Android

**Android** is a complete set of software for mobile devices such as tablet computers, notebooks, smartphones, electronic book readers, set-top boxes etc.

It contains a **linux-based Operating System**, **middleware** and **key mobile applications**.

It can be thought of as a mobile operating system. But it is not limited to mobile only. It is currently used in various devices such as mobiles, tablets, televisions etc.

What is Android

**Android** is a software package and linux based operating system for mobile devices such as tablet computers and smartphones.

It is developed by Google and later the OHA (Open Handset Alliance). Java language is mainly used to write the android code even though other languages can be used.

The goal of android project is to create a successful real-world product that improves the mobile experience for end users.

There are many code names of android such as Lollipop, Kitkat, Jelly Bean, Ice cream Sandwich, Froyo, Ecliar, Donut etc which is covered in next page.

## What is Open Handset Alliance (OHA)

It's a consortium of 84 companies such as google, samsung, AKM, synaptics, KDDI, Garmin, Teleca, Ebay, Intel etc.

It was established on 5th November, 2007, led by Google. It is committed to advance open standards, provide services and deploy handsets using the Android Plateform.

## Features of Android

After learning what is android, let's see the features of android. The important features of android are given below:

1) It is open-source.

2) Anyone can customize the Android Platform.

3) There are a lot of mobile applications that can be chosen by the consumer. Rich UI design.

4) It provides many interesting features like weather details, opening screen, live RSS (Really Simple Syndication) feeds etc.

It provides support for messaging services(SMS and MMS), web browser, storage (SQLite), connectivity (GSM, CDMA, Blue Tooth, Wi-Fi etc.), media, handset layout etc.

# History of Android

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

1) Initially, **Andy Rubin** founded Android Incorporation in Palo Alto, California, United States in October, 2003.

2) In 17th August 2005, Google acquired android Incorporation. Since then, it is in the subsidiary of Google Incorporation.

3) The key employees of Android Incorporation are **Andy Rubin**, **Rich Miner**, **Chris White** and **Nick Sears**.

4) Originally intended for camera but shifted to smart phones later because of low market for camera only.

5) Android is the nick name of Andy Rubin given by coworkers because of his love to robots.

6) In 2007, Google announces the development of android OS.

7) In 2008, HTC launched the first android mobile.

## 

## Android Versions, Codename and API

Let's see the android versions, codenames and API Level provided by Google.

|  |  |  |
| --- | --- | --- |
| **Version** | **Code name** | **API Level** |
| 1.5 | Cupcake | 3 |
| 1.6 | Donut | 4 |
| 2.1 | Eclair | 7 |
| 2.2 | Froyo | 8 |
| 2.3 | Gingerbread | 9 and 10 |
| 3.1 and 3.3 | Honeycomb | 12 and 13 |
| 4.0 | Ice Cream Sandwitch | 15 |
| 4.1, 4.2 and 4.3 | Jelly Bean | 16, 17 and 18 |
| 4.4 | KitKat | 19 |
| 5.0 | Lollipop | 21 |

## Android Project Files

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

.idea

Directory for IntelliJ IDEA settings.

app

Application module directories and files.

build

This directory stores the build output for all project modules.

gradle

Contains the gradler-wrapper files.

.gitignore

Specifies the untracked files that Git should ignore.

build.gradle

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

gradle.properties

Project-wide Gradle settings.

gradlew

Gradle startup script for Unix.

gradlew.bat

Gradle startup script for Windows.

local.properties

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

.iml

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

settings.gradle

Specifies the sub-projects to build.

Project modules

build/

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

libs/

Contains private libraries. Stored in the main application module.

src/

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

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

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

Contains Java code source for the app activities.

main/jni/

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

main/gen/

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

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.

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.

anim/

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

color/

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

drawable/

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

mipmap/

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

layout/

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

menu/

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

raw/

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

values/

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

xml/

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

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

.gitignore/

Specifies the untracked files ignored by git.

app.iml/

IntelliJ IDEA module

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.

Android - Application Components

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:

|  |  |
| --- | --- |
| **Components** | **Description** |
| Activities | They dictate the UI and handle the user interaction to the smart phone screen |
| Services | They handle background processing associated with an application. |
| Broadcast Receivers | They handle communication between Android OS and applications. |
| 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.

