ANDROID

**Introduction**

**A**ndroid is an open source and Linux-based operating system for the devices such as smartphones, wear, TV’s, tablet computers. Android was developed by the Open Handset Alliance, led by Google, and other companies. Open Handset Alliance is a consortium of 84 companies which was established on 5th November, 2007. It is committed to advance open standards, provide services and deploy handsets using the Android Platform. 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. Android applications are usually developed in the Java language using the Android Software Development Kit (SDK). The first beta version of the Android Software Development Kit was released by Google in 2007 where as the first commercial version, Android 1.0, was released in September 2008.

**History of Android**

The code names of android ranges from A to M currently, such as Petit Four, Cupcake, Donut, Éclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean, Kit Kat, Lollipop, Marshmallow.

**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 | Kit Kat 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 | ÉCLAIR |  |
| Android 1.6 | 4 | DONUT |  |
| Android 1.5 | 3 | CUPCAKE |  |
| Android 1.1 | 2 | PETIT FOUR |  |
| Android 1.0 | 1 | NO CODE NAME |  |

**Features of Android**

**A**ndroid is a powerful operating system competing with Apple 4GS and supports great features. Few of them are listed below:

|  |  |
| --- | --- |
| **Feature** | **Description** |
| Beautiful UI | Android OS basic screen provides a beautiful and intuitive user interface. |
| Connectivity | GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX. |
| Storage | SQLite, a lightweight relational database, is used for data storage purposes. |
| 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 |
| Messaging | SMS and MMS |
| Web browser | Based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine supporting HTML5 and CSS3. |
| Multi-touch | Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero. |
| Multi-tasking | User can jump from one task to another and same time various application can run simultaneously. |
| Resizable widgets | Widgets are resizable, so users can expand them to show more content or shrink them to save space |
| Multi-Language | Supports single direction and bi-directional text. |
| 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. |
| Wi-Fi Direct | A technology that lets apps discover and pair directly, over a high-bandwidth peer-to-peer connection. |
| Android Beam | A popular NFC-based technology that lets users instantly share, just by touching two NFC-enabled phones together. |

**Android Project Files**

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

1. .idea

Directory for IntelliJ IDEA settings.

1. app

Application module directories and files.

1. build

This directory stores the build output for all project modules.

1. gradle

Contains the gradler-wrapper files.

1. .gitignore

Specifies the untracked files that Git should ignore.

1. build.gradle

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

1. gradle.properties

Project-wide Gradle settings

1. gradlew

Gradle startup script for Unix

1. gradlew.bat

Gradle startup script for Windows.

1. local.properties

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

1. .iml

Module file created by the IntelliJ IDEA to store module information

1. settings.gradle

Specifies the sub-projects to build.

**Android Application Modules**

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

1. build/

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

1. libs/

Contains private libraries. Stored in the main application module

1. src/

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

1. androidTest/

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

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

Contains Java code source for the app activities

1. main/jni/

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

1. main/gen/

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

1. main/assets/

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

1. main/res/

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

1. anim/

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

1. color/

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

1. drawable/

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

1. mipmap/

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

1. layout/

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

1. menu/

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

1. raw/

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

1. values/

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

1. xml/

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

1. AndroidManifest.xml

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

1. .gitignore/

Specifies the untracked files ignored by git.

1. app.iml/

IntelliJ IDEA module

1. build.gradle

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

1. proguard-rules.pro

ProGuard settings file.

**App Components**

**A**pp components are the essential building blocks of an Android app. Each component is a different point through which the system can enter your app. Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific role—each one is a unique building block that helps define app's overall behavior.

There are four different types of app components. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.

**Here are the four types of app components:**

*Activities*

**A**n *activity* represents a single screen with a user interface. For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email app, each one is independent of the others. As such, a different app can start any one of these activities (if the email app allows it). For example, a camera app can start the activity in the email app that composes new mail, in order for the user to share a picture.An activity is implemented as a subclass of [Activity](http://developer.android.com/reference/android/app/Activity.html).

*Services*

**A** *service* is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different app, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it.A service is implemented as a subclass of [Service](http://developer.android.com/reference/android/app/Service.html).

