Mobility and Mobile App Development Landscape

Ever since the famous mobile revolution, the number of fields where mobile phones are used has been increasing at a pace no human would have predicted at the beginning. Mobile phones are everywhere, and so are mobile apps. Don't believe me? Go to the app store you prefer and browse through the variety of apps. I wouldn't be surprised if I find an app on app store using which I can mark the places I have pooped("Places I've Pooped" is the app I'm talking about). Jokes aside, there seriously is an app for almost everything! Businesses especially find it better to have an app since it helps them in a lot many ways. To help you understand the current mobility landscape, as well as that of the future, let's dig into what both have to offer!

Current Mobility and Mobile App Development Landscape

The current situation is pretty impressive and you can clearly see that we have come a long way from the very first apps that came around in the market. We have been progressing constantly and have come forth with mind-blowing technologies. Technologies such as Google Home and Apple Homepod were something straight out of Sci-Fi not so long ago. We still use rather primitive languages such as JAVA, C, and C++ for development, but the scenario is changing and now we have started using new and upcoming languages for developing state-of-the-art mobile apps. In fact, technologies are molding the industry along with them. At the end of 2016, AMP was still a technology not known by many. But by February 2017, AMP Pages accounted for about seven percent of the web traffic of top US publishers. The progress is similar in other fields of mobility domain. The cloud is another domain which has seen a lot of growth(in terms of security though). More and more people are turning to cloud, thanks to the improving situation of internet connectivity all over the world. People now prefer to keep their images and videos on cloud storage. Moreover, music streaming apps are used by people which is removing the need for them to download music. This additionally helps in reducing piracy which was a widespread concern for the music industry a while back.

Future of Mobility

The future of mobility lies in Artificial Intelligence, Augmented Reality and Virtual Reality. Recently, Facebook shut its AI project because it went a bit overboard. Such is the potential that AI holds in the future landscape of mobility and mobile app development. AI has been used extensively, but not exclusively, for chatbots until now. This has been limiting its potential but now that it is being used in other domains, the true potential can be unleashed and I'm looking forward to seeing what AI would look like in the future. AR and VR are already well-discussed topics. I have covered a wee bit of information on my blog regarding the same. Do have a look at it(Psst! It's free!) Other wearable alternatives to general devices are going to get a big boost in the future. It wouldn't be too surprising if, in the future, we turn to wearable devices more than

mobile phones. Imagine a world where people don't lose their mobile phones. Awesome, right? So is the future of mobility, so ride the wave with mobility experts!

Mobile platforms

Presently a day's *mobile application development* is turning out to be increasingly popular. Many individuals are utilizing cell phones like iPhone, Android and Blackberry. Presently every ventures are demonstrating more enthusiasm for creating mobile applications for their business, to expand more clients and income to the business. They are creating mobile applications for their business to draw in more clients. *Mobile application development companies* develop mobile applications on different platforms in view of their client require. However, the significant platforms are android, iOS, Cross-platform and web platform and beneath are very much utilized as a part of past years.

iOS:

- You can use Xcode and Swift
- Apple Developer

Android:

- You can use Android Studio and Java
- Android Developers

Windows:

- You can use Xamarin
- Mobile App Development & App Creation Software

Cross Platform:

- Ionic
- Cordova
- Xamarin
- React Native

What is Android?

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.

