The name of the file storing the AVD configuration is derived from the AVD name.
AVD ID (Advanced)	The AVD filename is derived from the ID, and you can use the ID to refer to the AVD from the command line.
Hardware Profile	Click Change to select a different hardware profile in the Select Hardware page.
System Image	Click Change to select a different system image in the System Image page. An active internet connection is required to download a new image.
Startup Orientation	Select one option for the initial emulator orientation: Portrait - Oriented taller than wide. Landscape - Oriented wider than tall. An option is enabled only if it's selected in the hardware profile. When running the AVD in the emulator, you can change the orientation if both portrait and landscape are supported in the hardware profile.
Camera (Advanced)	To enable a camera, select one or both options: Front - The lens faces away from the user. Back - The lens faces toward the user. The Emulated setting produces a software-generated image, while the Webcam setting uses your development computer webcam to take a picture. This option is available only if it's selected in the hardware profile; it's not available for Wear OS and Android TV.
Network: Speed (Advanced)	Select a network protocol to determine the speed of data transfer: GSM - Global System for Mobile Communications HSCSD - High-Speed Circuit-Switched Data GPRS - Generic Packet Radio Service EDGE - Enhanced Data rates for GSM Evolution UMTS - Universal Mobile Telecommunications System HSDPA - High-Speed Downlink Packet Access LTE - Long-Term Evolution Full (default) - Transfer data as quickly as your computer allows.
Network: Latency (Advanced)	Select a network protocol to set how much time (delay) it takes for the protocol to transfer a data packet from one point to another point.

Emulated Performance: Graphics	Select how graphics are rendered in the emulator: Hardware - Use your computer graphics card for faster rendering. Software - Emulate the graphics in software, which is useful if you're having a problem with rendering in your graphics card. Automatic - Let the emulator decide the best option based on your graphics card.
Emulated Performance: Boot option (Advanced)	Cold boot - Start the device each time by powering up from the device-off state. Quick boot - Start the device by loading the device state from a saved snapshot. For details, see Run the emulator with Quick Boot.
Emulated Performance: Multi-Core CPU (Advanced)	Select the number of processor cores on your computer that you'd like to use for the emulator. Using more processor cores speeds up the emulator.
Memory and Storage: RAM	The amount of RAM on the device. This value is set by the hardware manufacturer, but you can override it, if needed, such as for faster emulator operation. Increasing the size uses more resources on your computer. Type a RAM size and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte).
Memory and Storage: VM Heap	The VM heap size. This value is set by the hardware manufacturer, but you can override it, if needed. Type a heap size and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte). For more information on Android VMs, see Memory Management for Different Virtual Machines.
Memory and Storage: Internal Storage	The amount of nonremovable memory space available on the device. This value is set by the hardware manufacturer, but you can override it, if needed. Type a size and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte).
Memory and Storage: SD Card	The amount of removable memory space available to store data on the device. To use a virtual SD card managed by Android Studio, select Studio-managed, type a size, and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte). A minimum of 100 MB is recommended to use the camera. To manage the space in a file, select External file and click ... to specify the file and location. For more information, see mksdcard and AVD data directory.
Device Frame: Enable Device Frame	Select to enable a frame around the emulator window that mimics the look of a real device.
Custom Skin Definition (Advanced)	Select a skin that controls what the device looks like when displayed in the emulator. Remember that specifying a screen size that's too big for the skin can mean that the screen is cut off, so you can't see the whole screen. See Create an emulator skin for more information.
Keyboard: Enable Keyboard Input (Advanced)	Select this option if you want to use your hardware keyboard to interact with the emulator. It's disabled for Wear OS and Android TV.

Create an emulator skin

An Android emulator skin is a collection of files that define the visual and control elements of an emulator display. If the skin definitions available in the AVD settings don't meet your requirements, you can create your own custom skin definition, and then apply it to your AVD.