*Content providers*

**A** *content provider* manages a shared set of app data. You can store the data in the file system, a SQLite database, on the web, or any other persistent storage location your app can access. Through the content provider, other apps can query or even modify the data (if the content provider allows it). For example, the Android system provides a content provider that manages the user's contact information. As such, any app with the proper permissions can query part of the content provider (such as [ContactsContract.Data](http://developer.android.com/reference/android/provider/ContactsContract.Data.html) ) to read and write information about a particular person.

Content providers are also useful for reading and writing data that is private to your app and not shared. For example, the [Note Pad](http://developer.android.com/resources/samples/NotePad/index.html) sample app uses a content provider to save notes.

A content provider is implemented as a subclass of [ContentProvider](http://developer.android.com/reference/android/content/ContentProvider.html) and must implement a standard set of APIs that enable other apps to perform transactions.

*Broadcast receivers*

**A** *broadcast receiver* is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Apps can also initiate broadcasts—for example, to let other apps know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don't display a user interface, they may [create a status bar notification](http://developer.android.com/guide/topics/ui/notifiers/notifications.html) to alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event.A broadcast receiver is implemented as a subclass of [BroadcastReceiver](http://developer.android.com/reference/android/content/BroadcastReceiver.html) and each broadcast is delivered as an [Intent](http://developer.android.com/reference/android/content/Intent.html) object.

A unique aspect of the Android system design is that any app can start another app’s component. For example, if you want the user to capture a photo with the device camera, there's probably another app that does that and your app can use it, instead of developing an activity to capture a photo yourself. You don't need to incorporate or even link to the code from the camera app. Instead, you can simply start the activity in the camera app that captures a photo. When complete, the photo is even returned to your app so you can use it. To the user, it seems as if the camera is actually a part of your app.

When the system starts a component, it starts the process for that app (if it's not already running) and instantiates the classes needed for the component. For example, if your app starts the activity in the camera app that captures a photo, that activity runs in the process that belongs to the camera app, not in your app's process. Therefore, unlike apps on most other systems, Android apps don't have a single entry point (there's no main ( ) function, for example).

Because the system runs each app in a separate process with file permissions that restrict access to other apps, your app cannot directly activate a component from another app. The Android system, however, can. So, to activate a component in another app, you must deliver a message to the system that specifies your intent to start a particular component. The system then activates the component for you.

**Create a Project with Android Studio**

1. In Android Studio, create a new project:
   * If you don't have a project opened, in the **Welcome** screen, click **New Project**.
   * If you have a project opened, from the **File** menu, select **New Project**. The Create New Project screen appears.
2. Fill out the fields on the screen, and click **Next**.

It is easier to follow these lessons if you use the same values as shown.

* + **Application Name** is the app name that appears to users. For this project, use "My First App."
  + **Company domain** provides a qualifier that will be appended to the package name; Android Studio will remember this qualifier for each new project you create.
  + **Package name** is the fully qualified name for the project (following the same rules as those for naming packages in the Java programming language). Your package name must be unique across all packages installed on the Android system. You can **Edit** this value independently from the application name or the company domain.
  + **Project location** is the directory on your system that holds the project files.

1. Under **Select the form factors your app will run on**, check the box for **Phone and Tablet**.
2. For **Minimum SDK**, select **API 8: Android 2.2 (Froyo)**.

The Minimum Required SDK is the earliest version of Android that your app supports, indicated using the [API level](http://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels). To support as many devices as possible, you should set this to the lowest version available that allows your app to provide its core feature set. If any feature of your app is possible only on newer versions of Android and it's not critical to the app's core feature set, you can enable the feature only when running on the versions that support it (as discussed in [Supporting Different Platform Versions](http://developer.android.com/training/basics/supporting-devices/platforms.html)).

1. Leave all of the other options (TV, Wear, and Glass) unchecked and click **Next.**
2. Under **Add an activity to <template>**, select **Blank Activity** and click **Next**.
3. Under **Customize the Activity**, change the **Activity Name** toMyActivity. The **Layout Name** changes to activity\_my, and the**Title** to MyActivity. The **Menu Resource Name** is menu\_my.
4. Click the **Finish** button to create the project.

Your Android project is now a basic "Hello World" app that contains some default files.