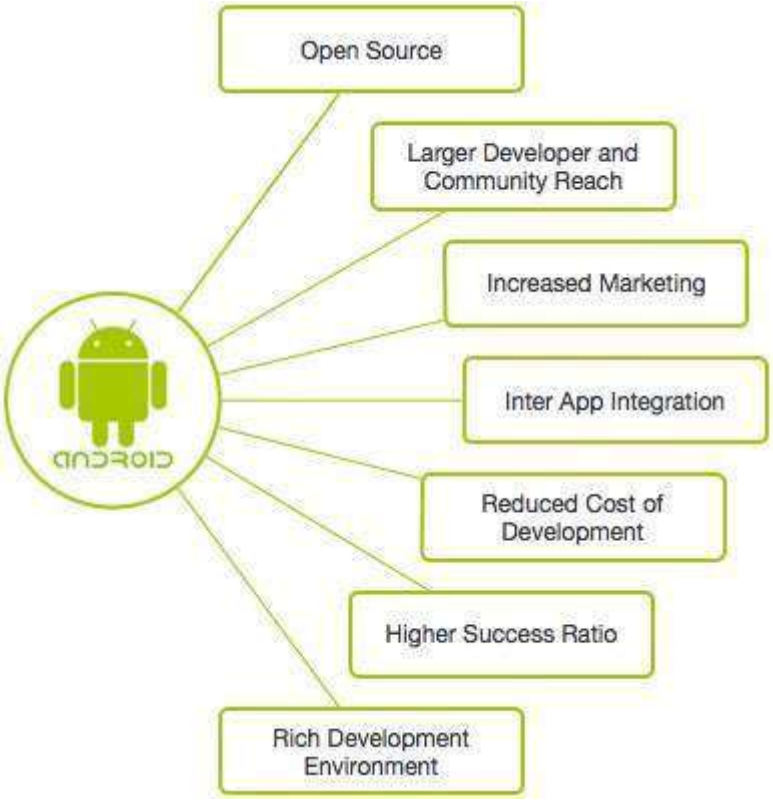
Android offers a unified approach to application development for mobile devices which means developers need only develop for Android, and their applications should be able to run on different devices powered by Android.

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.

Why 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.

Android - Environment Setup

You will be glad to know that you can start your Android application development on either of the following operating systems −

- Microsoft Windows XP or later version.
- Mac OS X 10.5.8 or later version with Intel chip.
- Linux including GNU C Library 2.7 or later.

Second point is that all the required tools to develop Android applications are freely available and can be downloaded from the Web. Following is the list of software's you will need before you start your Android application programming.

- Java JDK5 or later version
- Android Studio

Here last two components are optional and if you are working on Windows machine then these components make your life easy while doing Java based application development. So let us have a look how to proceed to set required environment.

Set-up Java Development Kit (JDK)

You can download the latest version of Java JDK from Oracle's Java site − Java SE Downloads.

You will find instructions for installing JDK in downloaded files, follow the given instructions to install and configure the setup. Finally set PATH and JAVA_HOME environment variables to refer to the directory that contains **java** and **javac**, typically java_install_dir/bin and java_install_dir respectively.

If you are running Windows and installed the JDK in C:\jdk1.8.0_102, you would have to put the following line in your C:\autoexec.bat file.

```
set PATH=C:\jdk1.8.0_102\bin;%PATH%
set JAVA_HOME=C:\jdk1.8.0_102
```

Alternatively, you could also right-click on *My Computer*, select *Properties*, then *Advanced*, then *Environment Variables*. Then, you would update the PATH value and press the OK button.

On Linux, if the SDK is installed in /usr/local/jdk1.8.0_102 and you use the C shell, you would put the following code into your **.cshrc** file.

```
setenv PATH /usr/local/jdk1.8.0_102/bin:$PATH
setenv JAVA_HOME /usr/local/jdk1.8.0_102
```

Alternatively, if you use Android studio, then it will know automatically where you have installed your Java.