Each emulator skin contains:

- A `hardware.ini` file
- Layout files for supported orientations (landscape, portrait) and physical configuration
- Image files for display elements, such as background, keys and buttons

To create and use a custom skin:

1. Create a new directory where you will save your skin configuration files.
2. Define the visual appearance of the skin in a text file named `layout`. This file defines many characteristics of the skin, such as the size and image assets for specific buttons. For example:

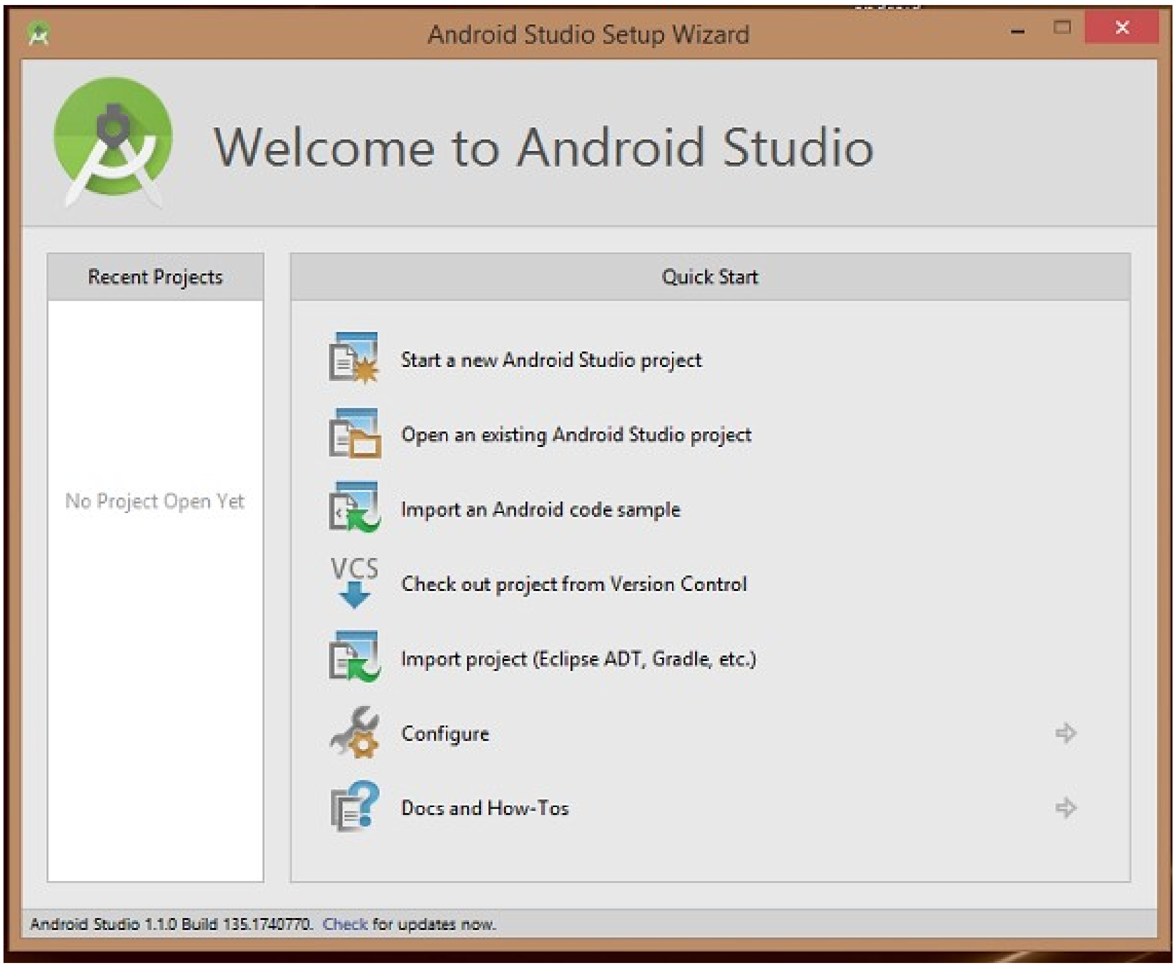
```
parts {
    device {
        display {
            width   320
            height  480
            x       0
            y       0
        }
    }


portrait {
    background {
        image background_port.png
    }

    buttons {
        power {
            image  button_vertical.png
            x 1229
            y 616
        }
    }
}
...




}
```