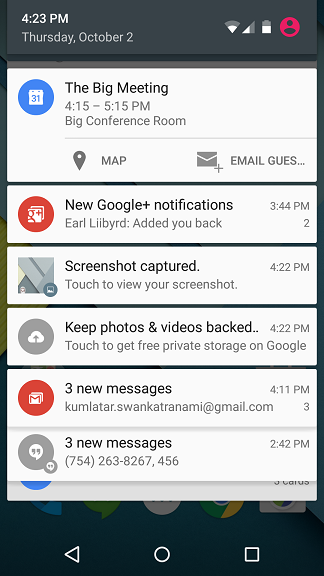
## Additional Components

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

|  |  |
| --- | --- |
| **Components** | **Description** |
| Fragments | Represents a portion of user interface in an Activity. |
| Views | UI elements that are drawn on-screen including buttons, lists forms etc. |
| Layouts | View hierarchies that control screen format and appearance of the views. |
| Intents | Messages wiring components together. |
| Resources | External elements, such as strings, constants and drawable pictures. |
| Manifest | Configuration file for the application. |

Notifications

A notification is a message you can display to the user outside of your application's normal UI. 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.Example:-



## Creating a Notification

You 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))

### Optional notification contents and settings

All other notification settings and contents are optional. To learn more about them, see the reference documentation for [NotificationCompat.Builder](http://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html).

### Notification actions

Although they're optional, you should add at least one action to your notification. An action allows users to go directly from the notification to an [Activity](http://developer.android.com/reference/android/app/Activity.html) in your application, where they can look at one or more events or do further work.

A notification can provide multiple actions. You should always define the action that's triggered when the user clicks the notification; usually this action opens an [Activity](http://developer.android.com/reference/android/app/Activity.html) in your application. You can also add buttons to the notification that perform additional actions such as snoozing an alarm or responding immediately to a text message; this feature is available as of Android 4.1. If you use additional action buttons, you must also make their functionality available in an [Activity](http://developer.android.com/reference/android/app/Activity.html) in your app; see the section [Handling compatibility](http://developer.android.com/guide/topics/ui/notifiers/notifications.html#Compatibility) for more details.

Inside a [Notification](http://developer.android.com/reference/android/app/Notification.html), the action itself is defined by a [PendingIntent](http://developer.android.com/reference/android/app/PendingIntent.html) containing an [Intent](http://developer.android.com/reference/android/content/Intent.html) that starts an[Activity](http://developer.android.com/reference/android/app/Activity.html) in your application. To associate the [PendingIntent](http://developer.android.com/reference/android/app/PendingIntent.html) with a gesture, call the appropriate method of[NotificationCompat.Builder](http://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html). For example, if you want to start [Activity](http://developer.android.com/reference/android/app/Activity.html) when the user clicks the notification text in the notification drawer, you add the [PendingIntent](http://developer.android.com/reference/android/app/PendingIntent.html) by calling [setContentIntent()](http://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html#setContentIntent(android.app.PendingIntent)).

Starting an [Activity](http://developer.android.com/reference/android/app/Activity.html) when the user clicks the notification is the most common action scenario. You can also start an [Activity](http://developer.android.com/reference/android/app/Activity.html) when the user dismisses a notification. In Android 4.1 and later, you can start an [Activity](http://developer.android.com/reference/android/app/Activity.html)from an action button. To learn more, read the reference guide for [NotificationCompat.Builder](http://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html).

NotificationCompat.Builder mBuilder =  
        new NotificationCompat.Builder(this)  
        .setSmallIcon(R.drawable.notification\_icon)  
        .setContentTitle("My notification")  
        .setContentText("Hello World!");  
// Creates an explicit intent for an Activity in your app  
Intent resultIntent = new Intent(this, ResultActivity.class);  
  
// The stack builder object will contain an artificial back stack for the  
// started Activity.  
// This ensures that navigating backward from the Activity leads out of  
// your application to the Home screen.  
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);  
// Adds the back stack for the Intent (but not the Intent itself)  
stackBuilder.addParentStack(ResultActivity.class);  
// Adds the Intent that starts the Activity to the top of the stack  
stackBuilder.addNextIntent(resultIntent);  
PendingIntent resultPendingIntent =  
        stackBuilder.getPendingIntent(  
            0,  
            PendingIntent.FLAG\_UPDATE\_CURRENT  
        );  
mBuilder.setContentIntent(resultPendingIntent);  
NotificationManager mNotificationManager =  
    (NotificationManager) getSystemService(Context.NOTIFICATION\_SERVICE);  
// mId allows you to update the notification later on.  
mNotificationManager.notify(mId, mBuilder.build());

### Handling compatibility

Not all notification features are available for a particular version, even though the methods to set them are in the support library class [NotificationCompat.Builder](http://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html). For example, action buttons, which depend on expanded notifications, only appear on Android 4.1 and higher, because expanded notifications themselves are only available on Android 4.1 and higher.