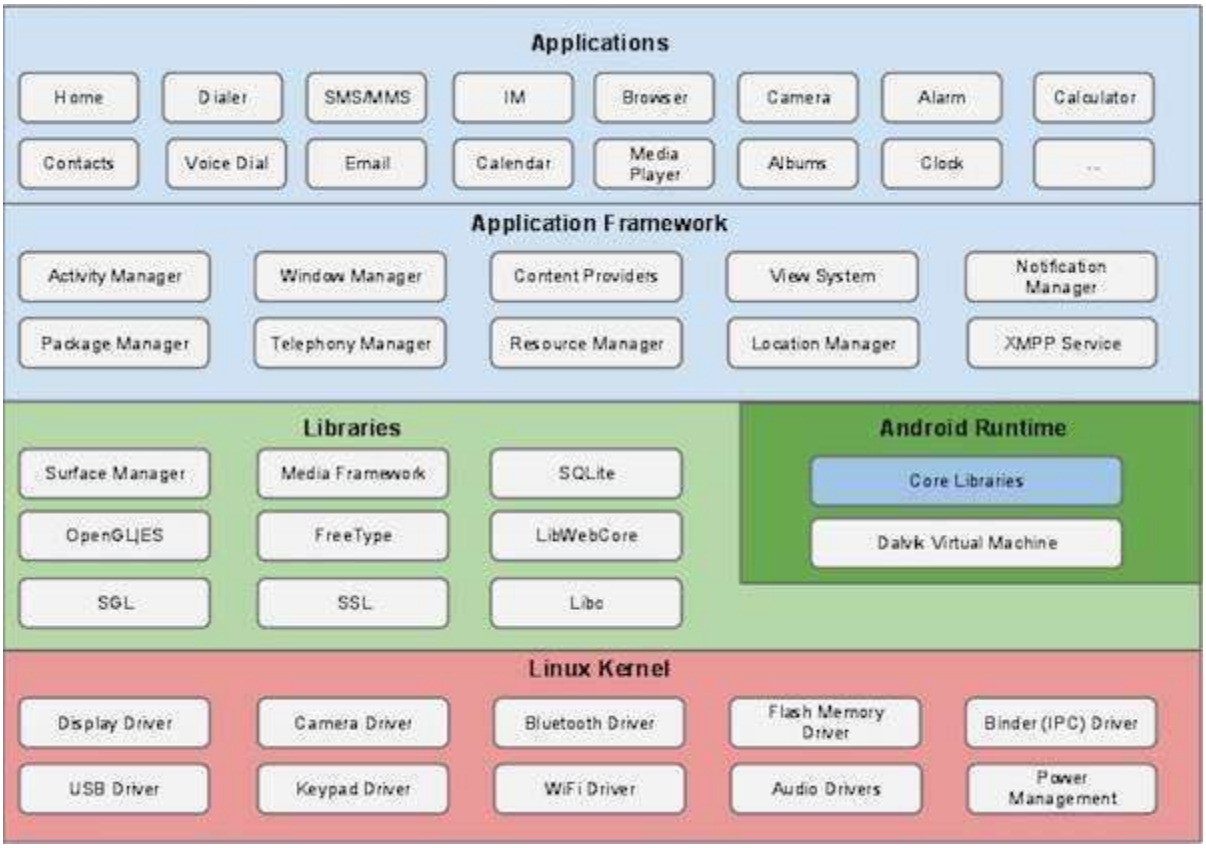
Android IDEs

There are so many sophisticated Technologies are available to develop android applications, the familiar technologies, which are predominantly using tools as follows

- Android Studio
- Eclipse IDE(Deprecated)

Android - Architecture

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.

Android - UI Design

In this chapter we will look at the different UI components of android screen. This chapter also covers the tips to make a better UI design and also explains how to design a UI.

UI screen components

A typical user interface of an android application consists of action bar and the application content area.

- Main Action Bar
- View Control
- Content Area
- Split Action Bar

These components have also been shown in the image below −

Understanding Screen Components

The basic unit of android application is the activity. A UI is defined in an xml file. During compilation, each element in the XML is compiled into equivalent Android GUI class with attributes represented by methods.

View and ViewGroups

An activity is consist of views. A view is just a widget that appears on the screen. It could be button e.t.c. One or more views can be grouped together into one GroupView. Example of ViewGroup includes layouts.

Types of layout

There are many types of layout. Some of which are listed below −

- Linear Layout
- Absolute Layout
- Table Layout
- Frame Layout
- Relative Layout

Linear Layout

Linear layout is further divided into horizontal and vertical layout. It means it can arrange views in a single column or in a single row. Here is the code of linear layout(vertical) that includes a text view.

```
<?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:layout_width=”fill_parent”
    android:layout_height=”wrap_content”
    android:text=”@string/hello” />
</LinearLayout>
```

AbsoluteLayout

The AbsoluteLayout enables you to specify the exact location of its children. It can be declared like this.

```
<AbsoluteLayout
  android:layout_width=”fill_parent”
  android:layout_height=”fill_parent”
  xmlns:android=”http://schemas.android.com/apk/res/android” >

  <Button
    android:layout_width=”188dp”
    android:layout_height=”wrap_content”
    android:text=”Button”
    android:layout_x=”126px”
    android:layout_y=”361px” />
</AbsoluteLayout>
```