3. Add the bitmap files of the device images in the same directory.
4. Specify additional hardware-specific device configurations in a `hardware.ini` file for the device settings, such as `hw.keyboard` and `hw.lcd.density`.
5. Archive the files in the skin folder and select the archive file as a custom skin.

For more detailed information about creating emulator skins, see the Android Emulator Skin File Specification in the tools source code.

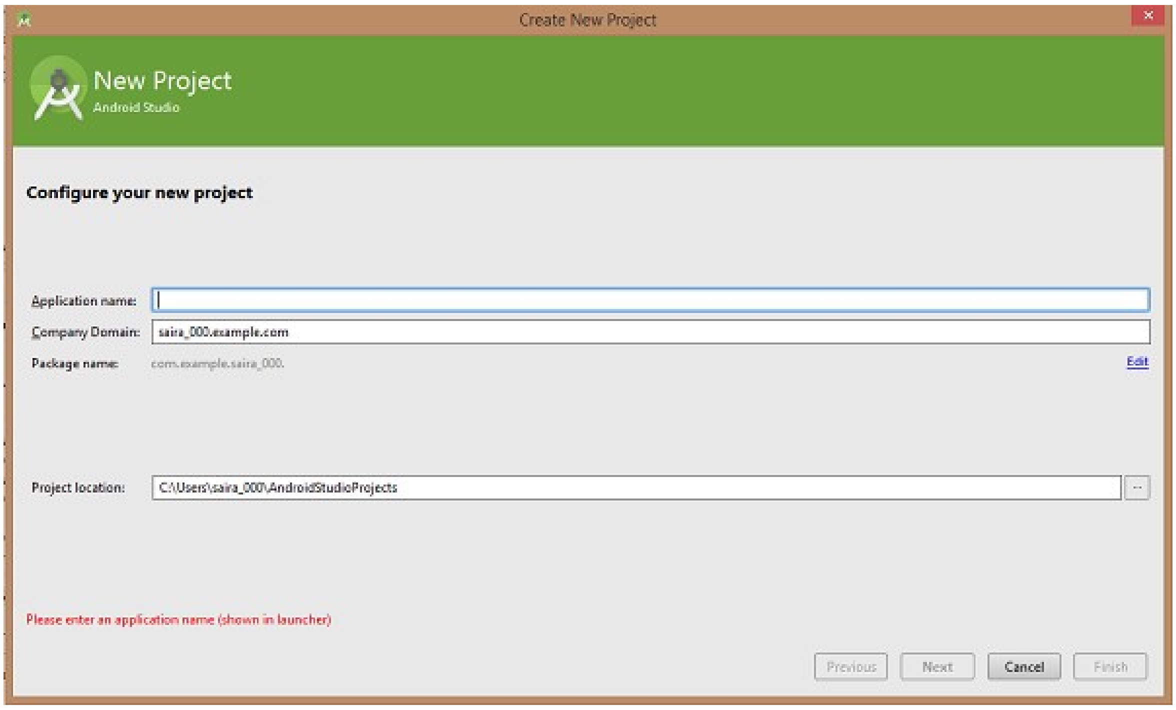
Create Android Application

The first step is to create a simple Android Application using Android studio. When you click on Android studio icon, it will show screen as shown below


Android Studio Setup Wizard
Welcome to Android Studio
Recent Projects
No Project Open Yet
Quick Start
Start a new Android Studio project
Open an existing Android Studio project
Import an Android code sample
VCS
Check out project from Version Control
Import project (Eclipse ADT, Gradle, etc.)
Configure
Docs and How-Tos

Android Studio 1.1.0 Build 135.1740770. Check for updates now.