To ensure the best compatibility, create notifications with [NotificationCompat](http://developer.android.com/reference/android/support/v4/app/NotificationCompat.html) and its subclasses, particularly[NotificationCompat.Builder](http://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html). In addition, follow this process when you implement a notification:

1. Provide all of the notification's functionality to all users, regardless of the version they're using. To do this, verify that all of the functionality is available from an [Activity](http://developer.android.com/reference/android/app/Activity.html) in your app. You may want to add a new[Activity](http://developer.android.com/reference/android/app/Activity.html) to do this.

For example, if you want to use [addAction()](http://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html#addAction(android.support.v4.app.NotificationCompat.Action)) to provide a control that stops and starts media playback, first implement this control in an [Activity](http://developer.android.com/reference/android/app/Activity.html) in your app.

1. Ensure that all users can get to the functionality in the [Activity](http://developer.android.com/reference/android/app/Activity.html), by having it start when users click the notification. To do this, create a [PendingIntent](http://developer.android.com/reference/android/app/PendingIntent.html) for the [Activity](http://developer.android.com/reference/android/app/Activity.html). Call [setContentIntent()](http://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html#setContentIntent(android.app.PendingIntent)) to add the[PendingIntent](http://developer.android.com/reference/android/app/PendingIntent.html) to the notification.
2. Now add the expanded notification features you want to use to the notification. Remember that any functionality you add also has to be available in the [Activity](http://developer.android.com/reference/android/app/Activity.html) that starts when users click the notification.

### Removing notifications

Notifications remain visible until one of the following happens:

* The user dismisses the notification either individually or by using "Clear All" (if the notification can be cleared).
* The user clicks the notification, and you called [setAutoCancel()](http://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html#setAutoCancel(boolean)) when you created the notification.
* You call [cancel()](http://developer.android.com/reference/android/app/NotificationManager.html#cancel(int)) for a specific notification ID. This method also deletes ongoing notifications.
* You call [cancelAll()](http://developer.android.com/reference/android/app/NotificationManager.html#cancelAll()), which removes all of the notifications you previously issued.

VIEW

The basic building block for user interface is a **View** object which is created from the View class and occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for widgets, which are used to create interactive UI components like buttons, text fields, etc.

The **ViewGroup** is a subclass of **View** and provides invisible container that hold other Views or other ViewGroups and define their layout properties.

At third level we have different layouts which are subclasses of ViewGroup class and a typical layout defines the visual structure for an Android user interface and can be created either at run time using **View/ViewGroup** objects or you can declare your layout using simple XML file **main\_layout.xml** which is located in the res/layout folder of your project.

#### LAYOUT PARAMS

This tutorial is more about creating your GUI based on layouts defined in XML file. A layout may contain any type of widgets such as buttons, labels, textboxes, and so on. Following is a simple example of XML file having LinearLayout:

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:orientation="vertical" >

<TextView android:id="@+id/text"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="This is a TextView" />

<Button android:id="@+id/button"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="This is a Button" />

<!-- More GUI components go here -->

</LinearLayout>

Once your layout has created, you can load the layout resource from your application code, in your *Activity.onCreate()* callback implementation as shown below −

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

## Android Layout Types

There are number of Layouts provided by Android which you will use in almost all the Android applications to provide different view, look and feel.