TableLayout

The TableLayout groups views into rows and columns. It can be declared like this.

```
<TableLayout
  xmlns:android=”http://schemas.android.com/apk/res/android”
  android:layout_height=”fill_parent”
  android:layout_width=”fill_parent” >

  <TableRow>
    <TextView
      android:text=”User Name:”
      android:width =”120dp”
      />

    <EditText
      android:id=”@+id/txtUserName”
      android:width=”200dp” />
  </TableRow>

</TableLayout>
```

RelativeLayout

The RelativeLayout enables you to specify how child views are positioned relative to each other.It can be declared like this.

```
<RelativeLayout
  android:id=”@+id/RLayout”
  android:layout_width=”fill_parent”
  android:layout_height=”fill_parent”
  xmlns:android=”http://schemas.android.com/apk/res/android” >
</RelativeLayout>
```

FrameLayout

The FrameLayout is a placeholder on screen that you can use to display a single view. It can be declared like this.

```
<?xml version=”1.0” encoding=”utf-8”?>
<FrameLayout
  android:layout_width=”wrap_content”
  android:layout_height=”wrap_content”
  android:layout_alignLeft=”@+id/lblComments”
  android:layout_below=”@+id/lblComments”
```

```
  android:layout_centerHorizontal="true" >

  <ImageView
    android:src = "@drawable/droid"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
</FrameLayout>
```

Apart form these attributes, there are other attributes that are common in all views and ViewGroups. They are listed below −

Sr.No	View & description
1	**layout_width** Specifies the width of the View or ViewGroup
2	**layout_height** Specifies the height of the View or ViewGroup
3	**layout_marginTop** Specifies extra space on the top side of the View or ViewGroup
4	**layout_marginBottom** Specifies extra space on the bottom side of the View or ViewGroup
5	**layout_marginLeft** Specifies extra space on the left side of the View or ViewGroup

6	**layout_marginRight**	Specifies extra space on the right side of the View or ViewGroup
7	**layout_gravity**	Specifies how child Views are positioned
8	**layout_weight**	Specifies how much of the extra space in the layout should be allocated to the View

Units of Measurement

When you are specifying the size of an element on an Android UI, you should remember the following units of measurement.

Sr.No	Unit & description
1	**dp** Density-independent pixel. 1 dp is equivalent to one pixel on a 160 dpi screen.
2	**sp** Scale-independent pixel. This is similar to dp and is recommended for specifying font sizes
3	**pt** Point. A point is defined to be 1/72 of an inch, based on the physical screen size.

4	**px** Pixel. Corresponds to actual pixels on the screen

Screen Densities

Sr.No	Density & DPI
1	**Low density (ldpi)** 120 dpi
2	**Medium density (mdpi)** 160 dpi
3	**High density (hdpi)** 240 dpi
4	**Extra High density (xhdpi)** 320 dpi

Optimizing layouts

Here are some of the guidelines for creating efficient layouts.

- Avoid unnecessary nesting
- Avoid using too many Views
- Avoid deep nesting

Android Drawable Resources Tutorial

A drawable resource represents a graphic file or xml drawable file that can be drawn on screen. In Android, these files are stored in res/drawable folder. To prevent blurred images, drawable resources for different screen resolutions are provided in screen resolution specific drawable folders such as drawable-mdpi, drawable-hdpi, drawable-xhdpi and drawable-xxhdpi. Please see providing and accessing android resources for more information about alternate android resources.

Different types of drawables are bitmap, xml bitmap, nine-patch, layer list, state list, level list, transition drawable, clip drawable, scale drawable and shape drawable.

Bitmap

Bitmap files are .png, .jpg and .gif. The preferred bitmap file for android is .png. Bitmap file needs to be saved in res/drawable folder and can be accessed from xml and java code. For example if you save a bitmap file panda.png in your project, you can access from xml and java code as shown below.

```
<ImageButton android:id="@+id/panda"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:src="@drawable/panda"/>
((ImageButton)findViewById(R.id.panda)).setImageResource(R.drawable.panda);
```

Bitmap file can be read as Drawable object using Resources object.

```
Drawable panda = getResources().getDrawable(R.drawable.panda);
```