You can start your application development by calling start a new android studio project. in a new installation frame should ask Application name, package information and location of the project.−

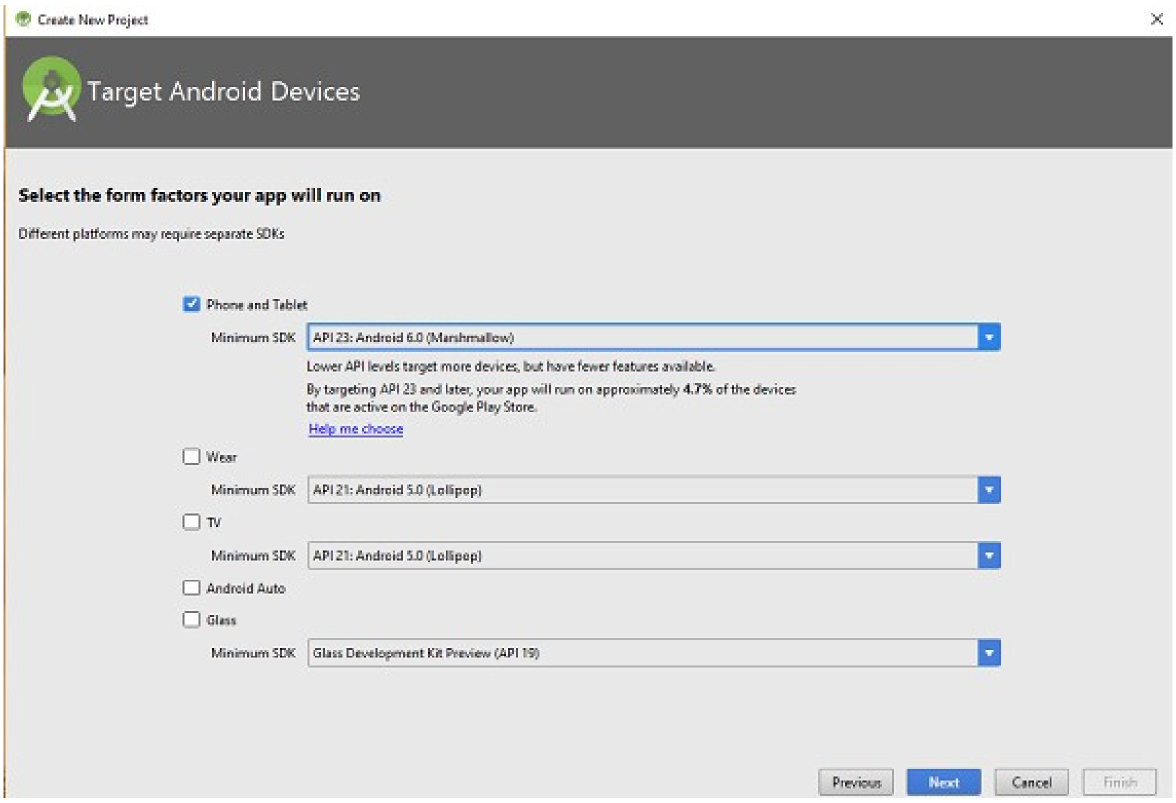


Configure your new project

Application name:
Company Domain: saira_000.example.com
Package name: com.example.saira_000.
Project location: C:\Users\saira_000\AndroidStudioProjects

Please enter an application name (shown in launcher)

After entered application name, it going to be called select the form factors your application runs on, here need to specify Minimum SDK, in our tutorial, I have declared as API23: Android 6.0(Mashmallow) −