**Create an AVD**

1. Launch the Android Virtual Device Manager:
   * In Android Studio, select **Tools > Android > AVD Manager**, or click the AVD Manager icon in the toolbar. The AVD Manager screen appears.
   * Or, from the command line, change directories to sdk/ and execute:

tools/android avd

**Note:** The AVD Manager that appears when launched from the command line is different from the version in Android Studio, so the following instructions may not all apply.

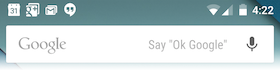
1. On the AVD Manager main screen, click **Create Virtual Device**.
2. In the Select Hardware window, select a device configuration, such as Nexus 6, then click **Next**.
3. Select the desired system version for the AVD and click **Next**.
4. Verify the configuration settings, then click **Finish**.

**Run the app from Android Studio**

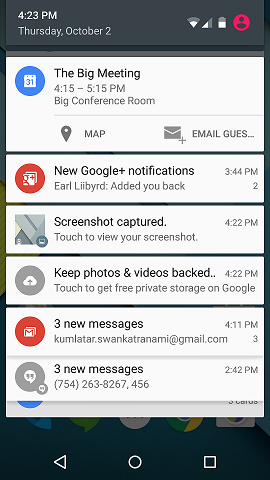
1. In **Android Studio**, select your project and click **Run** from the toolbar.
2. In the **Choose Device** window, click the **Launch emulator** radio button.
3. From the **Android virtual device** pull-down menu, select the emulator you created, and click **OK**.

Notifications

**A** notification is a message you can display to the user outside of your application's normal UI. The notification system allows users to keep informed about relevant and timely events in your app, such as new chat messages from a friend or a calendar event. Think of notifications as a news channel that alerts the user to important events as they happen or a log that chronicles events while the user is not paying attention. When you tell the system to issue a notification, it first appears as an icon in the **notification area**. To see the details of the notification, the user opens the **notification drawer**. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.



**Img : notification area**

****

**Img : notification drawer**

**Creating a Notification**

**Y**ou specify the UI information and actions for a notification in a [NotificationCompat.Builder](http://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html) object.To create the notification itself, you call [NotificationCompat.Builder.build()](http://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html#build()), which returns a [Notification](http://developer.android.com/reference/android/app/Notification.html) object containing your specifications. To issue the notification, you pass the [Notification](http://developer.android.com/reference/android/app/Notification.html) object to the system by calling [NotificationManager.notify()](http://developer.android.com/reference/java/lang/Object.html#notify()).

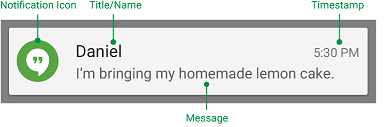
**Required notification contents**

**A** [Notification](http://developer.android.com/reference/android/app/Notification.html) object must contain the following:

* A small icon, set by [setSmallIcon()](http://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html#setSmallIcon(int))
* A title, set by [setContentTitle()](http://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html#setContentTitle(java.lang.CharSequence))
* Detail text, set by [setContentText()](http://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html#setContentText(java.lang.CharSequence))

At a minimum, all notifications consist of following:

* The notification's **icon**. The icon symbolizes the originating app. It may also potentially indicate notification type if the app generates more than one type.
* A notification **title** and additional **text**.
* A **timestamp**.



**Layouts**

**A** layout defines the visual structure for a user interface, such as the UI for an [activity](http://developer.android.com/guide/components/activities.html) or [app widget](http://developer.android.com/guide/topics/appwidgets/index.html)

**Common Layouts**

**E**ach subclass of the [ViewGroup](http://developer.android.com/reference/android/view/ViewGroup.html) class provides a unique way to display the views you nest within it. Below are some of the more common layout types that are built into the Android platform.

**Note:** Although you can nest one or more layouts within another layout to acheive your UI design, you should strive to keep your layout hierarchy as shallow as possible. Your layout draws faster if it has fewer nested layouts (a wide view hierarchy is better than a deep view hierarchy).

#### [Linear Layout](http://developer.android.com/guide/topics/ui/layout/linear.html)

**A** layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.



#### [Relative Layout](http://developer.android.com/guide/topics/ui/layout/relative.html)

**E**nables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).



#### [List View](http://developer.android.com/guide/topics/ui/layout/listview.html)

**D**isplays a scrolling single column list.



#### [Grid View](http://developer.android.com/guide/topics/ui/layout/gridview.html)

**D**isplays a scrolling grid of columns and rows.