XML Bitmap

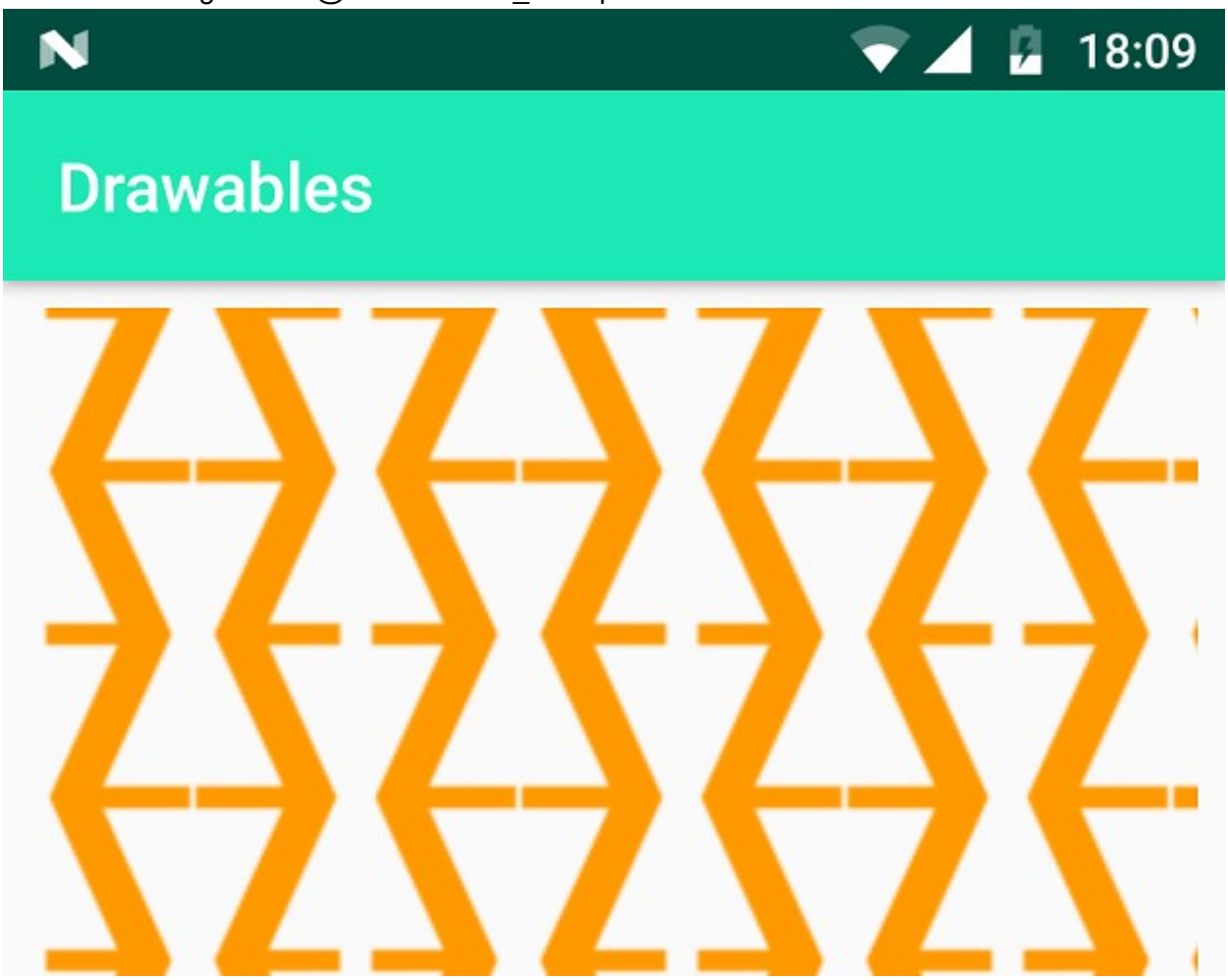
You can add properties to bitamp in xml and use it as drawable. Some properties which can be added to bitmap are dither, used when image and screen pixel don't match, filter, used to smooth bitmap appearance when it's stretched or shrunken, and tile mode, used to repeat the image. Below is an example xml bitmap with tileMode set to mirror.

If a bitmap is smaller than its container, you can specify the position of drawable in its container using gravity attribute. Gravity attribute values such as top, bottom, left, right and center won't scale bitmap if it is smaller than the container view.

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap xmlns:android="http://schemas.android.com/apk/res/android"
   android:src="@drawable/zoftino"
   android:tileMode="mirror" />
```

Used it as image view background.