Create New Project

Target Android Devices

Select the form factors your app will run on

Different platforms may require separate SDKs

[✓] Phone and Tablet
Minimum SDK: API 23: Android 6.0 (Marshmallow)
Lower API levels target more devices, but have fewer features available.
By targeting API 23 and later, your app will run on approximately 4.7% of the devices that are active on the Google Play Store.
Help me choose

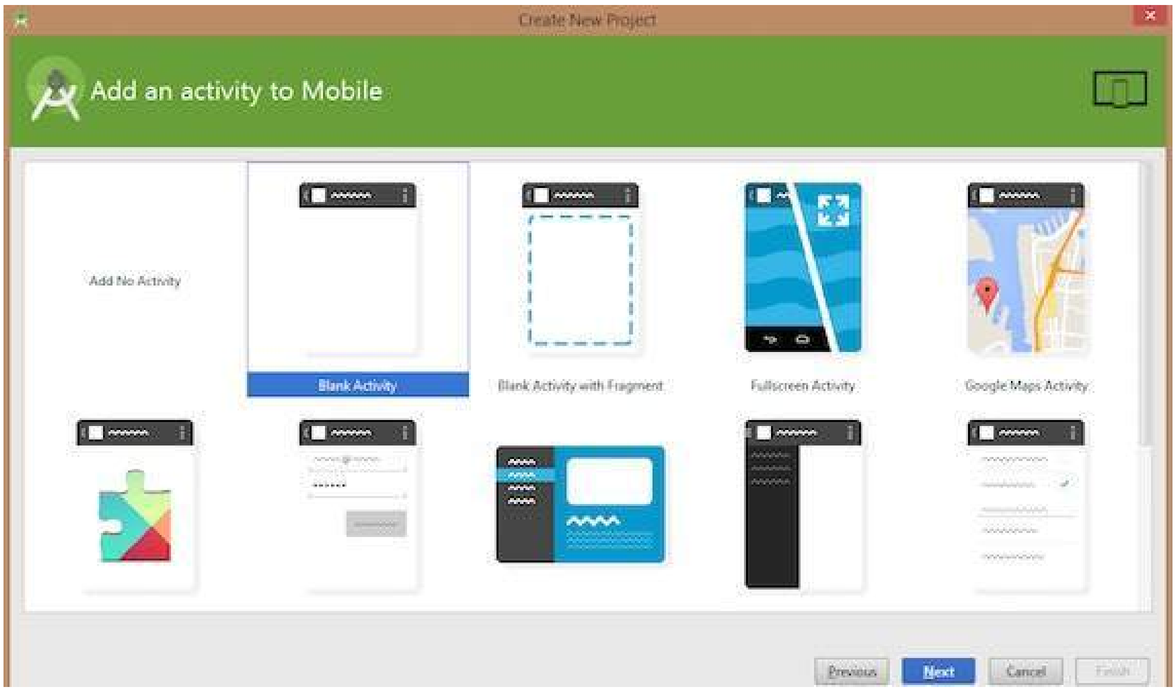
[ ] Wear
Minimum SDK: API 21: Android 5.0 (Lollipop)

[ ] TV
Minimum SDK: API 21: Android 5.0 (Lollipop)