|  |  |
| --- | --- |
| **Sr.No** | **Layout & Description** |
| 1 | [**Linear Layout**](http://www.tutorialspoint.com/android/android_linear_layout.htm)  LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally. |
| 2 | [**Relative Layout**](http://www.tutorialspoint.com/android/android_relative_layout.htm)  RelativeLayout is a view group that displays child views in relative positions. |
| 3 | [**Table Layout**](http://www.tutorialspoint.com/android/android_table_layout.htm)  TableLayout is a view that groups views into rows and columns. |
| 4 | [**Absolute Layout**](http://www.tutorialspoint.com/android/android_absolute_layout.htm)  AbsoluteLayout enables you to specify the exact location of its children. |
| 5 | [**Frame Layout**](http://www.tutorialspoint.com/android/android_frame_layout.htm)  The FrameLayout is a placeholder on screen that you can use to display a single view. |
| 6 | [**List View**](http://www.tutorialspoint.com/android/android_list_view.htm)  ListView is a view group that displays a list of scrollable items. |
| 7 | [**Grid View**](http://www.tutorialspoint.com/android/android_grid_view.htm)  GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid. |

## Layout Attributes

Each layout has a set of attributes which define the visual properties of that layout. There are few common attributes among all the layouts and their are other attributes which are specific to that layout. Following are common attributes and will be applied to all the layouts:

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| android:id | This is the ID which uniquely identifies the view. |
| android:layout\_width | This is the width of the layout. |
| android:layout\_height | This is the height of the layout |
| android:layout\_marginTop | This is the extra space on the top side of the layout. |
| android:layout\_marginBottom | This is the extra space on the bottom side of the layout. |
| android:layout\_marginLeft | This is the extra space on the left side of the layout. |
| android:layout\_marginRight | This is the extra space on the right side of the layout. |
| android:layout\_gravity | This specifies how child Views are positioned. |
| android:layout\_weight | This specifies how much of the extra space in the layout should be allocated to the View. |
| android:layout\_x | This specifies the x-coordinate of the layout. |
| android:layout\_y | This specifies the y-coordinate of the layout. |
| android:layout\_width | This is the width of the layout. |
| android:layout\_width | This is the width of the layout. |
| android:paddingLeft | This is the left padding filled for the layout. |
| android:paddingRight | This is the right padding filled for the layout. |
| android:paddingTop | This is the top padding filled for the layout. |
| android:paddingBottom | This is the bottom padding filled for the layout. |

Here width and height are the dimension of the layout/view which can be specified in terms of dp (Density-independent Pixels), sp ( Scale-independent Pixels), pt ( Points which is 1/72 of an inch), px( Pixels), mm ( Millimeters) and finally in (inches).

You can specify width and height with exact measurements but more often, you will use one of these constants to set the width or height −

* **android:layout\_width=wrap\_content** tells your view to size itself to the dimensions required by its content.
* **android:layout\_width=match\_parent** tells your view to become as big as its parent view.

LAYOUT

This tutorial is more about creating your GUI based on layouts defined in XML file. A layout may contain any type of widgets such as buttons, labels, textboxes, and so on. Following is a simple example of XML file having LinearLayout:

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:orientation="vertical" >

<TextView android:id="@+id/text"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="This is a TextView" />

<Button android:id="@+id/button"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="This is a Button" />

<!-- More GUI components go here -->

</LinearLayout>

Once your layout has created, you can load the layout resource from your application code, in your *Activity.onCreate()* callback implementation as shown below −

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

## Android Layout Types

There are number of Layouts provided by Android which you will use in almost all the Android applications to provide different view, look and feel.

|  |  |
| --- | --- |
| **Sr.No** | **Layout & Description** |
| 1 | [**Linear Layout**](http://www.tutorialspoint.com/android/android_linear_layout.htm)  LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally. |
| 2 | [**Relative Layout**](http://www.tutorialspoint.com/android/android_relative_layout.htm)  RelativeLayout is a view group that displays child views in relative positions. |
| 3 | [**Table Layout**](http://www.tutorialspoint.com/android/android_table_layout.htm)  TableLayout is a view that groups views into rows and columns. |
| 4 | [**Absolute Layout**](http://www.tutorialspoint.com/android/android_absolute_layout.htm)  AbsoluteLayout enables you to specify the exact location of its children. |
| 5 | [**Frame Layout**](http://www.tutorialspoint.com/android/android_frame_layout.htm)  The FrameLayout is a placeholder on screen that you can use to display a single view. |
| 6 | [**List View**](http://www.tutorialspoint.com/android/android_list_view.htm)  ListView is a view group that displays a list of scrollable items. |
| 7 | [**Grid View**](http://www.tutorialspoint.com/android/android_grid_view.htm)  GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid. |

## Layout Attributes

Each layout has a set of attributes which define the visual properties of that layout. There are few common attributes among all the layouts and their are other attributes which are specific to that layout. Following are common attributes and will be applied to all the layouts:

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| android:id | This is the ID which uniquely identifies the view. |
| android:layout\_width | This is the width of the layout. |
| android:layout\_height | This is the height of the layout |
| android:layout\_marginTop | This is the extra space on the top side of the layout. |
| android:layout\_marginBottom | This is the extra space on the bottom side of the layout. |
| android:layout\_marginLeft | This is the extra space on the left side of the layout. |
| android:layout\_marginRight | This is the extra space on the right side of the layout. |
| android:layout\_gravity | This specifies how child Views are positioned. |
| android:layout\_weight | This specifies how much of the extra space in the layout should be allocated to the View. |
| android:layout\_x | This specifies the x-coordinate of the layout. |
| android:layout\_y | This specifies the y-coordinate of the layout. |
| android:layout\_width | This is the width of the layout. |
| android:layout\_width | This is the width of the layout. |
| android:paddingLeft | This is the left padding filled for the layout. |
| android:paddingRight | This is the right padding filled for the layout. |
| android:paddingTop | This is the top padding filled for the layout. |
| android:paddingBottom | This is the bottom padding filled for the layout. |

Here width and height are the dimension of the layout/view which can be specified in terms of dp (Density-independent Pixels), sp ( Scale-independent Pixels), pt ( Points which is 1/72 of an inch), px( Pixels), mm ( Millimeters) and finally in (inches).

You can specify width and height with exact measurements but more often, you will use one of these constants to set the width or height −

* **android:layout\_width=wrap\_content** tells your view to size itself to the dimensions required by its content.
* **android:layout\_width=match\_parent** tells your view to become as big as its parent view.

## Android UI Controls

There are number of UI controls provided by Android that allow you to build the graphical user interface for your app.

|  |  |
| --- | --- |
| **S.N.** | **UI Control & Description** |
| 1 | [**TextView**](http://www.tutorialspoint.com/android/android_textview_control.htm)  This control is used to display text to the user. |
| 2 | [**EditText**](http://www.tutorialspoint.com/android/android_edittext_control.htm)  EditText is a predefined subclass of TextView that includes rich editing capabilities. |
| 3 | [**AutoCompleteTextView**](http://www.tutorialspoint.com/android/android_autocompletetextview_control.htm)  The AutoCompleteTextView is a view that is similar to EditText, except that it shows a list of completion suggestions automatically while the user is typing. |
| 4 | [**Button**](http://www.tutorialspoint.com/android/android_button_control.htm)  A push-button that can be pressed, or clicked, by the user to perform an action. |
| 5 | [**ImageButton**](http://www.tutorialspoint.com/android/android_imagebutton_control.htm)  AbsoluteLayout enables you to specify the exact location of its children. |
| 6 | [**CheckBox**](http://www.tutorialspoint.com/android/android_checkbox_control.htm)  An on/off switch that can be toggled by the user. You should use check box when presenting users with a group of selectable options that are not mutually exclusive. |
| 7 | [**ToggleButton**](http://www.tutorialspoint.com/android/android_togglebutton_control.htm)  An on/off button with a light indicator. |
| 8 | [**RadioButton**](http://www.tutorialspoint.com/android/android_radiobutton_control.htm)  The RadioButton has two states: either checked or unchecked. |
| 9 | [**RadioGroup**](http://www.tutorialspoint.com/android/android_radiogroup_control.htm)  A RadioGroup is used to group together one or more RadioButtons. |
| 10 | [**ProgressBar**](http://www.tutorialspoint.com/android/android_progressbar.htm)  The ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background. |
| 11 | [**Spinner**](http://www.tutorialspoint.com/android/android_spinner_control.htm)  A drop-down list that allows users to select one value from a set. |
| 12 | [**TimePicker**](http://www.tutorialspoint.com/android/android_timepicker_control.htm)  The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode. |
| 13 | [**DatePicker**](http://www.tutorialspoint.com/android/android_datepicker_control.htm)  The DatePicker view enables users to select a date of the day. |

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

main/assetThis 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.

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.

anim/

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

color/

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

drawable/

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

mipmap/

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

layout/

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

menu/

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

raw/

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

values/

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

xml/

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

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

.gitignore/

Specifies the untracked files ignored by git.

app.iml/

IntelliJ IDEA module

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.

proguard-rules.pro

ProGuard settings file.