```
<ImageView
   android:id="@+id/img"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:background="@drawable/xml_bitmap"/>
```



Nine-Patch File

Similar to bitmap, you can use nine-patch files and nine-patch xml. Please see create 9-patch file to know how to convert bitmap into nine-patch image file

Layer List

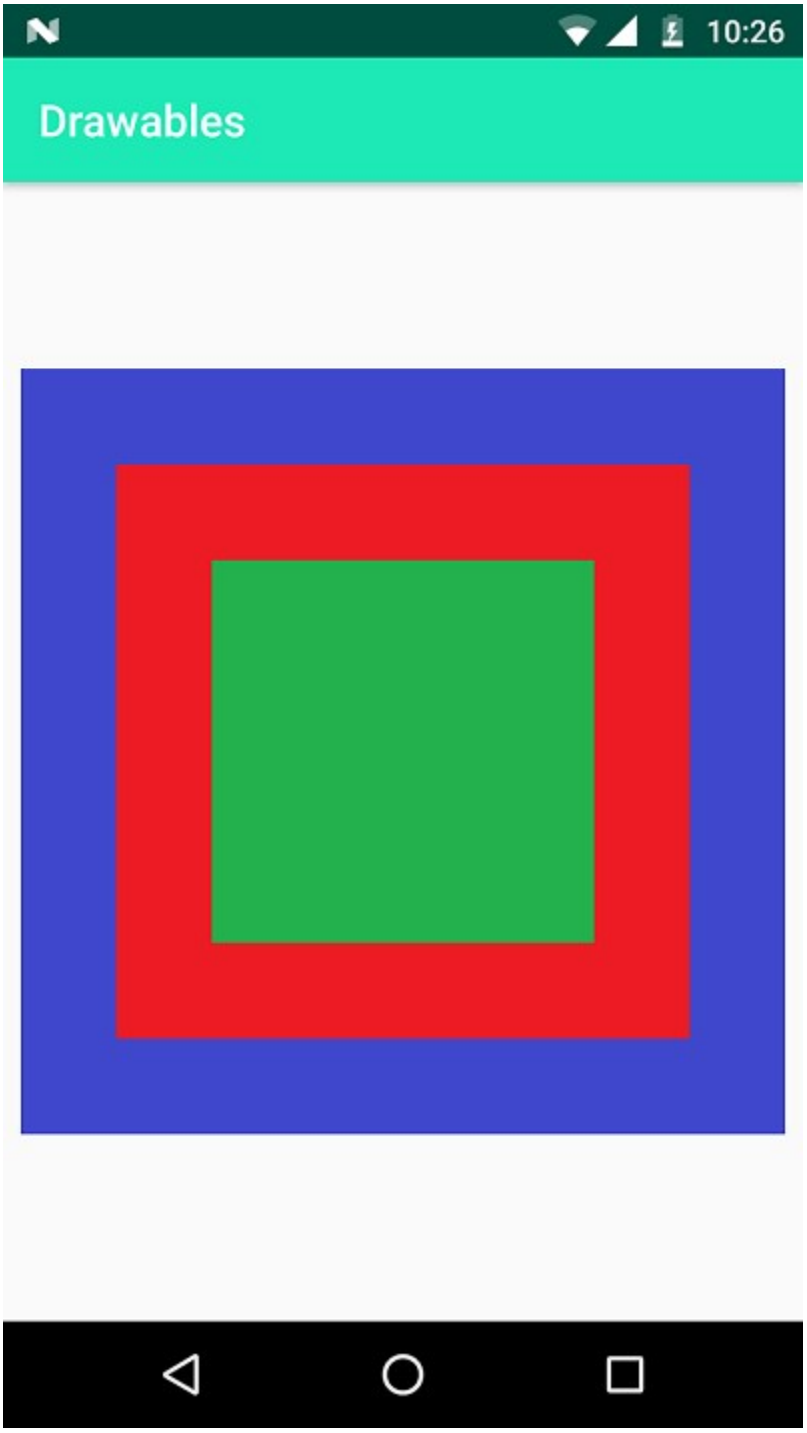
You can define a list of drawables in xml and use it as drawable resource in your android app. This drawable is called layer list drawable. Layer list drawable draws drawable items in the order they are defined in the xml and each drawable item is drawn on top of previous drawable item. The last drawable item is drawn on the top.

The root element of layer list drawable xml is layer-list element, you can define layers of drawable items using item element. Below is an example of layer list drawable.

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
   <item>
      <bitmap android:src="@drawable/square_blue"
         android:gravity="center" />
   </item>
   <item>
      <bitmap android:src="@drawable/square_red"
         android:gravity="center" />
   </item>
   <item>
      <bitmap android:src="@drawable/square_green"
         android:gravity="center" />
   </item>
</layer-list>
```