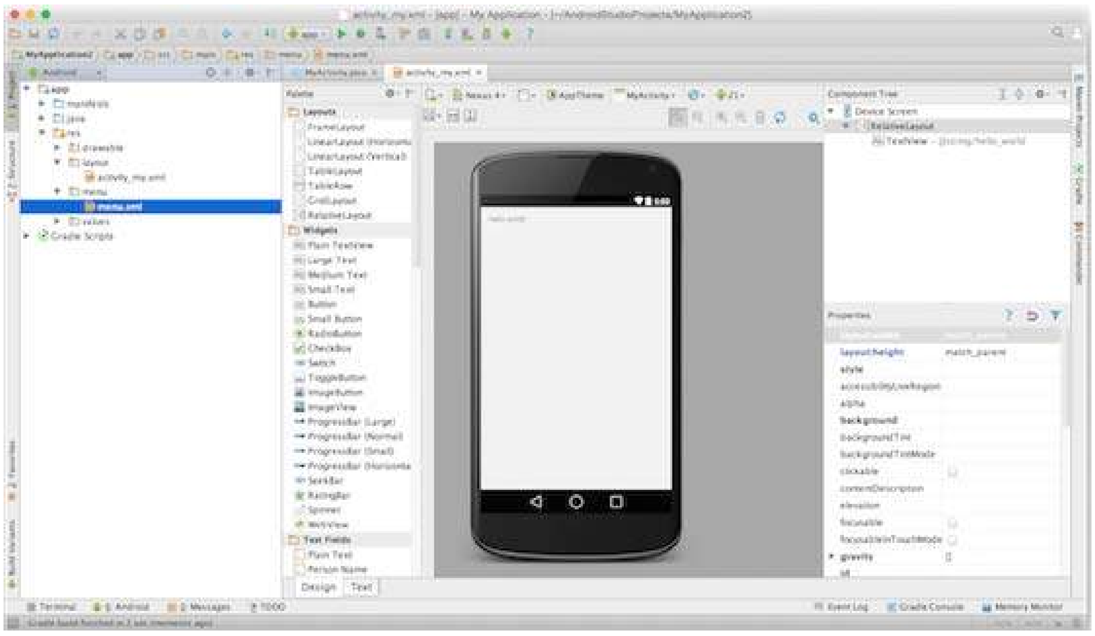
[ ] Android Auto

[ ] Glass
Minimum SDK: Glass Development Kit Preview (API 19)

The next level of installation should contain selecting the activity to mobile, it specifies the default layout for Applications.

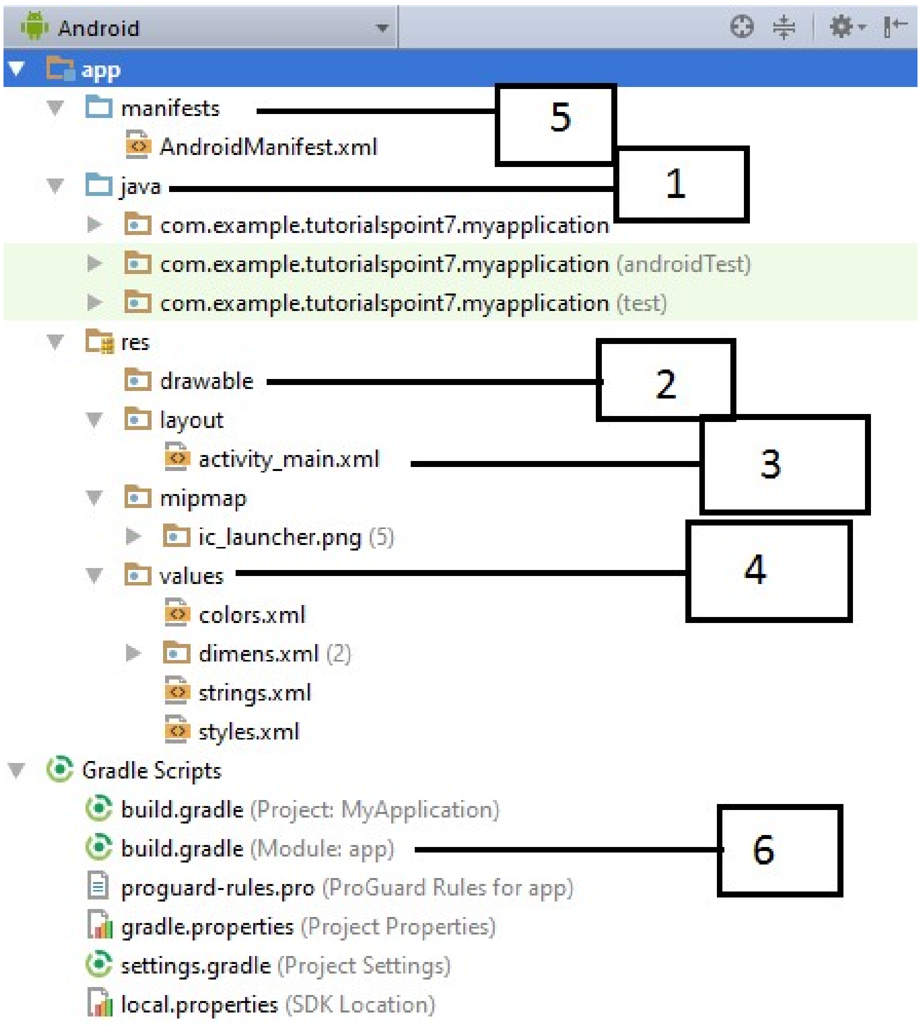


At the final stage it going to be open development tool to write the application code.



Anatomy of Android Application

Before you run your app, you should be aware of a few directories and files in the Android project –



```
Android
▼ app
    ▼ manifests
        AndroidManifest.xml
    ▼ java
        ▶ com.example.tutorialspoint7.myapplication
        ▶ com.example.tutorialspoint7.myapplication (androidTest)
        ▶ com.example.tutorialspoint7.myapplication (test)
    ▼ res
        drawable
        ▼ layout
            activity_main.xml
        ▼ mipmap
            ▶ ic_launcher.png (5)
        ▼ values
            colors.xml
            ▶ dimens.xml (2)
            strings.xml
            styles.xml
▼ Gradle Scripts
    build.gradle (Project: MyApplication)
    build.gradle (Module: app)
    proguard-rules.pro (ProGuard Rules for app)
    gradle.properties (Project Properties)
    settings.gradle (Project Settings)
    local.properties (SDK Location)
```

Sr.No.	Folder, File & Description
1	**Java** This contains the **.java** source files for your project. By default, it includes an *MainActivity.java* source file having an activity class that runs when your app is launched using the app icon.
2	**res/drawable-hdpi** This is a directory for drawable objects that are designed for high-density screens.
3	**res/layout** This is a directory for files that define your app's user interface.
4	**res/values** This is a directory for other various XML files that contain a collection of resources, such as strings and colours definitions.
5	**AndroidManifest.xml** This is the manifest file which describes the fundamental characteristics of the app and defines each of its components.
6	**Build.gradle** This is an auto generated file which contains compileSdkVersion, buildToolsVersion, applicationId, minSdkVersion, targetSdkVersion, versionCode and versionName

Following section will give a brief overview of the important application files.

The Main Activity File

The main activity code is a Java file **MainActivity.java**. This is the actual application file which ultimately gets converted to a Dalvik executable and runs your application. Following is the default code generated by the application wizard for *Hello World!* application −

```
package com.example.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
}
```

Here, *R.layout.activity_main* refers to the *activity_main.xml* file located in the *res/layout* folder. The *onCreate()* method is one of many methods that are figured when an activity is loaded.

The Manifest File

Whatever component you develop as a part of your application, you must declare all its components in a *manifest.xml* which resides at the root of the application project directory. This file works as an interface between Android OS and your application, so if you do not declare your component in this file, then it will not be considered by the OS. For example, a default manifest file will look like as following file −

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.tutorialspoint7.myapplication">

   <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:supportsRtl="true"
      android:theme="@style/AppTheme">