Layer list drawable is used as value of src attribute of ImageView.

```
<ImageView
   android:id="@+id/img"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:src="@drawable/layer_list"/>
```



State List

State list drawable allows you to define list of drawables for different states of a view. When state list drawable is set to one of the drawable attributes of a view, the attribute's value is derived from the state list drawable xml for the current state of the view. As view's state

changes, value for the attribute will change to the drawable defined in the state list drawable xml for the changed state.

Root element of state list drawable xml is selector. You define drawable item for each state using item element. View states are focused, selected, pressed, hovered, checkable, clicked, enabled and activated.

Below is an example of state list drawable xml which can be used in styles as value for button attribute for checkbox. Please see custom checkbox example for more information.

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:state_checked="true"
      android:drawable="@drawable/checked" />
   <item android:state_focused="true"
      android:drawable="@drawable/focused" />
   <item android:state_enabled="false"
      android:drawable="@drawable/diabled" />
   <item android:drawable="@drawable/unchecked" />
</selector>
```

Level List

Using level list drawable, drawable attributes of a view can be set to deferent drawables at run time by calling setLevel on Drawable object and passing level value to it. The level value points to a drawable in the level list drawable.

You can define level list drawable xml with different drawable items setting max and min values for each drawable. At run time, depending on the value passed to setLevel method, a drawable from the level drawable list will be picked and displayed.

Below example defines level list xml with three dawable items with different levels and level ranges. On clicking a button, level is incremented by calling setLevel on Drawable that will make it show corresponding drawable from the level list. Every time the button is clicked, image will be changed to reflect level value.

```
<?xml version="1.0" encoding="utf-8"?>
<level-list xmlns:android="http://schemas.android.com/apk/res/android" >
   <item
      android:drawable="@drawable/square_red"
      android:maxLevel="0" />
   <item
   android:drawable="@drawable/square_blue"
   android:maxLevel="1" />
```

```
    <item
        android:drawable="@drawable/square_green"
        android:minLevel="2"
        android:maxLevel="4"/>
</level-list>
```

Layout xml

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:orientation="vertical"
    android:layout_margin="8dp"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ImageView
        android:id="@+id/img"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/level_list"/>
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginRight="10dp"
        android:text="Button"/>
</LinearLayout>
```

Activity

```
public class LevelListActivity extends AppCompatActivity {
    private int levelInt = 0;
    private ImageView img;
    private Drawable drawable;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.level_list_activity);

        drawable = getResources().getDrawable(R.drawable.level_list);
        img = findViewById(R.id.img);

        findViewById(R.id.button).setOnClickListener(new View.OnClickListener() {
            @Override
```

```
        public void onClick(View view) {
            if(levelInt >= 4){
                levelInt = 0;
            }else{
                levelInt++;
            }
            drawable.setLevel(levelInt);
            img.setImageDrawable(drawable);
            img.refreshDrawableState();
        }
    });
  }
}
```

Transition Drawable

Transition drawable allows you to define two drawable items in xml and it transitions from first drawable to second drawable on calling startTransition method on it. You can specify transition time as an argument to startTransition method. To make it transition from second drawable item to first drawable, method reverseTransition needs to be called. In the below example, every time a button is clicked, image transition will occur.

```
<?xml version="1.0" encoding="utf-8"?>
<transition xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:drawable="@drawable/square_red" />
   <item android:drawable="@drawable/square_green" />
</transition>
img = findViewById(R.id.img);
findViewById(R.id.button).setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View view) {
      TransitionDrawable drawable = (TransitionDrawable) img.getDrawable();
      drawable.startTransition(1000);
   }
});
```