      <activity android:name=".MainActivity">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>
</manifest>
```

Here <application>...</application> tags enclosed the components related to the application. Attribute *android:icon* will point to the application icon available under *res/drawable-hdpi*. The application uses the image named ic_launcher.png located in the drawable folders

The <activity> tag is used to specify an activity and *android:name* attribute specifies the fully qualified class name of the *Activity* subclass and the *android:label* attributes specifies a string to use as the label for the activity. You can specify multiple activities using <activity> tags.

The **action** for the intent filter is named *android.intent.action.MAIN* to indicate that this activity serves as the entry point for the application. The **category** for the intent-filter is named *android.intent.category.LAUNCHER* to indicate that the application can be launched from the device's launcher icon.

The *@string* refers to the *strings.xml* file explained below. Hence, *@string/app_name* refers to the *app_name* string defined in the strings.xml file, which is "HelloWorld". Similar way, other strings get populated in the application.

Following is the list of tags which you will use in your manifest file to specify different Android application components −

- <activity>elements for activities
- <service> elements for services
- <receiver> elements for broadcast receivers
- <provider> elements for content providers

The Strings File

The **strings.xml** file is located in the *res/values* folder and it contains all the text that your application uses. For example, the names of buttons, labels, default text, and similar types of strings go into this file. This file is responsible for their textual content. For example, a default strings file will look like as following file −

```
<resources>
   <string name="app_name">HelloWorld</string>
   <string name="hello_world">Hello world!</string>
   <string name="menu_settings">Settings</string>
   <string name="title_activity_main">MainActivity</string>
</resources>
```

The Layout File

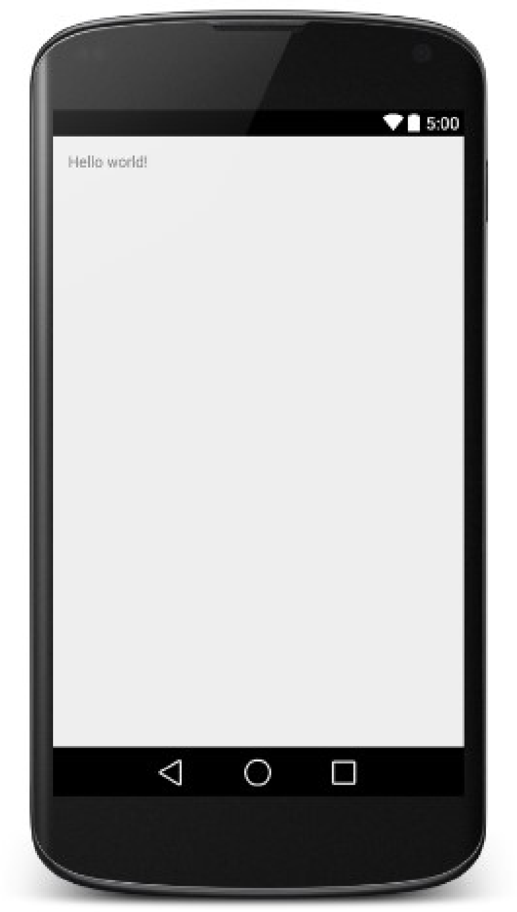
The **activity_main.xml** is a layout file available in *res/layout* directory, that is referenced by your application when building its interface. You will modify this file very frequently to change the layout of your application. For your "Hello World!" application, this file will have following content related to default layout −

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent" >

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerHorizontal="true"
      android:layout_centerVertical="true"
      android:padding="@dimen/padding_medium"
      android:text="@string/hello_world"
      tools:context=".MainActivity" />

</RelativeLayout>
```

This is an example of simple *RelativeLayout* which we will study in a separate chapter. The *TextView* is an Android control used to build the GUI and it have various attributes like *android:layout_width*, *android:layout_height* etc which are being used to set its width and height etc.. The *@string* refers to the strings.xml file located in the res/values folder. Hence, @string/hello_world refers to the hello string defined in the strings.xml file, which is "Hello World!".

Running the Application

Let's try to run our **Hello World!** application we just created. I assume you had created your **AVD** while doing environment set-up. To run the app from Android studio, open one of your project's activity files and click Run icon from the tool bar. Android studio installs the app on your AVD and starts it and if everything is fine with your set-up and application, it will display following Emulator window −



Congratulations!!! you have developed your first Android Application and now just keep following rest of the tutorial step by step to become a great Android Developer. All the very best.