Inset Drawable

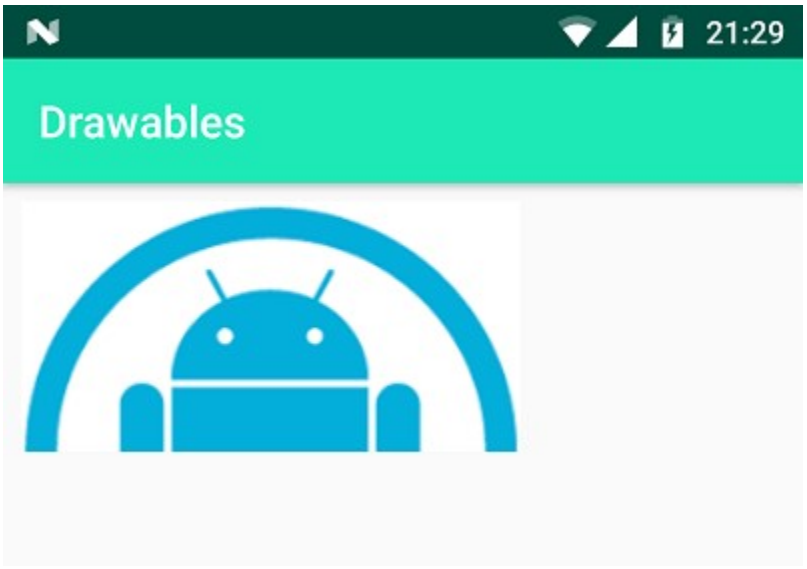
You can use inset drawable when the drawable is smaller than View to make it visible.

```
<?xml version="1.0" encoding="utf-8"?>
<inset xmlns:android="http://schemas.android.com/apk/res/android"
   android:drawable="@drawable/orange"
   android:insetTop="20dp"
```

```
    android:insetLeft="20dp" />
```

Clip Drawable

Clip drawable allows you to hide a drawable and slowly make it visible. By calling setLevel on drawable object and increasing the level will reveal the image slowly as the level increases. In the below example as button is clicked, image will be slowly visible.




```
<?xml version="1.0" encoding="utf-8"?>
<clip xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/android"
    android:clipOrientation="vertical"
    android:gravity="top" />
public class ClipDrawableActivity extends AppCompatActivity {
    private int levelInt = 0;
    private ImageView img;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.clip_drawable_activity);

        img = findViewById(R.id.img);
        findViewById(R.id.button).setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                ClipDrawable drawable = (ClipDrawable) img.getDrawable();
                levelInt = levelInt +500;
                drawable.setLevel(levelInt);
            }
```

```
        });
    }
}
```

Scale Drawable

Using scale drawable, you can specify how a drawable can be scaled depending on the level by setting scale height, scale width and scale gravity. Below is an example scale drawable which is displayed in image view. On clicking a button, the drawable's level will be increased by calling setLevel and passing level. As the level increases, the drawable that is displayed in ImageView will scale.

```
<?xml version="1.0" encoding="utf-8"?>
<scale xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/zoftino"
    android:scaleGravity="center_vertical|center_horizontal"
    android:scaleHeight="60%"
    android:scaleWidth="60%" />
```

```
        img = findViewById(R.id.img);
        findViewById(R.id.button).setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                ScaleDrawable drawable = (ScaleDrawable) img.getDrawable();
                levelInt = levelInt +800;
                drawable.setLevel(levelInt);
            }
        });
```

Shape Drawable

Shape drawable allows you to define different shapes in xml and the defined xml can be used as value of drawable attributes of views. You can define rectangle, square, oval and ring shapes using shape drawable. For more information on shape drawables, please see shape drawable examples.

Facebook Login Implementation for Android Apps

Android Multi Window Display Concepts & Configuration

Android App Links

Facebook Deep Linking in Android

Testing Android App on USB Connected Devices in Android Studio

Android User Interface Layouts

Android Constraint Layout

Adding Views & Constraints to Android Constraint Layout Programmatically

Android ScrollView Examples

RecyclerView and GridLayoutManager Example

RecyclerView ItemDecoration

Android Bottom Navigation View

Android Copy and Paste Text Example

Android Spinner Dropdown List

Android Spinner Custom Adapter & Layout

Android JSON Parsing

Handling JSON Using Gson in Android

Android OkHttp Example

Android OkHttp Cache Authenticator and Interceptor Example